Linear algebra for videogames

MATHEMATICS IMPLEMENTATION IN C++

Vito Domenico Tagliente
[NOME DELLA SOCIETÀ] | [INDIRIZZO DELLA SOCIETÀ]

Linear Algebra for videogames

VITO DOMENICO TAGLIENTE

Prefazione

Questo libro presenta una spiegazione degli elementi fondamentali dell'algebra lineare tramite un approccio orientato alla programmazione. Infatti, i vari concetti teorici saranno spiegati attraverso il formalismo matematico, arricchito di implementazione in codice. L'implementazione dei vari concetti matematici, quali vettori, matrici, consente di definire un approccio allo studio più concreto, permettendo il lettore di capire meglio gli argomenti trattati, permettendogli di svolgere diversi esercizi utilizzando le strutture dati che verranno definite attraverso i vari capitoli.

Lo scopo di questo libro è quello fornire una infarinatura generale sull'algebra lineare, senza entrare troppo nel dettaglio con informazioni che possono essere approfondite in manuali appositi. In particolare, verranno esaminati gli aspetti più importanti, strettamente correlati al contesto dei videogiochi. Le conoscenze acquisite, a temine volume, permetteranno il lettore di avere più consapevolezza sui concetti generali proposti da diversi motori grafici (Unity o Unreal Engine, per citarne alcuni).

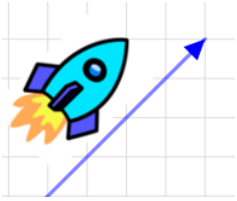
Note sull'autore

Il mio nome è Vito Domenico Tagliente. Mi sono laureato con lode alla facoltà di Ingegneria Informatica del politecnico di Bari, Italia. TODO.

Sommario

1. Vettori

In natura esistono grandezze determinate dal numero che le misura rispetto a una prefissata unità, come per esempio la lunghezza, l'area, il volume, il tempo. Queste grandezze sono dette scalari. Altre grandezze, come per esempio lo spostamento e la velocità, sono rappresentate da un numero, una direzione e un verso. Tali grandezze vengono chiamate grandezze vettoriali e vengono descritte mediante vettori.



Ad esempio, se vogliamo descrivere il movimento di una navicella spaziale, dire che questa si muove ad una certa velocità non è sufficiente. Affinché la descrizione sia completa, occorre specificare anche dove questa si sta muovendo, in particolare in quale direzione e con quale verso questa si sta spostando.

Nell'ambito dei videogiochi, i vettori sono ampiamente utilizzati per esprimere, nella considerazione più semplice possibile, gli spostamenti dei diversi oggetti che compongono la scena. Non solo, i vettori possono essere utilizzati per esprimere una direzione di osservazione applicata alle camere di gioco o semplicemente agli attori della scena che devono spostarsi e quindi ruotarsi e osservare lungo la direzione di spostamento. Lo stesso concetto può essere adoperato per indicare la direzione di sparo in contesti di giochi FPS. Gli scenari di applicazioni sono numerosi, l'importante è capire che i vettori devono essere utilizzati lì dove sono necessarie delle informazioni in più rispetto alla semplice grandezza numerica.

1.1 Notazione vettoriale

La notazione matriciale venne introdotta principalmente per esprimere le relazioni dell'algebra lineare in forma compatta, allo scopo di incrementarne la leggibilità. A prova di ciò, consideriamo un insieme di relazioni lineari definite tra un insieme $X = \{x_1, x_2, \dots, x_n\}$ di n elementi e un insieme $Y = \{y_1, y_2, \dots, y_m\}$ di m elementi.

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = y_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = y_2$$

$$\dots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = y_m$$

Il contenuto informativo della relazione può essere formalmente rappresentato dalla seguente notazione:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Se consideriamo di associare un nome ai diversi elementi che appaiono nell'equazione $\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \boldsymbol{A}$, $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \boldsymbol{x}$ e $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \boldsymbol{y}$, due oggetti matematici possono essere individuati:

- Il vettore a due dimensioni **A**, detto anche matrice
- I vettori a singola dimensione, detti vettori colonna o semplicemente vettori, quali **x** e **y**.

Applicando la nuova notazione, si ricava che la relazione può essere espressa in forma matriciale $\boldsymbol{Ax} = \boldsymbol{y}$.

Definiamo vettore, un insieme di n elementi che può essere espresso come:

$$\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$$

Questo oggetto è chiamato vettore colonna. In seguito, sarà chiaro il perché un vettore possa essere considerato come un caso speciale di matrice. Una matrice avente n righe ed una sola colonna.

Un vettore può anche essere espresso secondo la notazione $\boldsymbol{x} = [x_1\ x_2 \dots x_n]$, tale rappresentazione prende il nome di vettore riga. In generale si parla semplicemente di vettori. Se non viene specificato alcun qualificatore, si sottintende si tratti, per convenzione, di un vettore colonna.

I vettori sono rappresentati nei testi di algebra con la notazione $\vec{v}$, in realtà per questioni di comodità relative alla videoscrittura, i vettori vengono anche spesso identificati con una lettera minuscola in grassetto **x**.

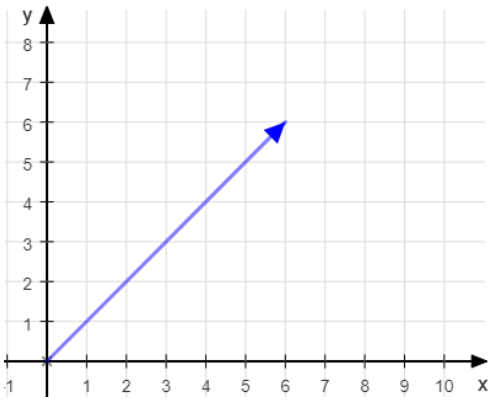
Gli elementi che costituiscono un vettore sono chiamati componenti, dove il simbolo n viene adoperato per indicare la quantità di componenti, l'ordine del vettore. Per esempio, definito il vettore $\boldsymbol{v} = [2 \quad 0]$, la prima componente è 2, la seconda è 0. L'ordine di **v** è 2, in quanto $n = 2$.

Molto importante è il concetto di modulo di un vettore, ovvero la lunghezza del vettore in relazione allo spazio euclideo in cui questo viene rappresentato. Formalmente, il modulo di un vettore, detto anche norma euclidea, si esprime con $||v|| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$. Nel caso in due dimensioni, si ricava $||v|| = \sqrt{x^2 + y^2}$.

1.2 Definizione geometrica di un vettore

Abbiamo discusso riguardo alla rappresentazione matematica di un vettore, esaminiamo la rappresentazione geometrica. Geometricamente, un vettore può essere rappresentato con un segmento orientato. Un segmento AB può essere percorso in due modi: da A verso B, oppure da B verso A. Nel primo caso, il segmento orientato verrà indicato con la notazione $\overrightarrow{AB}$; nel secondo caso con $\overrightarrow{BA}$. Tale segmento è caratterizzato da una lunghezza, da una direzione e da un verso. Ciò significa che i segmenti $\overrightarrow{AB}$ e $\overrightarrow{BA}$, che sono lo stesso insieme di punti, devono essere considerati diversi come segmenti orientati, perché differisce l'ordine con cui si considerano gli estremi.

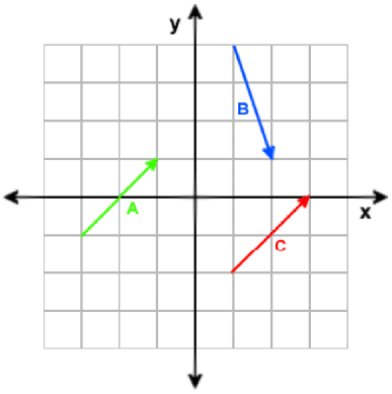
In figura è possibile notare la rappresentazione grafica di un vettore in uno spazio a due dimensioni, pertanto poggiato su un sistema cartesiano.



I vettori sono utilizzati per descrivere grandezze in base alla loro direzione e verso. La posizione non è specificata nella definizione del vettore stesso, piuttosto è relativa al punto in cui questo è applicato.

A livello di sistema cartesiano, ogni vettore è descritto dalle due coordinate del sistema, x e y. È da tenere a mente che tali coordinate sono necessarie al fine di identificare la pendenza della retta su cui giace il vettore, ma il punto di applicazione è irrilevante. Vettori aventi componenti identiche, possono essere rappresentati (paralleli tra loro) posti in posizioni differenti.

In figura, infatti, i vettori **a** e **c**, presentano stessa direzione, il che vuol dire che sono descritti dalle medesime componenti. La posizione è differente, questo perché tale informazione prescinde dalla notazione di vettore.



Pertanto, è bene distinguere il concetto di punto, inteso come concetto in grado di descrivere la posizione di un oggetto in un sistema di riferimento, rispetto al concetto di vettore, adatto a descrivere uno spostamento.

I vettori descrivono uno spostamento, sono quindi riferiti a posizioni relative. I punti, invece, descrivono la posizione assoluta di un oggetto. In realtà i due elementi, anche se concettualmente distinti, sono molto simili dal punto di vista matematico. Infatti, se consideriamo di applicare un vettore all'origine di un piano cartesiano. Dire che il vettore $\boldsymbol{v} = [x, y]$ si muove dall'origine alla posizione definita dal punto (x, y), non fa altro che mettere in relazione i due concetti espressi da un formalismo matematico simile.

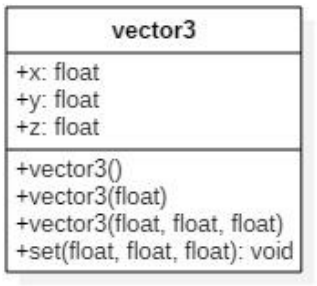
1.3 Stesura del codice

Spesso risulta molto complesso modellare concetti reali in codice, specialmente quando si è alle prime armi con la programmazione. In particolare, i concetti matematici sembrano presentare qualche difficoltà in più, sarà per la complessità degli argomenti o per il grado di astrazione richiesta dal compito.

La caratteristica chiave del C++ è il concetto di classe, inteso come tipo di dato definito dall'utente atto a rappresentare una entità più o meno complessa nel codice di un programma. L'utilizzo di classi permette non solo di modellare meglio le diverse entità che caratterizzano un programma, ma anche di organizzare meglio il codice, migliorando notevolmente la leggibilità dello stesso. Pertanto, possiamo considerare una classe come un contenitore logico di variabili, che forniscono la descrizione dell'oggetto, e di funzioni, che definiscono le modalità con cui quest'ultimo interagisce con il "mondo".

Consideriamo di creare il modello una persona. È possibile individuare diverse caratteristiche importanti ai fini della descrizione dell'entità stessa. Per esempio, si può decidere di caratterizzarla in base al nome, cognome, altezza, peso. Diventa fondamentale il contesto nel quale il modello viene sviluppato, un modello sarà dettagliato tanto quanto richiesto dal caso in esame. Ci possono essere casistiche in cui è necessario

specificare il colore degli occhi, altre in cui non lo è, per esempio. Da ciò si ricava che un modello è anche frutto della percezione e del bagaglio di conoscenza personale.



Modelliamo il concetto di vettore. Per semplificare le cose, consideriamo l'implementazione di un vettore di ordine 3, ovvero caratterizzato da tre componenti.

Individuate le caratteristiche descrittive dell'entità considerata, dette anche attributi, occorre definire le funzionalità, dette metodi, che permetteranno al contesto (il programma) di interagire con l'oggetto così definito.

È buona norma prendersi del tempo per pensare e modellare bene i concetti. Una volta identificate le caratteristiche che compongono e descrivono un oggetto, è possibile procedere con la stesura del codice.

```
class vector3
{
public:
    float x, y, z;

    vector3() {
        x = y = z = 0.0f;
    }

    vector3(float value) {
        x = y = z = value;
    }

    vector3(float _x, float _y, float _z) {
        x = _x;
        y = _y;
        z = _z;
    }

    vector3(const vector3& other) {
        x = other.x;
        y = other.y;
        z = other.z;
    }

    ~vector3() {}

    vector3& set(float _x, float _y, float _z) {
        x = _x;
        y = _y;
        z = _z;
        return (*this);
    }
}
```

Osservando il codice riportato, è possibile evidenziare diversi aspetti importanti:

- In C++, come in qualsiasi linguaggio di programmazione orientato agli oggetti, è possibile definire il livello di visibilità delle informazioni. Durante la definizione di una classe occorre specificare quali attributi e quali metodi apparterranno ad un contesto di accessibilità pubblico e quali privato. Le informazioni pubbliche sono visibili all'esterno, ogni entità del programma che ha una relazione stretta con

l'oggetto designato è in grado di interagirvi attraverso l'interfaccia pubblica. Le informazioni private, al contrario, hanno ambito di visibilità strettamente correlato al contesto di esistenza dell'oggetto stesso.

- È buona regola utilizzare il costruttore per definire le logiche di inizializzazione delle informazioni di un oggetto.
- Le informazioni di maggior rilievo vengono spesso assegnate ad un ambito di visibilità locale (privato). Questo permette di controllare e vigilare gli accessi e le modifiche di queste ultime, preservando la consistenza del contenuto informativo di tutto l'oggetto.

La classe vector3 possiede tre attributi chiamati rispettivamente x, y e z. Tali attributi definiscono la posizione dell'oggetto nello spazio, in questo caso in tre dimensioni. Si è deciso di quantificarle come float, ovvero numeri a virgola mobile. Tali scelta permette di rendere compatibile tale implementazione con le differenti API grafiche disponibili sul mercato. Questo perché un float, meglio di un intero, permette di specificare con precisione la posizione di un oggetto all'interno di una scena tridimensionale, per esempio.

La classe presenta diversi costruttori che permettono di inizializzare le componenti dell'oggetto. Occorre confrontare il formalismo matematico con quanto implementato.

Dal punto di vista del codice, il vettore così modellato non fa distinzione tra la rappresentazione come vettore riga e colonna. L'oggetto si occupa semplicemente di mantenere in memoria le informazioni sulle componenti.

Ricaviamo che la seguente notazione $\boldsymbol{v} = [2 \quad 0 \quad 1]$ si implementa in

```
vector3 v(2, 0, 1);
```

che risulta equivalente alla seguente forma

$$\boldsymbol{v} = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

che a livello di codice, cambiando l'indentazione può apparire più chiara

```
vector3 v(
    2,
    0,
    1
);
```

Caso particolare è il vettore nullo, ossia un vettore avente modulo 0. Si definisce vettore nullo, quel vettore le cui componenti sono tutte pari a zero, $\mathbf{0} = [0 \quad 0 \quad \cdots \quad 0]$. Per il vettore nullo non è possibile definire una direzione (e dunque nemmeno un verso), poiché il vettore nullo si riduce ad un punto, e per un punto passano infinite rette. A livello implementativo è possibile definire una funzione specifica per lo scopo:

```
static vector3 zero() {
    return vector3(0.0f);
}
```

In questo modo si può definire un vettore nullo anche utilizzando la funzione statica specificata, invece di utilizzare i costruttori messi a disposizione.

```
vector3 zero = vector3::zero();
```

Si definisce versore, quel vettore avente modulo unitario. I versori sono meglio indicati per specificare un verso ed una direzione, in quanto privi dell'informazione sul modulo. Dato un qualunque vettore **v** (diverso dal vettore nullo che è l'unico ad avere modulo pari a zero), è possibile individuarne il versore moltiplicandolo per il reciproco del suo modulo. Formalmente si ricava

$$\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{||\boldsymbol{v}||}$$

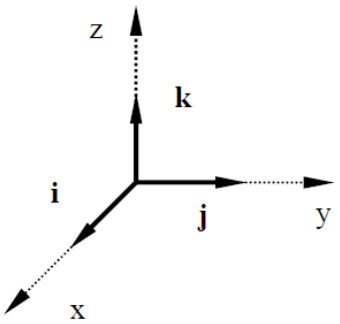
Tale operazione prende il nome di normalizzazione di un vettore, che può essere implementata quanto segue.

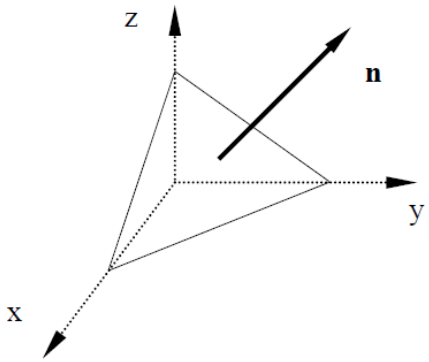
```
float magnitude() const {
    return sqrt(x*x + y*y + z*z);
}

vector3& normalize() {
    return (*this *= (1.0f / magnitude()));
}
```

Si ricava che

```
vector3 vNormalized = v.normalize();
```

Si denota con $\boldsymbol{e}_i$, quel versore le cui componenti sono tutte zero all'infuori della i-esima, pari a uno. I versori corrispondenti agli assi cartesiani, in uno spazio tridimensionale, spesso sono indicati come **i**,**j**,**k**. Infine, si denota con **n** il versore avente direzione ortogonale a un generico piano π.





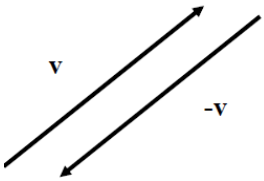
In uno spazio a tre dimensioni, come riportato in figura in alto

- $\boldsymbol{i} = [1 \quad 0 \quad 0]$
- $\boldsymbol{j} = [0 \quad 1 \quad 0]$
- $\boldsymbol{k} = [0 \quad 0 \quad 1]$

```
vector3 i(1, 0, 0);
vector3 j(0, 1, 0);
vector3 k(0, 0, 1);
```

Definito un vettore **v**, si chiama vettore opposto quel vettore **-v** che possiede stesso modulo, stessa direzione, ma verso contrario a **v**.

Per esempio, dato $\boldsymbol{v} = [2 \quad 0 \quad 1]$, l'opposto è $\text{-}\boldsymbol{v} = [-2 \quad 0 \quad -1]$.




```
vector3 operator-() const {
    return vector3(-x, -y, -z);
}
```

È da notare che per raggiungere tale risultato, è stato sovrascritto l'operatore -, che ci permette di avere la stessa notazione matematica anche a livello di codice:

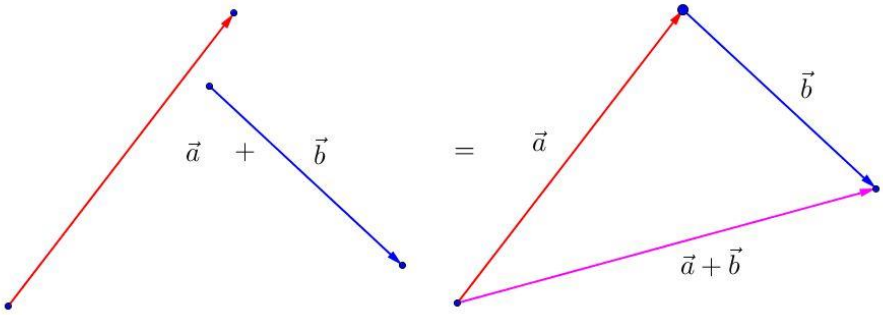
```
vector3 v(2, 0, 1);
vector3 v1 = -v;
```

1.4 Operazioni con i vettori

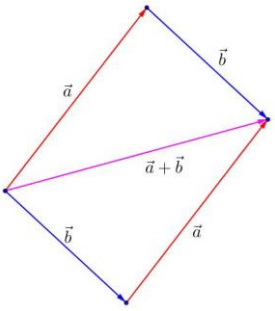
Come tra i numeri (detti in questo contesto scalari), tra i vettori si possono effettuare delle operazioni, come la somma e il prodotto per scalare.

Esistono principalmente due metodi per calcolare la somma tra due vettori: il metodo punta-coda e il metodo del parallelogramma, che sono tra loro equivalenti. La somma di due vettori sarà sempre comunque un vettore. Definiti due vettori, **a** e **b**

- Il metodo punta-coda, tenendo fissato un vettore dei due (in questo caso **a**), trasla l'altro (in questo caso **b**) mantenendolo parallelo a se stesso, in modo tale che il suo punto di applicazione coincida con la punta del primo vettore. Il vettore somma $\boldsymbol{a} + \boldsymbol{b}$ è il vettore che congiunge la "coda" del primo vettore (il punto di applicazione), con la "punta" del secondo.



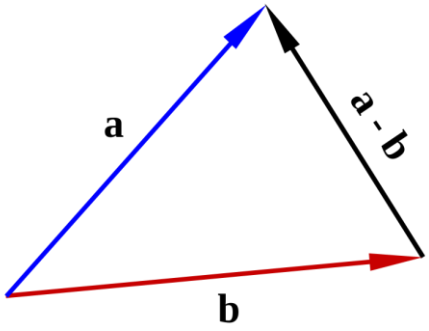
- Il metodo del parallelogramma consiste nel costruire un parallelogramma di lati i vettori dati. La somma dei due vettori è la diagonale del parallelogramma che parte dal punto di applicazione di uno dei due, e arriva alla punta dell'altro.



Dal punto di visto algebrico, possiamo sommare e sottrarre due vettori aventi stessa dimensione. La somma di due vettori consiste nella somma delle corrispettive componenti

$$\boldsymbol{v} + \boldsymbol{w} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} + \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} v_1 + w_1 \\ \vdots \\ v_n + w_n \end{bmatrix}$$

La somma tra due vettori è un'operazione associativa, commutativa e possiede l'elemento neutro che è il vettore nullo. La sottrazione può essere interpretata come $\boldsymbol{v} - \boldsymbol{w} = \boldsymbol{v} + (-\boldsymbol{w})$.



La sottrazione, al contrario dell'addizione, non è commutativa. Infatti, se $\boldsymbol{v} + \boldsymbol{w} = \boldsymbol{w} + \boldsymbol{v}$, lo stesso non vale per la sottrazione $\boldsymbol{v} - \boldsymbol{w} = -(\boldsymbol{w} - \boldsymbol{v})$.

1.4.1 Overloading degli operatori

Per implementare la seguente operazione anche a livello di codice, occorre definire il concetto di overloading degli operatori. Per overloading di operatori si intende la capacità di un linguaggio di programmazione di specificare quale logica comportamentale utilizzare nel momento in cui si applicano gli operatori matematici a tipi semplici e/o specificati dall'utente.

L'utilizzo di una notazione concisa per le operazioni di uso comune è di importanza fondamentale. In tutti i linguaggi, di programmazione e non, gli operatori sono dei simboli convenzionali che rendono più agevole la presentazione e lo sviluppo di concetti di uso frequente. Per esempio, la notazione $a + b * c$ risulta molto più agevole della frase "moltiplica b per c e aggiungi il risultato ad a".

Il C++ supporta un insieme di operazioni per i suoi tipi nativi. Tuttavia, la maggior parte dei concetti utilizzati comunemente non sono facilmente rappresentabili per mezzo di tipi nativi, e bisogna spesso ricorrere a tipi complessi. Per esempio, i numeri complessi, i vettori, le matrici, sono tutte entità che meglio si prestano a essere modellate e rappresentate mediante le classi. In questo caso risulta molto più agevole e coinciso ridefinire il significato delle operazioni nell'ambito di questi tipi definiti dall'utente, in alternativa alla chiamata di funzioni. Si ricava $a + b * c$ rispetto a $add(a, multiply(b, c))$

Per ottenere l'overload di un operatore in C++, occorre creare una funzione il cui nome deve essere costituito dalla parola chiave operator, seguita dal simbolo dell'operatore che si vuole ridefinire. Gli argomenti dell'operatore devono corrispondere agli operandi dello stesso. Ne consegue che per gli operatori unari è necessario specificare un solo argomento, per quelli binari ne occorrono due.

Occorre sottolineare che non è possibile "inventare" nuovi simboli, ma si possono utilizzare solo quelli predefiniti. Inoltre, le regole di precedenza e associatività restano legate al simbolo e non al suo significato. Per esempio, un operatore in overload associato al simbolo *, non può mai essere definito unario e ha sempre la precedenza sull'operatore associato al simbolo +, qualunque sia il significato di entrambi.

Tornando alle operazioni tra vettori, vediamo come è possibile definire l'operatore di somma fra due vettori.

```
vector3 operator+(const vector3& v) const {
    return vector3(x + v.x, y + v.y, z + v.z);
}
```

Con la seguente definizione, diventa possibile definire l'espressione `s = v + w;` anziché chiamare una ipotetica funzione `vector3 s = add(v, w);`

Notare:

- La funzione ha un valore di ritorno di tipo vector3
- Gli argomenti sono passati per riferimento e dichiarati costanti, questa pratica è comune in C++ quando si vuole fornire delle informazioni che non devono essere modificate dal corpo della funzione chiamata.

Abbiamo visto come ridefinire il comportamento di un operatore, ma non ci siamo occupati di specificare dove tali funzioni devono essere scritte. In particolare, quando queste devono accedere a informazioni private della classe, è opportuno ridefinire gli operatori tramite metodi pubblici della classe stessa, oppure attraverso delle funzioni friend.

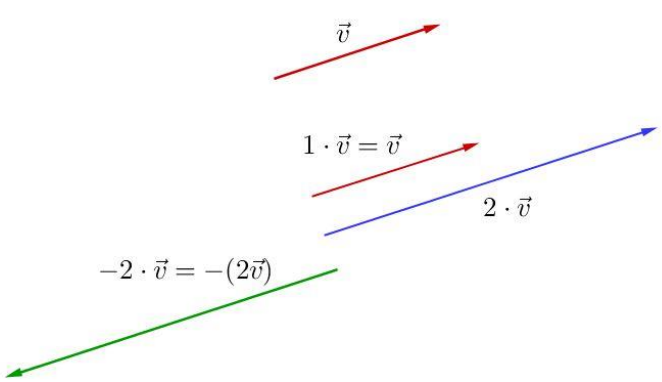
In generale, quando il primo operando è oggetto della classe o la funzione lo restituisce come lvalue, è buona regola definire l'overloading dell'operatore come metodo della classe stessa. La miglior progettazione degli operatori di una classe consiste nell'individuare un insieme ben definito di metodi per le operazioni che si applicano su un unico oggetto o che modificano il loro primo operando. Usare funzioni esterne (o friend) per le altre.

Nel primo caso, risulta importante utilizzare l'operatore this. Il this punta allo stesso oggetto della classe definita, in cui il metodo è incapsulato e viene automaticamente inserito dal C++. Si ricava che nel momento in cui una ridefinizione di operatore viene descritta come metodo della classe stessa, deve essere ridotto di una unità il numero di argomenti rispetto al numero di operandi, in quanto il primo operando è l'oggetto stesso. Si ricava che se l'operatore è binario, la funzione non deve avere argomenti, se l'operatore è binario, la funzione è caratterizzata da u solo argomento.

Se il risultato dell'operazione è l'oggetto stesso, l'istruzione di ritorno sarà caratterizzata dall'espressione `return *this`, utilizzata per accedere al valore dell'oggetto corrente.

1.4.2 Prodotto per scalare

Non si può sommare un vettore con uno scalare, essendo queste grandezze totalmente differenti. Piuttosto, è posssibile moltiplicare un vettore per uno scalare. Il risultato sarà un vettore $\boldsymbol{v}' = \boldsymbol{v} * s$, parallelo al vettore $\boldsymbol{v}$, avente lunghezza differente e possibilmente verso opposto, dipendentemente al segno di s.



Definito s uno scalare, si ricava:

$$s\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} s = \begin{bmatrix} sv_1 \\ \vdots \\ sv_n \end{bmatrix}$$

A livello di implementazione l'operazione $\boldsymbol{v} * s$, può essere implementata come metodo della classe vector3, in quanto il l'oggetto stesso è primo operando dell'operazione.

```
vector3 operator*(const float s) const {
    return vector3(x * s, y * s, z * s);
}
```

Nel caso commutativo $s * \boldsymbol{v}$, occorre definire una funzione esterna alla classe, perché l'oggetto è secondo operando dell'operazione.

```
inline vector3 operator*(const float s, const vector3& v) {
    return vector3(v.x * s, v.y * s, v.z * s);
}
```

1.4.3 Esempi

In questo capitolo viene mostrata l'equivalenza tra la notazione matematica classica e l'implementazione realizzata. In particolare, è fondamentale porre l'attenzione sulla comodità e la leggibilità ricavata dall'applicazione dell'overloading dei vari operatori.

Questo capitolo, inoltre, permette di approfondire anche a livello numerico e pratico i concetti esaminati, al fine di verificare quanto acquisito.

Definiti i vettori $\boldsymbol{a} = [3 \quad 0 \quad 1]$ e $\boldsymbol{b} = [-2 \quad 5 \quad 3]$

```
vector3 a(3, 0, 1);
vector3 b(-2, 5, 3);
```

La somma dei due vettori sarà data da:

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 + (-2) \\ 0 + 5 \\ 1 + 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 4 \end{bmatrix}$$

È da notare che anche a livello di codice viene rispettata la stessa notazione:

```
vector3 c = a + b;
```

La differenza dei due vettori, rispettivamente sarà data da:

$$\boldsymbol{d} = \boldsymbol{a} + \boldsymbol{b} = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} -2 \\ 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 - (-2) \\ 0 - 5 \\ 1 - 3 \end{bmatrix} = \begin{bmatrix} 7 \\ -5 \\ -2 \end{bmatrix}$$

```
vector3 d = a - b;
```

Allo stesso modo, è possibile moltiplicare un vettore per uno scalare k. Definito $k = 2$, si ricava:

$$\boldsymbol{c} = \boldsymbol{a}k = \boldsymbol{a} * 2 = \begin{bmatrix}3\\0\\1\end{bmatrix} k = \begin{bmatrix}3*k\\0*k\\1*k\end{bmatrix} = \begin{bmatrix}3*2\\0*2\\1*2\end{bmatrix} = \begin{bmatrix}6\\0\\2\end{bmatrix} = k * \boldsymbol{a} = 2\begin{bmatrix}3\\0\\1\end{bmatrix}$$

```
vector3 c = a * 2;
vector3 c = 2 * a;
```

1.5 Moltiplicazione tra vettori

Abbiamo visto come è possibile moltiplicare un vettore per uno scalare, ma possiamo moltiplicare anche vettori tra loro. Esistono due tipologie di prodotto tra vettori, quello scalare e quello vettoriale.

1.5.1 Prodotto scalare

Il prodotto scalare, in algebra lineare, si esprime quanto segue

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

È possibile esplicitare l'operazione appena definita per i casi specifici nelle 2 e 3 dimensioni

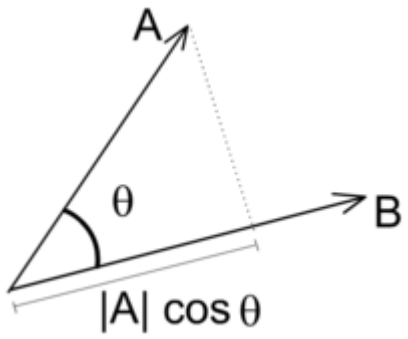
- $\boldsymbol{a} \cdot \boldsymbol{b} = a_x b_x + a_y b_y$, in due dimensioni
- $\boldsymbol{a} \cdot \boldsymbol{b} = a_x b_x + a_y b_y + a_z b_z$, in tre dimensioni

Il prodotto scalare è commutativo, ciò vuol dire che vale la relazione $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{b} \cdot \boldsymbol{a}$.

```
float dot(const vector3& v) const {
    return (*this)*v;
}

static float dot(const vector3& v1, const vector3& v2) {
    return v1*v2;
}

float operator*(const vector3& v) const {
    return (x*v.x + y*v.y + z*v.z);
}
```

Esaminiamo l'interpretazione geometrica del prodotto scalare. Considerati due vettori ***a*** e ***b***, applicati nello stesso punto. Il prodotto scalare si definisce come

$$\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| \cos\theta$$

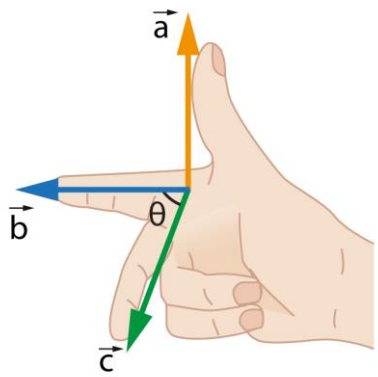


A livello geometrico, dato che il prodotto cartesiano dipende dal coseno di theta (l'angolo comrpeso tra i due vettori), si ricava che se il prodotto scalare di due vettori è pari a zero, allora tali vettori sono ortogonali. Se theta è acuto, il prodotto scalare sarà una quantità positiva. Se theta è ottuso, il prodotto scalare risulterà negativo.

È da notare che il risultato di un prodotto scalare è una quantità, non un vettore.

1.5.2 Prodotto vettoriale

Per prodotto vettoriale si intende un'operazione tra due vettori che avviene nello spazio tridimensionale, il cui risultato è un vettore.



Dati due vettori $\boldsymbol{a}$ e $\boldsymbol{b}$, il loro prodotto vettoriale, indicato con $\boldsymbol{a} \times \boldsymbol{b}$, è un vettore che ha:

- Direzione perpendicolare al piano che contiene i due vettori $\boldsymbol{a}$ e $\boldsymbol{b}$
- Verso dato dalla regola della mano destra, secondo cui se si pone il pollice nel verso del vettore $\boldsymbol{a}$ e l'indice nel verso di $\boldsymbol{b}$, il vettore $\boldsymbol{a} \times \boldsymbol{b}$ è uscente dal palmo della mano.
- Modulo pari all'area del parallelogramma generato dai due vettori $\boldsymbol{a}$ e $\boldsymbol{b}$

Formalmente il prodotto vettoriale si descrive come

$$\boldsymbol{a} \times \boldsymbol{b} = \boldsymbol{n}|\boldsymbol{a}||\boldsymbol{b}| \sin\theta$$

Dove $0 < \theta < \pi$ è l'angolo compreso tra $\boldsymbol{a}$ e $\boldsymbol{b}$, $\boldsymbol{n}$ è un versore normale al piano formato dai due vettori, che fornisce la direzione del prodotto vettoriale. Si nota che $|\boldsymbol{a}||\boldsymbol{b}| \sin\theta$ rappresenta l'area del parallelogramma individuati dai due vettori.

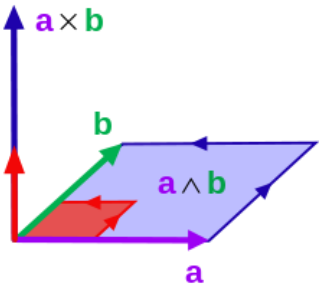
Esplicitamente, definiti $\boldsymbol{i}$,$\boldsymbol{j}$ e $\boldsymbol{k}$ i versori di una base ortonormale di R^3, il prodotto di $\boldsymbol{a} = (a_x, a_y, a_z)$ e $\boldsymbol{b} = (b_x, b_y, b_z)$ può essere scritto come il determinante di una matrice (le matrici verranno esaminate nel dettaglio nel capitolo 2):

$$\boldsymbol{a} \times \boldsymbol{b} = det\begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix} = (a_y b_z - a_z b_y)\boldsymbol{i} - (a_x b_z - a_z b_x)\boldsymbol{j} + (a_x b_y - a_y b_x)\boldsymbol{k} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_x b_z - a_z b_x \\ a_x b_y - a_y b_x \end{bmatrix}$$

A livello implementativo, poiché in ambito delle applicazioni multimediali quali i videogiochi le prestazioni sono una caratteristica chiave da tenere sempre ben in considerazione, è opportuno esplicitare la formula risultante, permettendo al calcolatore di trascurare calcoli e operazioni non necessarie.

```
vector3 cross(const vector3& v) const {
    return vector3(
        y*v.z - z*v.y,
        z*v.x - x*v.z,
        x*v.y - y*v.x
    );
}

static vector3 cross(const vector3& v1, const vector3& v2) {
    return vector3(
        v1.y*v2.z - v1.z*v2.y,
        v1.z*v2.x - v1.x*v2.z,
        v1.x*v2.y - v1.y*v2.x
    );
}
```

Dal punto di vista geometrico, ricaviamo che il prodotto vettoriale permette di conoscere la natura dei vettori partecipanti nel prodotto. Infatti, se i due sono paralleli tra loro, il prodotto vettoriale sarà nullo, in quanto dipendente dal $\sin\theta$.



Un'ultima considerazione. Il prodotto vettoriale è definito solo nello spazio tridimensionale, trattasi di un caso particolare di semplificazione del prodotto esterno che vale per vettori in spazi ad n dimensioni.

Il prodotto esterno di due vettori è un bivettore, cioè un elemento di piano orientato. Dati due vettori $\boldsymbol{a}$ e $\boldsymbol{b}$, il bivettore $\boldsymbol{a}\wedge\boldsymbol{b}$ è il parallelogramma orientato formato dai due vettori.

Con questo piccolo accenno al prodotto esterno, sorvoliamo l'argomento in quanto non è indispensabile per la trattazione matematica di un contesto di applicazione ai videogiochi, strettamente legato al mondo tridimensionale. Per approfondire l'argomento, si consiglia di consultare un testo di algebra lineare adatto.

1.5.3 Esempi

In questo capitolo, sono stati esaminate le due operazioni più importanti tra vettori. Tali operazioni, il prodotto scalare e vettoriale, sono fondamentali in quanto permettono di identificare la direzione reciproca dei due. Come visto, il prodotto scalare permette di capire se due vettori sono ortogonali, mentre il prodotto vettoriale permette di individuare se due vettori sono paralleli tra loro.

Trattasi però di operazioni molto macchinose e soggette ad errori di calcolo, se fatte manualmente; al contrario vediamo che con l'implementazione provvista il risultato è immediato. Occorre tenere a mente solamente il concetto e capire bene quando e come usare le informazioni acquisite. In particolare, il prodotto scalare servirà nel capitolo seguente, dove andremo ad introdurre le matrici e tutte le operazioni ad esse annesse.

Definiti i vettori $\boldsymbol{a} = [3 \quad 0 \quad 1]$ e $\boldsymbol{b} = [-2 \quad 5 \quad 3]$, abbiamo visto come il prodotto scalare dei due sia dato dalla formula

$$\boldsymbol{a}\cdot\boldsymbol{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = 3*(-2) + 0*5 + 1*3 = -6 + 0 + 3 = -2$$

```
vector3 a(3, 0, 1);
vector3 b(-2, 5, 3);
```

Il prodotto scalare può essere ricavato con uno tra le seguenti possibilità

- `float p = a * b;`
- `p = a.dot(b);`
- `p = vector3::dot(a, b);`

Il prodotto vettoriale, invece, sarà dato da

$$\boldsymbol{a} \times \boldsymbol{b} = det\begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix} = (a_y b_z - a_z b_y)\boldsymbol{i} - (a_x b_z - a_z b_x)\boldsymbol{j} + (a_x b_y - a_y b_x)\boldsymbol{k} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_x b_z - a_z b_x \\ a_x b_y - a_y b_x \end{bmatrix} =$$

$$= \begin{bmatrix} 0 * 3 - 1 * 5 \\ 3 * 3 - 1 * (-2) \\ 3 * 5 - 0 * (-2) \end{bmatrix} = \begin{bmatrix} 3 - 5 \\ 9 + 2 \\ 15 - 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 11 \\ 15 \end{bmatrix}$$

Anche questo risultato, a livello implementativo, può essere raggiunto secondo diverse modalità:

```
vector3 cross = a.cross(b);
cross = vector3::cross(a, b);
```

2. Matrici