Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Magistrale in Informatica

SVILUPPO DI UN SIMULATORE PER LA PROPAGAZIONE DEL CALORE NEI DATA CENTER

Tesi di Laurea in Algoritmi e Strutture dati

Relatore:
Dott.
Moreno Marzolla

Presentata da:
Diego Rodriguez

Sessione I
Anno Accademico 2011-2012

Questa tesi è dedicata a mia nonna, che ha sempre supportato il mio studio e la mia passione per l'informatica.

Introduzione

La simulazione consente, tramite una rappresentazione virtuale di un'ipotetica situazione reale, di valutare a priori gli effetti e l'evoluzione di un sistema preso in esame. Questa tesi ha come obiettivo principale la creazione di un software con cui è possibile riprodurre scenari reali (in particolare i data center) in cui è di vitale importanza determinare se la stabilità del sistema può essere alterata dal calore presente nell'ambiente.

Nel campo della Computational Fluid Dynamics (CFD) ossia la fluidodinamica computazionale, la simulazione permette di analizzare scenari la cui realizzazione sarebbe troppo costosa. Si pensi ad esempio alla valutazione dei costi per l'impianto di raffreddamento in un data center, in questo caso sarebbe veramente utile uno strumento che permetta di capire se la disposizione degli elementi nell'ambiente garantisca che non ci siano zone a temperatura troppo elevata. Altri scenari potrebbero considerare le reazioni tra componenti elettronici di un circuito, il movimento del gas dentro un oleodotto, l'iniezione di benzina all'interno di un motore ecc ....

La CFD in sostanza permette di simulare al calcolatore l'evoluzione di un fluido con determinate proprietà (temperatura, viscosità, velocità ...) in tutti quei casi in cui è possibile trasformare la realtà continua in una sua rappresentazione discreta. Bisogna specificare però che quasi sempre il costo computazionale della simulazione è fortemente influenzato dalla dimensione del problema e in particolare dal livello di dettaglio considerato per la discretizzazione dell'ambiente reale.

L'organizzazione dei capitoli di questa tesi è la seguente: il primo capitolo

illustra la composizione tipica di un data center delineando tutti gli elementi appartenenti a tale scenario comprendendo anche il fluido e le proprietà studiate legate ad esso; il secondo capitolo contiene i concetti, le assunzioni e le nozioni teoriche senza le quali non sarebbe stato possibile creare il simulatore; il terzo capitolo è relativo allo sviluppo dell'applicazione e descrive quindi l'architettura adottata e gli aspetti implementativi più importanti; il quarto capitolo analizza i risultati sperimentali ottenuti valutando l'efficienza e la correttezza del software prodotto. Infine si forniscono dei possibili sviluppi futuri che potrebbero arricchire le funzionalità e migliorare la precisione del simulatore.

Indice

Elenco delle figure

Capitolo 1

Scenario

In questa tesi si è interessati a simulare il trasporto di calore in un data center anche se è possibile considerare scenari di applicazione generali per studiare le meccaniche di interazione del fluido.

1.1 Data Center

Il termine data center indica un ambiente (spesso una stanza) contenente diverse unità server organizzate in armadietti e solitamente collegati tra loro. Questi calcolatori vengono poi utilizzati per fornire servizi di connettività come hosting di siti web, cloud computing, macchine virtuali.

Solitamente un data center è sempre attivo e questo comporta un'alimentazione costante garantita tramite gruppi di continuità e la presenza di impianti di condizionamento per raffreddare l'ambiente. Questi requisiti ovviamente vengono accompagnati da una serie di costi che sarebbe opportuno minimizzare, non si esclude infatti che il simulatore possa aiutare a stimare il numero di unità di condizionamento, il numero di server e il fabbisogno energetico richiesto dall'intero sistema.

1.1.1 Rack

All'interno di un data center i server sono organizzati in armadietti chiamati Rack. Questi contenitori possono ospitare da un minimo di dodici fino ad un massimo di quarantadue unità opportunamente organizzate in box accessibili tramite un meccanismo a scorrimento.

In realtà il termine Rack viene anche utilizzato come unità di misura per la dimensione standard dei server pari a $larghezza \times lunghezza \times altezza = 482.6 \times 600 \times 44.45\ mm$.

Infine bisogna aggiungere che solitamente i Rack sono chiusi lateralmente e superiormente ma possono presentare delle aperture nella zona inferiore, posteriore e frontale (parzialmente) per permettere all'aria di fluire all'interno e raffreddare i componenti elettronici.

1.1.2 Impianto di condizionamento

I sistemi attualmente in commercio spaziano da soluzioni basate sulla dissipazione tramite condizionamento d'aria opportunamente indirizzata nelle aree calde a soluzioni che sfruttano circuiti idraulici per raffreddamento a liquido (a volte anche sistemi ibridi tra le due categorie).

Gli impianti che sfruttano il condizionamento sono comunemente chiamati CRAC, acronimo di Computer Room Air Conditioner ossia condizionatore d'aria per una stanza di computer. Questo sistema è composto da una serie di condizionatori che hanno il compito di risucchiare l'aria calda, raffreddarla e pomparla nel pavimento dove seguendo i condotti di aerazione fuoriesce dalle grate poste nelle vicinanze dei Rack. Un esempio di disposizione è riportato nella figura 1.1 dove è possibile vedere la formazione di due *corridoi* di aria calda e fredda tra CRAC e armadietti.

Le figure 1.2 e 1.3 rappresentano invece due esempi di sistemi per convogliare l'aria in maniera più efficiente: la prima prevedere l'aggiunta di condotti di areazione sulla sommità degli armadietti e dei CRAC in modo da creare un flusso d'aria circolare tra il soffitto e il pavimento; la seconda in-

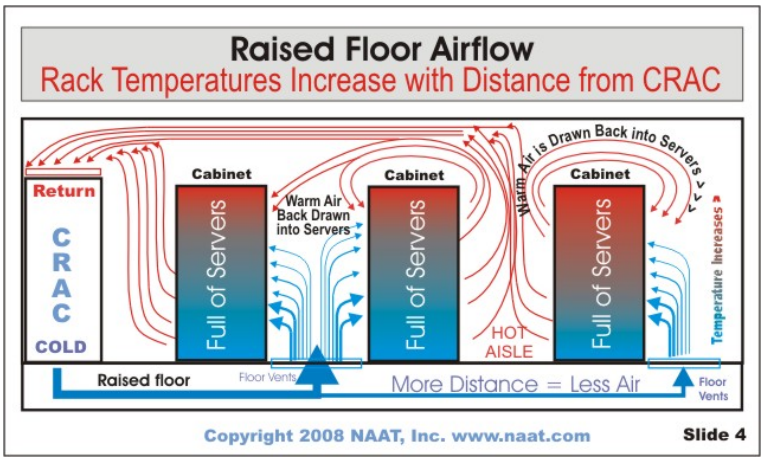

Figura 1.1: Visione laterale di un impianto di condizionamento CRAC (notare le grate poste sul pavimento), NAAT Inc. 2008.

vece prevede una struttura di contenimento che evita che l'aria fredda venga dispersa lateralmente.

Gli impianti a liquido rappresentano invece un sistema di refrigeramento più innovativo. Solitamente si parla di raffreddamento ravvicinato perché di solito tra coppie di armadietti Rack vengono posti dei circuiti idraulici che tramite il liquido (ad esempio acqua) dissipano il calore generato dai server. La figura 1.4 infatti mostra due armadietti a cui è stata aggiunta una terza unità contenente il sistema idraulico dove i tubi contenenti il liquido refrigerante vengono fatti passare tra i rack in modo da creare un vero e proprio circuito di raffreddamento.

1.2 Fluido

La definizione di fluido:

> Una sostanza che si deforma illimitatamente (fluisce) se sottoposta a uno sforzo di taglio, indipendentemente dall'entità di quest'ultimo; è un particolare stato della materia che compren-

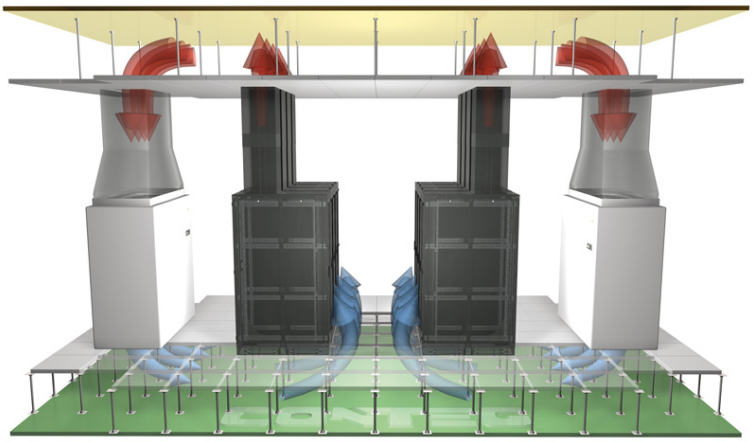

Figura 1.2: Un impianto di condizionamento CRAC modificato (notare i condotti posti sulla cima dei Rack e CRAC), Conteg.

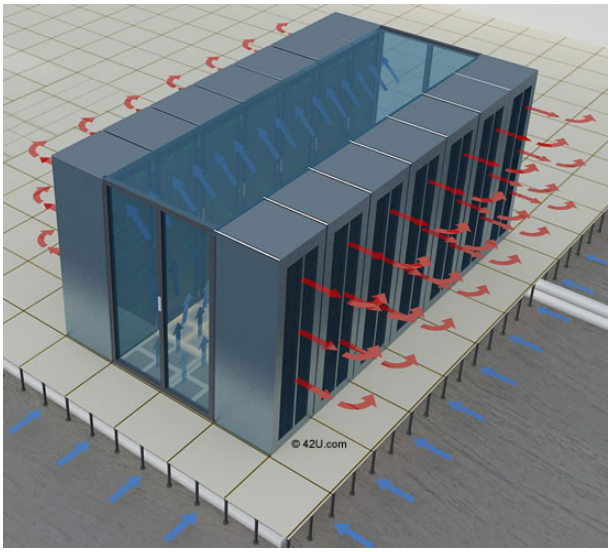

Figura 1.3: I Rack possono essere raggruppati per contenere meglio l'aria fredda, 42U.com .

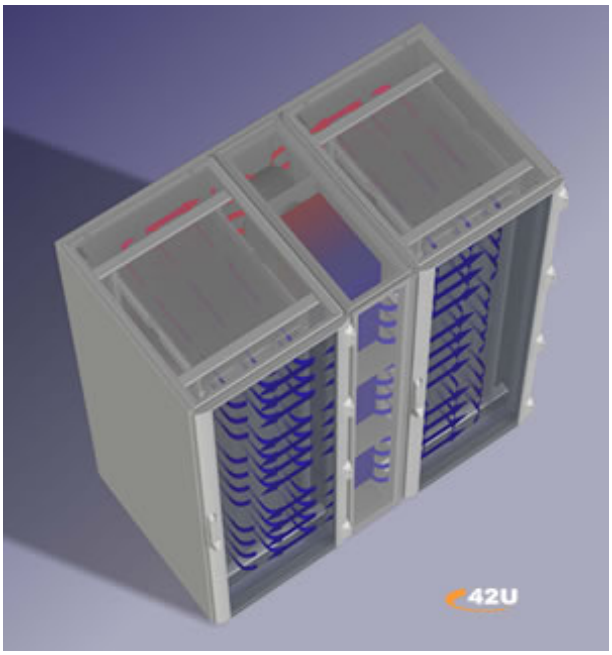

Figura 1.4: Il sistema di raffreddamento a liquido costituisce un vero e proprio micro clima neutrale al resto della stanza, 42U.com .

de i liquidi, i gas, il plasma e, in taluni casi, i solidi plastici (Wikipedia).

In questo scenario si considera una normale stanza contenente aria a temperatura ambiente e pressione di una atmosfera. è difficile far riferimento ad altri tipi di fluidi perché deve essere sempre possibile accedere ai server in caso di guasti o malfunzionamenti anche se tale operazione di accesso è ristretta.

Tra le proprietà che un fluido possiede quelle considerate sono:

- **Viscosità**
  E' una grandezza scalare fisica che stabilisce la resistenza di un fluido allo scorrimento (Wikipedia).

- **Velocità**
  Grandezza vettoriale che determinata quanto il fluido si sposta nell'unità di tempo.

- **Temperatura**
  Grandezza scalare che stabilisce lo stato termico del fluido.

- **Densità**
  Grandezza scalare definita come il rapporto tra la massa e il volume del fluido.

- **Calore specifico**
  La quantità di calore necessaria per innalzare la temperatura di una unità di massa di un Kelvin (Wikipedia).

- **Conduttività termica**
  Grandezza definita come il rapporto tra il flusso di calore e gradiente di temperatura; stabilisce la capacità di una sostanza di trasmettere calore.

- **Diffusività termica**
  Grandezza definita come il rapporto tra la conduttività termica e il prodotto tra la densità e il calore specifico.

- **Espansività termica**
  Grandezza che esprime la capacità di una sostanza di cambiare il proprio volume in risposta a una variazione di temperatura.

Le unità di misura per queste grandezze sono espresse nella seguente tabella:

Grandezza	**Unità di misura**
Viscosità	$\frac{Kg}{m \times s}$
Velocità	$\frac{m}{s}$
Temperatura	K
Densità	$\frac{Kg}{m^3}$
Calore specifico	$\frac{J}{Kg \times K}$
Conduttività termica	$\frac{W}{m \times K}$
Diffusività termica	$\frac{m^2}{s}$
Espansività termica	$\frac{m}{s^2 \times K}$

Capitolo 2

Teoria alla base del Simulatore

Avendo stabilito lo scenario di riferimento e la relativa composizione dei suoi elementi bisogna capire quali sono i modelli matematici per la rappresentazione di un fluido per poi concretizzarli nel codice che farà parte del simulatore. La teoria qui presentata riprende i concetti di [5].

2.1 Rappresentazione matematica dei fluidi

Esistono due principali punti di vista da cui guardare un fluido, chiamati Euleriano e Langrangiano: il primo divide lo spazio in regioni finite alle quali vengono attribuite delle proprietà (temperatura, pressione, velocità ...) da studiare; il secondo invece rappresenta il fluido come un insieme di particelle libere di muoversi nello spazio, alle quali sono attribuite delle caratteristiche. La differenza è che nella visione Euleriana i punti di interesse rimangono fissi e non cambiano nel tempo mentre nella Lagrangiana questi sono determinati dallo spostamento delle particelle.

In letteratura il modello più usato è quello Euleriano perché presenta dei vantaggi dal punto di vista implementativo. Secondo questo modello lo spazio viene partizionato in regioni di uguale dimensione; in questo modo si viene a creare una vera e propria griglia di cui se ne identificano due versioni:

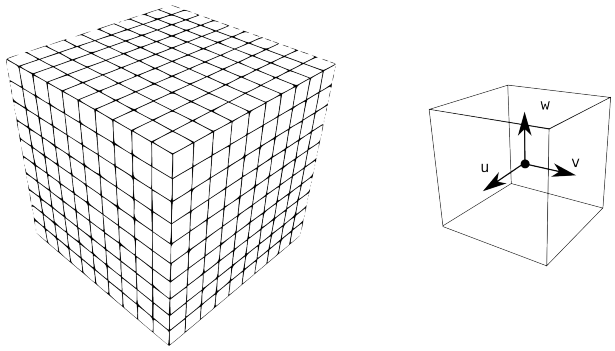

Figura 2.1: Lo spazio considerato suddiviso in voxel(notare le tre componenti del vettore velocità definite nel centro del cubo).

- **Co-Localizzata (Co-Located grid)**
  Dove i valori di ciascuna proprietà sono racchiusi al centro di ogni cella della griglia

- **Sfalsata (Staggered grid)**
  Dove i valori di alcune proprietà (spesso solo la velocità) sono rappresentati sulle facce delle celle mentre altri al centro di esse.

La versione sfalsata porta a risultati più precisi ma rende i calcoli leggermente più complessi rispetto a quella co-localizzata, per questo motivo è stata adottata la prima.

Lo spazio di dimensioni $[N_x, N_y, N_z]$ viene quindi ripartito in tanti cubetti di stesso lato h chiamati voxel come rappresentato in figura 2.1. Questo significa che si avranno un totale di $\frac{N_x}{h} \times \frac{N_y}{h} \times \frac{N_z}{h}$ voxel che tende a $\left(\frac{N^3}{h}\right)$ se si considera una stanza cubica. Per questo motivo è importante sottolineare che più è piccolo il valore di h più i risultati saranno precisi perché aumenteranno il numero di voxel; tuttavia tale miglioramento incrementerà il tempo di calcolo.

2.1.1 Calcolo vettoriale

Per poter capire come un fluido evolve nel tempo bisogna prima definire le operazioni utilizzate nei calcoli di fluidodinamica:

Operatore	**Definizione**	**Forma finita**
Gradiente	$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}, \frac{\partial p}{\partial z}\right)$	$\frac{p_{i+1,j,k} - p_{i-1,j,k}}{2\delta x}$, $\frac{p_{i,j+1,k} - p_{i,j-1,k}}{2\delta y}$, $\frac{p_{i,j,k+1} - p_{i,j,k-1}}{2\delta z}$
Divergenza	$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}$	$\frac{u_{i+1,j,k} - u_{i-1,j,k}}{2\delta x}$ + $\frac{v_{i,j+1,k} - v_{i,j-1,k}}{2\delta y}$ + $\frac{w_{i,j,k+1} - w_{i,j,k-1}}{2\delta z}$
Laplaciano	$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial p^2}{\partial z^2}$	$\frac{p_{i+1,j,k} - 2p_{i,j,k} + p_{i-1,j,k}}{(\delta x)^2}$+ $\frac{p_{i,j+1,k} - 2p_{i,j,k} + p_{i,j-1,k}}{(\delta y)^2}$+ $\frac{p_{i,j,k+1} - 2p_{i,j,k} + p_{i,j,k-1}}{(\delta z)^2}$

L'applicazione del gradiente a un campo scalare restituisce come risultato un vettore di derivate parziali, mentre la divergenza esegue l'operazione inversa prendendo in input un campo vettoriale e restituendo in output un campo scalare (nella tabella viene applicata al vettore **u**). Infine se al risultato del gradiente si applica la divergenza si ottiene l'operatore laplaciano da cui la definizione $\nabla^2 p = \nabla \cdot \nabla p$.

Tutti questi operatori, come è possibile notare nella tabella, considerano una distanza pari a $[\delta x, \delta y, \delta z]$ tra i centri di ogni coppia di celle adiacenti in ciascuna delle tre direzioni degli assi cartesiani; in realtà poiché si attua una suddivisione dello spazio in cubetti di lato uguale su ciascuna componente,

è possibile semplificare tutte le formule ($\delta x = \delta y = \delta z = h$) ad esempio il laplaciano diventa:

$$\nabla^2 p = \frac{p_{i+1,j,k} + p_{i-1,j,k} + p_{i,j+1,k} + p_{i,j-1,k} + p_{i,j,k+1} + p_{i,j,k-1} - 6p_{i,j,k}}{h^2}$$

Avendo stabilito le operazioni utilizzate è possibile parlare del modello che matematicamente rappresenta meglio un fluido.

2.1.2 Equazioni Navier-Stokes

Le assunzioni fatte per l'idealizzazione di un fluido sono le seguenti:

- **Incomprimibilità**
  Il volume di una qualsiasi regione del fluido rimane costante nel tempo.

- **Omogeneità**
  La densità del fluido è costante nello spazio e nel tempo.

Queste due assunzioni sono comuni in CFD e non impediscono l'applicazione della simulazione per rappresentare fluidi reali come l'acqua e l'aria.

Data una griglia tridimensionale con coordinate spaziali $\mathbf{c} = (x, y, z)$ e il tempo t, un fluido è rappresentato da un campo vettoriale per la velocità $\mathbf{u}(\mathbf{c}, t)$ e da uno scalare per la pressione $p(\mathbf{c}, t)$, entrambi variabili nello spazio e nel tempo. Se si conoscono i valori di questi due campi nell'istante iniziale in cui $t = 0$, allora lo stato di un fluido può essere descritto tramite le equazioni di *Navier-Stokes* per un flusso incomprimibile:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F} \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.2}$$

Dove ρ è la densità costante, ν è la viscosità, ed $\mathbf{F} = (f_x, f_y, f_z)$ rappresenta le forze esterne che possono intervenire sul fluido. Bisogna specificare però

che la (2.1) in realtà è composta da tre equazioni data la natura vettoriale di **u**:

$$\frac{\partial u}{\partial t} = -(\mathbf{u} \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu\nabla^2 u + f_x \tag{2.3}$$

$$\frac{\partial v}{\partial t} = -(\mathbf{u} \cdot \nabla)v - \frac{1}{\rho}\nabla p + \nu\nabla^2 v + f_y \tag{2.4}$$

$$\frac{\partial w}{\partial t} = -(\mathbf{u} \cdot \nabla)w - \frac{1}{\rho}\nabla p + \nu\nabla^2 w + f_z \tag{2.5}$$

Analizzando (2.1) si evince che essa è composta da quattro termini che rappresentano tutti delle accelerazioni ma con significati distinti:

$$\frac{\partial \mathbf{u}}{\partial t} = \underbrace{-(\mathbf{u} \cdot \nabla)\mathbf{u}}_{1} \underbrace{-\frac{1}{\rho}\nabla p}_{2} \underbrace{+\nu\nabla^2 \mathbf{u}}_{3} \underbrace{+\mathbf{F}}_{4}$$

1. **Avvezione**

   La velocità di un fluido causa il trasporto di quantità come densità, temperatura e qualunque altro tipo di proprietà attraverso il flusso. Questo termine stabilisce quindi che il fluido esibisce un comportamento di auto-avvezione del campo di velocità ossia esso tende a *trasportare* se stesso insieme al resto.

2. **Pressione**

   Le molecole del fluido possono spostarsi liberamente nello spazio tendendo a *urtarsi* e *scivolare* l'una sull'altra. L'applicazione di una forza in un fluido quindi si propaga a partire dalle particelle più vicine alla fonte che generano lentamente una pressione su quelle più lontane.

3. **Diffusione**

   In natura esistono diversi fluidi, alcuni più *leggeri* di altri. La viscosità esprime la tendenza di un fluido a resistere al flusso risultando in una diffusione del momento e quindi della velocità.

4. **Forze esterne**

   Questo termine contiene tutte le forze esterne che possono influenzare

il fluido. Tra queste possiamo trovare accelerazioni locali (es. condizionatori d'aria) e globali (es. la gravità) a seconda che esse vengano applicate solo in alcune zone o a tutto l'area occupata dal fluido.

Per quanto riguarda invece l'equazione (2.2) essa è chiamata equazione di continuità e serve a far valere l'assunzione di incomprimibilità del fluido. L'operatore di divergenza infatti è usato in fisica per stabilire il tasso al quale la densità esce da una data regione dello spazio; in questo caso esso è applicato al flusso di velocità indicandone la variazione netta sulla superficie di una piccola area di fluido.

Le equazioni di *Navier-Stokes* possono essere risolte analiticamente soltanto in determinati casi. In realtà però è possibile tramite tecniche di integrazione numerica risolverle iterativamente.

Il metodo adottato per trovare le soluzioni delle equazioni è descritto in [2] e viene riassunto di seguito.

2.1.3 Scomposizione Helmholtz-Hodge

Il calcolo vettoriale stabilisce che un qualsiasi vettore $\mathbf{v} = (x, y, z)$ può essere visto come la somma delle sue componenti, $\mathbf{v} = (x, y, z) = x\hat{i} + y\hat{j} + z\hat{k}$, dove $\hat{i}, \hat{j}, \hat{k}$ sono i versori allineati con gli assi cartesiani. Lo stesso principio si può applicare ad un campo vettoriale visto come somma di campi vettoriali, utilizzando il seguente teorema:

Teorema 2.1.1. *Sia D una regione dello spazio avente un confine regolare (differenziabile) detto ∂D, con direzione normale $\boldsymbol{n}$.*

Un campo vettoriale $\boldsymbol{w}$ su D può essere scomposto univocamente:

$$\boldsymbol{w} = \boldsymbol{u} + \nabla p \tag{2.6}$$

Dove $\boldsymbol{u}$ ha divergenza nulla ed è parallelo a ∂D; ossia $\boldsymbol{u} \cdot \boldsymbol{n} = 0$ su ∂D.

In sostanza il teorema dice che un qualunque campo vettoriale può essere scomposto nella somma di due altri campi vettoriali: uno con divergenza nulla e l'altro come gradiente di un campo scalare. Inoltre stabilisce che il campo con divergenza nulla vale 0 sul contorno.

Considerazioni

L'applicazione del teorema (2.1.1) permette di semplificare le equazioni di *Navier-Stokes* giungendo a due importanti considerazioni:

- Ad ogni istante di tempo bisogna calcolare avvezione, diffusione e forze esterne ottenendo cosi' il campo della velocità, $\mathbf{w}$, la cui divergenza non è nulla andando in contraddizione con (2.2). Utilizzando il teorema (2.1.1) si può però correggere la divergenza della velocità sottraendogli il gradiente del campo relativo alla pressione:

$$\mathbf{u} = \mathbf{w} - \nabla p \tag{2.7}$$

- Applicando l'operatore divergenza a entrambi i lati dell'equazione (2.6) si ottiene:

$$\nabla \cdot \mathbf{w} = \nabla \cdot (\mathbf{u} + \nabla p) = \nabla \cdot \mathbf{u} + \nabla^2 p \tag{2.8}$$

  Ma siccome l'equazione (2.2) stabilisce che la divergenza di $\mathbf{u}$ è nulla, essa diventa:

$$\nabla^2 p = \nabla \cdot \mathbf{w} \tag{2.9}$$

  Che altro non è che l'equazione di *Poisson* per la pressione.
  In sostanza questo significa che una volta ottenuto il campo divergente per la velocità, $\mathbf{w}$, basta risolvere l'equazione (2.9) per calcolare p, e dopo utilizzare (2.7) per trovare il valore di $\mathbf{u}$ con divergenza nulla.

2.1.4 Risoluzione equazioni Navier-Stokes

Le considerazioni fatte portano ad una sola incognita, come calcolare $\mathbf{w}$. La definizione di prodotto vettoriale dice che è possibile determinare il valore della proiezione q di un vettore $\mathbf{r}$ su un vettore unitario $\hat{b}$, $q = \mathbf{r} \cdot \hat{b}$. Utilizzando il teorema (2.1.1) è possibile definire un operatore di proiezione $\mathbb{P}$ che proietti il campo vettoriale $\mathbf{w}$ su quello con divergenza nulla $\mathbf{u}$. Applicando $\mathbb{P}$ all'equazione (2.6) si ottiene:

$$\mathbb{P}\mathbf{w} = \mathbb{P}\mathbf{u} + \mathbb{P}(\nabla p)$$

Ma dalla definizione di $\mathbb{P}$, $\mathbb{P}\mathbf{w} = \mathbb{P}\mathbf{u} = \mathbf{u}$, da cui $\mathbb{P}(\nabla p) = 0$.

L'operatore di proiezione viene quindi applicato all'equazione (2.1):

$$\mathbb{P}\left(\frac{\partial \mathbf{u}}{\partial t}\right) = \mathbb{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{F}\right)$$

Poiché $\mathbf{u}$ ha divergenza nulla, $\mathbb{P}\left(\frac{\partial \mathbf{u}}{\partial t}\right) = \frac{\partial \mathbf{u}}{\partial t}$ e per le considerazioni precedenti che stabiliscono che $\mathbb{P}(\nabla p) = 0$, il termine relativo alla pressione viene eliminato ottenendo:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} + \mathbf{F}\right) \tag{2.10}$$

L'equazione cosi' ottenuta incapsula alla perfezione l'algoritmo che verrà implementato nel simulatore; infatti basta calcolare il valore dei termini nelle parentesi per ottenere il valore del campo $\mathbf{w}$ divergente. l'operatore $\mathbb{P}$ in pratica risolve l'equazione (2.9) per calcolare la pressione sottraendone poi il gradiente da $\mathbf{w}$ per ottenere il valore del campo vettoriale $\mathbf{u}$ con divergenza nulla.

Nell'implementazione reale però i termini che determinano il valore di $\mathbf{w}$ non sono calcolati separatamente ma vengono ottenuti tramite trasformazioni dello stato corrente del fluido. In pratica ogni termine prende in input lo stato attuale del fluido e dopo aver eseguito le operazioni relative restituisce in output il nuovo. Questo approccio suggerisce quindi la definizione di un operatore $\mathbb{S}$ che applica gli operatori $\mathbb{F}, \mathbb{D}, \mathbb{A}, \mathbb{P}$ (abbinati a ciascun componente dell'equazione) per trovare il valore del campo vettoriale aggiornato:

$$\mathbb{S}(\mathbf{u}) = \mathbb{P} \circ \mathbb{A} \circ \mathbb{D} \circ \mathbb{F} \tag{2.11}$$

Operatori

In questa sezione viene spiegato il funzionamento di ciascun operatore presente in (2.11):

- **Avvezione** ($\mathbb{A}$)
  Come accennato precedentemente, il processo di avvezione comporta il trasporto da parte del fluido di se stesso e delle altre quantità lungo il campo della velocità. Immaginando ogni voxel come una particella bisogna quindi determinare la nuova posizione raggiunta nell'ipotesi che sia passato tempo δt. Una prima idea per la risoluzione di questo problema potrebbe prendere in considerazione il metodo Euleriano che è definito secondo la seguente equazione:

$$\mathbf{r}(t + \delta t) = \mathbf{r}(t) + \mathbf{u}(t)\delta t \tag{2.12}$$

  Il problema di questo metodo però è che risulta instabile se $\mathbf{u}(t)\delta t$ è più grande del lato del voxel. Per questo motivo come spiegato in [2] si ricorre alla soluzione che traccia la traiettoria della particella indietro nel tempo. Detta q una quantità generica (velocità, temperatura, densità ...) di cui si vuole calcolare la posizione, l'equazione risultante è:

$$q(\mathbf{c}, t + \delta t) = q(\mathbf{c} - \mathbf{u}(\mathbf{c}, t)\delta t, t) \tag{2.13}$$

  In sostanza si determina la posizione di partenza $\mathbf{c}'$ a partire dalla attuale $\mathbf{c}$ e poi tramite interpolazione trilineare sfruttando le otto celle adiacenti a $\mathbf{c}'$ si determina il nuovo valore di q.

- **Diffusione** ($\mathbb{D}$)
  In maniera simile a quanto detto per l'avvezione, una prima equazione per calcolare la diffusione è:

$$\mathbf{u}(\mathbf{c}, t + \delta t) = \mathbf{u}(\mathbf{c}, t) + \nu\delta t\nabla^2\mathbf{u}(\mathbf{c}, t) \tag{2.14}$$

  Anche questa però soffre della stessa instabilità accennata precedentemente; sempre secondo Stam si può utilizzare una forma implicita:

$$(\mathbf{I} - \nu\delta t\nabla^2)\mathbf{u}(\mathbf{c}, t + \delta t) = \mathbf{u}(\mathbf{c}, t) \tag{2.15}$$

  Questa forma è chiamata anche equazione di *Poisson* per la velocità dove $\mathbf{I}$ è la matrice identità.

- **Forze esterne** ($\mathbb{F}$)

  Per quanto riguarda le forze in atto la formula generale utilizzata è:

  $$\mathbf{u}(\mathbf{c}, t + \delta t) = \mathbf{u}(\mathbf{c}, t) + \delta t \mathbf{F}(\mathbf{c}) + f_{gal}(\mathbf{c}) \tag{2.16}$$

  Questa formula in pratica si limita a calcolare la velocità aggiunta dopo che è passato δt tempo supponendo che $\mathbf{F}(\mathbf{c})$ sia l'accelerazione costante nel voxel alla posizione $\mathbf{c}$. In realtà questa equazione la si può utilizzare con opportune modifiche per altre quantità (es. energia e temperatura). Invece per modellare la presenza del moto naturale dei fluidi dovuto a particelle calde che si muovono verso l'alto e fredde verso il basso, si utilizzata questa equazione:

  $$f_{gal}(\mathbf{c}) = \delta t \beta (T_{amb} - T(\mathbf{c})) \tag{2.17}$$

  Dove f_{gal} è anche chiamato *forza di galleggiamento*, β è l'espansività termica, mentre T_{amb} e T sono rispettivamente le temperature dell'ambiente e del voxel.

2.1.5 Risoluzione equazioni di Poisson

Entrambe le equazioni di *Poisson* (2.9) e (2.15) relative alla pressione e velocità possono essere risolte tramite risolutori iterativi che partendo da una soluzione approssimata la migliorano ad ogni iterazione.

L'equazione di *Poisson* ha forma $\mathbf{Ax} = \mathbf{b}$, dove $\mathbf{x}$ è il vettore di cui si vuole trovare la soluzione (in questo caso velocità o pressione), $\mathbf{b}$ è un vettore di costanti, $\mathbf{A}$ è una matrice (in questo caso sarebbe il laplaciano ∇^2 che non viene esplicitamente espresso sotto forma di matrice).

Nel suo paper Stam utilizza il metodo di *Gauss-Siedel* perché è semplice e converge velocemente (esistono altri metodi, ad es. il *Jacobi*, che si prestano meglio per un implementazione parallela).

L'equazione utilizzata è dunque la seguente:

$$x_{i,j,k}^{(n+1)} = \frac{x0_{i,j,k} + \alpha(x_{i-1,j,k}^{(n)} + x_{i+1,j,k}^{(n)} + x_{i,j-1,k}^{(n)} + x_{i,j+1,k}^{(n)} + x_{i,j,k-1}^{(n)} + x_{i,j,k+1}^{(n)})}{\beta} \tag{2.18}$$

Dove n rappresenta il passo di iterazione, *x0* contiene i valori di partenza (pressione o velocità al tempo $t - \delta t$), mentre x sono le quantità che vengono calcolate progressivamente. I valori di α e β invece cambiano a seconda che si stia risolvendo l'equazione per la pressione o quella per la velocità:

$$\alpha = \begin{cases} \dfrac{\nu \delta t}{h^2} & \text{se eq. velocità} \\ 1 & \text{se eq. pressione} \end{cases} \quad \beta = \begin{cases} 1 + 6\alpha & \text{se eq. velocità} \\ 6 & \text{se eq. pressione} \end{cases}$$

Considerando l'equazione della velocità sia *x0* che x fanno riferimento al campo vettoriale $\mathbf{u}$, mentre nell'equazione della pressione $x0 = \nabla \cdot w$ e x è la pressione da calcolare.

2.2 Calore e Temperatura

Ricordando gli elementi presentati nello scenario è facile intuire che servono delle ulteriori equazioni che modellino l'aspetto termico del simulatore.

Calore e temperatura sono due grandezze fisiche che spesso vengono confuse ma che in realtà hanno significati profondamente diversi: il calore è il trasferimento di energia termica (espresso in *Joule*) che avviene tra un sistema e il suo ambiente in base al gradiente di temperatura presente tra i due; la temperatura rappresenta lo stato termico di un sistema. La relazione tra queste due grandezze è legata al fatto che fornire/togliere energia ad un sistema corrisponde a fornire/togliere calore provocando quindi un aumento/riduzione di temperatura. Se ad esempio si pone un oggetto avente una certa temperatura in una stanza, col passare del tempo esso cederà parte della sua energia termica all'ambiente fino a raggiungere con esso l'equilibrio termico (il gradiente di temperatura tra i due è nullo).

In realtà intervengono diversi fattori nello scambio di calore (il materiale, la dimensione, la conducibilità termica ...) ed esistono diverse modalità di trasferimento del calore in base al tipo di corpi considerati:

- **Convezione**
  è un tipo di modalità presente solo nei fluidi che consiste nel *trasporto* di particelle riscaldate o raffreddate, tramite scambio di calore tra corpi con cui il fluido viene a contatto, secondo le forze a cui il fluido è sottoposto. Esistono due tipi di convezione: *forzata* se le differenze di temperatura e densità sono trascurabili al punto che il moto è influenzato esclusivamente da fattori esterni come pompe o ventilatori; *naturale* se le differenze di temperatura e densità sono notevoli a tal punto da generare delle correnti note come forze di *galleggiamento* dovute all'eccitamento delle molecole sottoposte a calore.

- **Conduzione**
  Lo scambio di calore può avvenire o nello stesso corpo o tra due corpi differenti che sono a contatto.

- **Irraggiamento**
  Non c'è contatto tra i sistemi che si scambiano calore, il passaggio avviene infatti tramite assorbimento di onde elettromagnetiche.

Nel simulatore il trasferimento di calore considera solo conduzione e convezione (ipotesi ragionevole dato che i data center sono spesso ambienti chiusi con luce artificiale).

2.2.1 Equazioni Navier-Stokes e Temperatura

Come visto precedentemente per la velocità le equazioni di *Navier-Stokes* possono essere applicate per qualsiasi quantità trasportata dal fluido. La temperatura che è importante nel passaggio di calore non fa eccezione:

$$\frac{\partial T}{\partial t} = -(\mathbf{u} \cdot \nabla)T + \kappa \nabla^2 T + \mathbf{Q} \tag{2.19}$$

Dove T è la temperatura, κ è la diffusività termica e $\mathbf{Q}$ rappresenta l'apporto termico dovuto al calore.

Analizzando (2.19) è evidente che per calcolare il termine relativo all'avvezione, è necessario determinare prima il campo vettoriale relativo alla velocità su cui la temperatura è trasportata (il termine relativo alla pressione infatti è assente in quanto il campo **u** con divergenza nulla è stato già ottenuto). Guardando invece il termine di diffusione compare una nuova variabile κ che è la diffusività termica cosi' definita:

$$\kappa = \frac{k}{\rho c_p} \tag{2.20}$$

dove k è la conduttività termica ossia la capacità di una sostanza di condurre calore (definita anche come l'inverso della resistenza termica), ρ è la densità e c_p il calore specifico che è la capacità termica per unità di massa. Infine il termine **Q** va considerato in maniera leggermente differente rispetto al termine **F** dell'equazione (2.1): se infatti da un lato si assiste all'aggiunta di temperatura secondo un meccanismo simile a quanto avviene per la velocità, dall'altro bisogna considerare come l'energia si trasformi in calore trasmesso sia per convezione che per conduzione. In sostanza quindi il termine **Q** modella l'aggiunta di energia esterna (es. alimentazione di un calcolatore, un condizionatore, ecc ...) che comporta poi una modifica della temperatura del corpo che la riceve.

Analogamente a quanto fatto per la velocità va definito un operatore $\mathbb{T}$ (applicato ad ogni passo di simulazione) che è la composizione degli operatori $\mathbb{A}, \mathbb{D}, \mathbb{Q}$:

$$\mathbb{T}(T) = \mathbb{A} \circ \mathbb{D} \circ \mathbb{Q} \tag{2.21}$$

Operatori

Gli operatori presenti in $\mathbb{T}$ sono:

- **Avvezione** ($\mathbb{A}$)
  Il termine di avvezione viene risolto secondo quanto stabilito per la velocità nell'equazione (2.13) con la differenza che le posizioni di partenza che vengono calcolate per l'interpolazione trilineare devono essere tutte

celle di fluido altrimenti si verificherebbe una simulazione erronea dove la temperatura viene spostata direttamente da un corpo per mezzo del movimento del fluido (comportamento in contraddizione con le ipotesi fatte sui meccanismi che regolano il passaggio del calore).

- **Diffusione** ($\mathbb{D}$)
  Anche la diffusione viene calcolata come delineato per il campo vettoriale della velocità tramite l'equazione (2.15). In questo caso però la viscosità ν è sostituita con la diffusività termica κ che varia in base alla sostanza considerata, infatti questo termine considera sia la convezione che la conduzione modellando lo scambio di calore all'interno e all'esterno di un corpo/fluido.

- **Calore** ($\mathbb{Q}_*$)
  Il valore del campo scalare Q relativo al calore viene trasformato in temperatura secondo la seguente formula:

$$T(\mathbf{c}, t + \delta t) = T(\mathbf{c}, t) + \delta t \left(\frac{Q(\mathbf{c})}{c_p(\mathbf{c}) m(\mathbf{c})} \right) \tag{2.22}$$

  Questa equazione stabilisce la quantità di temperatura guadagnata/persa nel tempo δt dovuta al calore ricevuto. è importante notare che in questo caso il termine Q va propriamente riconosciuto come una potenza dal punto di vista fisico (misurato in $Watt = \frac{Joule}{sec}$, che è energia nell'unità di tempo).

2.3 Ostacoli

L'ultimo passo per completare la definizione delle dinamiche del fluido risiede nella gestione degli ostacoli, ossia tutte quelle entità che occupano parte del volume in cui il fluido risiede. [2] suggerisce una possibile implementazione degli ostacoli: L'idea in sostanza è insita nel modello stesso utilizzato per rappresentare l'ambiente dato che ogni quantità è rappresentata con un

campo (vettoriale o scalare), viene naturale fare lo stesso con gli ostacoli definendo un campo booleano che indica se un voxel è occupato.

Affinché il fluido non uscisse dall'ambiente simulato Stam considera come ostacoli solo il i bordi della stanza stabilendo che il fluido avrà velocità nulla al confine con essi (condizione di slittamento nullo). Per questo motivo la velocità del voxel ostacolo deve essere pari per intensità a quella del voxel fluido ad esso adiacente ma in direzione opposta. L'equazione della pressione di *Poisson* inoltre richiede che venga rispettata la condizione pura di *Neumann* ossia che il tasso di cambiamento della pressione in direzione normale all'ostacolo sia zero.

Dette u e p la velocità e la pressione al centro del voxel, bisogna quindi imporre a ogni passo di iterazione per ogni faccia di ogni cubetto ostacolo le seguenti condizioni:

$$\frac{u_{obs} + u_{flu}}{2} = 0 \tag{2.23}$$

$$\frac{p_{flu} - p_{obs}}{h} = 0 \tag{2.24}$$

dove la velocità e la pressione del voxel ostacolo sono u_{obs} e p_{obs}, mentre quelle relative al voxel contenente fluido sono u_{flu} e p_{flu}. Da queste poi si ricava che:

$$u_{obs} = -u_{flu} \tag{2.25}$$

$$p_{obs} = p_{flu} \tag{2.26}$$

L'unico problema che può presentarsi è dovuto alla tipologia di griglia adottata (co-localizzata): nel caso in cui l'ostacolo occupi un solo voxel lungo una delle tre dimensioni risulterebbe infatti ambiguo stabilire a quale delle due facce confinanti doversi opporre. Annullando sia il valore della velocità che quello della pressione è come se si stesse considerando un ostacolo di lato pari a $\frac{h}{2}$ che è una supposizione ragionevole in questo caso particolare.

Capitolo 3

Sviluppo del Simulatore

L'analisi teorica sui modelli atti alla rappresentazione dei fluidi e le informazioni su un possibile ambiente da ricostruire virtualmente sono entrambe i punti di partenza per lo sviluppo del simulatore.

3.1 Requisiti

Prima di poter creare un'applicazione è necessario avere ben presente i requisiti che essa deve soddisfare sia dal punto di vista funzionale che strutturale e inoltre serve un disegno che è l'architettura software che costituisce la struttura del sistema.

3.1.1 Requisiti funzionali

I requisiti funzionali descrivono tutte le possibili funzionalità che è possibile effettuare utilizzando l'applicazione:

- Creare, caricare, salvare file esperimento.
- Creare un ambiente di determinate dimensioni e scegliere il dettaglio di discretizzazione ossia il valore di h (lato del voxel).
- Creare ed eliminare entità con determinate proprietà (temperatura, velocità, posizione, volume, ...).

- Scegliere i parametri del fluido (temperatura iniziale, viscosità, conduttività termica, ...).
- Duplicare entità e le loro proprietà.
- Scegliere cosa deve essere *disegnato* sullo schermo.
- Iniziare e fermare la simulazione.

3.1.2 Requisiti non funzionali

I requisiti non funzionali invece devono stabilire le proprietà dell'applicazione:

- **Usabile**
  Permette di creare con facilità gli scenari senza costringere l'utente a eseguire operazioni macchinose o troppo complesse.
- **Configurabile**
  Prevede un meccanismo che salva le impostazioni dell'utente e permette di parametrizzare determinati aspetti della simulazione.
- **Efficiente**
  Esegue i calcoli velocemente sfruttando il minor numero di risorse possibili.
- **Portabile**
  Il codice può essere compilato senza troppe modifiche su piattaforme differenti.

3.2 Architettura

Prima di passare allo sviluppo vero e proprio, bisogna capire come è organizzata al suo interno l'applicazione indipendentemente dal linguaggio di

programmazione scelto. Considerando la portabilità occorre dividere il software in due parti: la prima relativa all'interfaccia grafica e la seconda per il cuore del sistema; i benefici di questa scelta comportano una facilità maggiore nel rendere disponibile il simulatore su piattaforme diverse (occupandosi di riscrivere solo il codice relativo alla grafica) e anche la possibilità di utilizzarlo solo da riga di comando.

Per quanto concerne l'organizzazione dei componenti bisogna dire che in letteratura esistono diverse tecniche per dividere grafica, logica e dati dell'applicazione; spesso viene adottato il pattern *Model-View-Controller* (MVC, visto in [15]) e la sua variante *Mediator* (scelta nel framework *Cocoa* in *Mac OSX*), in questo caso però l'approccio è leggermente differente perché il simulatore è composto da *Sistemi* e *Entità*: un entità è semplicemente un *contenitore* di componenti relativi alle proprietà che essa possiede; un sistema implementa la logica ad esso associata (es. il processo di rendering) basandosi solitamente su alcune caratteristiche delle entità su cui opera. A confronto con MVC si può dire che un sistema altro non è che un controller ma in realtà in questa architettura non è presente alcuna associazione implicita con la vista da aggiornare.

Manca però il tassello fondamentale per capire come i sistemi interagiscono tra loro; visto che esiste un unico sistema per ogni logica diversa è immediato l'accesso ad un generico dato contenuto in esso; ma durante l'utilizzo dell'applicazione possono verificarsi degli eventi (es. caricare file esperimento, inzio/fine simulazione, ...) dei quali sistemi diversi potrebbero volere essere avvisati. Questo suggerisce quindi che esista un gestore degli eventi che notifichi i sistemi interessati su determinati avvenimenti.

3.3 Implementazione

Secondo quanto detto per i requisiti è necessario che l'applicazione sia efficiente; per questo motivo è stato scelto come linguaggio di programmazione il *C/C++*. Data la necessità di utilizzare un ambiente interattivo tridimen-

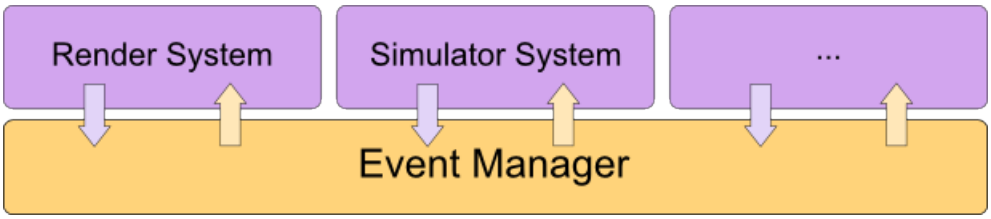

Figura 3.1: I sistemi si appoggiano sul gestore degli eventi per essere notificati su determinati avvenimenti.

sionale è stato scelto il framework *cinder* che semplifica notevolmente l'uso della libreria *OpenGL*; la nota dolente di *cinder* è che funziona solo su *Windows* e *Mac OSX*, quindi per altre piattaforme basate su *Unix* è necessario utilizzare un altro framework. Per quanto riguarda i file di esperimento è stato adottato il linguaggio *XML* e quindi è stata utilizzata la libreria *tinyxml2* per manipolare tali file. Infine per poter ottenere come output del simulatore delle immagini che rappresentino lo stato termico dell'ambiente simulato è stata scelta la libreria *CImg*. Le librerie utilizzate si possono ottenere seguendo gli indirizzi in [16], [17], [18].

Lo sviluppo del simulatore ha portato quindi a una sua divisione in due parti dette front-end e back-end (analogia con il linguaggio utilizzato nei compilatori) dove la prima si occupa della grafica e la seconda dei calcoli.

Infine bisogna notare che laddove è utilizzato il carattere `truetype` si fa riferimento a classi, procedure o file sviluppati.

3.3.1 Applicazione Cinder

Il framework *cinder* permette un rapido sviluppo di software ed è spesso adoperato in ambito multimediale o ludico. Un'applicazione basata su *cinder* quindi deve semplicemente definire il suo comportamento alla chiamata di alcuni metodi che il framework mette a disposizione:

- `setup`

  In questa funzione va inserito il codice che esegue le operazioni per

inizializzare le strutture dati e gli oggetti.

- `update`
  è la funzione richiamata ciclicamente dal framework in base al framerate impostato; qui vanno eseguite le procedure di aggiornamento dati se necessario.

- `draw`
  Anche questa è richiamata ciclicamente e deve contenere il codice che si occupa della rappresentazione grafica.

- `shutdown`
  è la funzione chiamata alla chiusura dell'applicazione; deve contenere il codice per liberare le risorse allocate.

Esistono poi altre procedure relative alla gestione dell'input (mouse e tastiera) e si rimanda alla documentazione di *cinder* per una descrizione più approfondita.

3.3.2 Back-End

Il back-end rappresenta il cuore del simulatore; esso contiene tutte le classi che ricoprono un ruolo fondamentale nell'architettura scelta:

- `System`
  è una classe astratta da cui devono derivare tutti i sistemi implementati

- `Component`
  è una classe astratta da cui devono derivare tutti i componenti che un entità può avere.

- `EntityManager`
  Rappresenta il gestore delle entità capace di eseguire creazione/ distruzione di entità e aggiunta/rimozione di componenti.

- `EventManager`
  è il gestore degli eventi che permette ai `System` di essere notificati.

- `Event`
  è una classe che contiene nel informazioni sull'evento che si è verificato.

- `SimulatorSystem`
  Il vero è proprio nucleo del simulatore. Si occupa di svolgere i calcoli di fluidodinamica.

3.3.3 Front-End

Per quanto riguarda il front-end invece, esso ha un'unica classe che è `RenderSystem` che possiede tutte le procedure per interpretare i dati di simulazione e fornirne una rappresentazione grafica.

3.3.4 Strutture dati

Secondo la teoria un fluido è visto come una serie di campi vettoriali e scalari per ciascuna delle sue proprietà. Da questa definizione però risulta più sensato pensare al valore di tali caratteristiche su tutto l'ambiente simulato e non soltanto per il fluido. Per questo motivo per ciascuna delle proprietà prese in esame vengono allocati due array che rappresentano rispettivamente il valore attuale e quello precedente del campo considerato. Gli array sono monodimensionali ma contengono i dati relativi a tutto l'ambiente che è tridimensionale; è stato adottato questo approccio per motivi di prestazioni.

La grandezza di questi array dipende fortemente dal valore h che rappresenta il lato dei voxel. Poiché quasi tutte le proprietà sono array di `float`, è possibile determinare quale dovrebbe essere il valore massimo di h in relazione alla quantità di RAM del calcolatore; considerato un ambiente di dimensione $[N_x, N_y, N_z]$ e detta d la quantità di byte richiesta per un singolo

elemento di un array

$$\frac{N_x}{h} \times \frac{N_y}{h} \times \frac{N_z}{h} = \textit{numero totale di voxel} \tag{3.1}$$

$$\frac{N_x N_y N_z}{h^3} \times d = \textit{quantità in byte richiesta per l'array} \tag{3.2}$$

Se il numero di array da rappresentare nel calcolatore è chiamato C e la quantità di RAM disponibile è detta R allora si ha che:

$$Cd\frac{N_x N_y N_z}{h^3} = R \tag{3.3}$$

Da cui si ricava che :

$$h = \sqrt[3]{\frac{CdN_x N_y N_z}{R}} \tag{3.4}$$

Questa formula è utile per capire che genere di dettaglio è possibile ottenere in base alle risorse disponibili.

Altre strutture dati importanti sono:

- `Event`
  Contiene un identificatore numerico intero che stabilisce il tipo di evento, e un puntatore `void*` che punta ai dati dell'evento.

- `Pool`
  è una classe template utilizzata per evitare la continua allocazione di risorse; ha bisogno di un'ulteriore classe `PoolFactory` anche essa template, per poter allocare gli oggetti generici.

3.3.5 Entità

Un entità rappresenta un elemento del sistema studiato a cui sono associate delle proprietà ben definite. `Entity` non è una classe ma un tipo primitivo `long` perché le entità in questa architettura sono rappresentate solo tramite un numero.

Preso cosi' da solo però un identificatore numerico ha poco senso; infatti una generica entità può possedere delle proprietà ad essa legate. E' stata

quindi creata la classe astratta `Component` che deve essere ereditata da tutte le classi che rappresentano una o un insieme di proprietà ben precise (figura 3.2):

- `NameComponent`
  Contiene una `string` che rappresenta il nome dell'entità.

- `VolumeComponent`
  Rappresenta il volume (parallelepipedo rettangolo) occupato dall'entità tramite tre `float` per ciascuna delle dimensioni dello spazio.

- `PositionComponent`
  è il centro in cui si trova l'entità; tale punto definito come tre `float` viene interpretato come il centro del volume.

- `MaterialComponent`
  Contiene le proprietà relative al materiale (densità, calore specifico, conduttività termica); inoltre stabilisce se l'entità rappresenta un ostacolo.

- `TemperatureComponent`
  Stabilisce lo stato termico iniziale e la temperatura massima /minima dovuta all'energia.

- `EnergyComponent`
  Contiene la quantità di energia, espressa come potenza, che viene aggiunta/rimossa nel tempo.

- `VelocityComponent`
  Esprime la velocità in forma di accelerazione presente nell'entità.

L'Allocazione e la deallocazione, aggiunta, rimozione e selezione di componenti di un `Entity` sono operazioni eseguite dal `EntityManager` che si occupa della gestione delle entità.

Tramite questo meccanismo in sostanza si ha la libertà di creare qualunque tipo di entità virtuale uno desideri. Inoltre va precisato che all'interno

di una classe derivata da `Component` sia assente qualsiasi tipo di logica che viene poi attuata all'interno dei sistemi.

3.3.6 Sistemi

La classe `System` è in realtà una vera e propria interfaccia in quanto contiene un solo metodo virtuale puro chiamato `update` che viene invocato sui sistemi che derivano da questa classe quando si ritiene che essi debbano attuare la logica per la quale sono stati creati. Gli unici sistemi sviluppati sono `RenderSystem` e `SimulatorSystem`: sul primo viene richiamato il metodo `update` durante la procedura `draw` dell'applicazione *cinder* cosi' da fargli aggiornare l'interfaccia utente; il secondo invece è aggiornato durante l'esecuzione della funzione `update` chiamato dal ciclo principale di *cinder*.

`SimulatorSystem`

Il sistema che si occupa dell'intera simulazione è contenuto in questa classe. In essa sono presenti quindi tutti gli array per ciascuna delle proprietà studiate, le variabili contenenti i parametri della simulazione e tutte le entità presenti nell'ambiente. Quello che accade caricato un file esperimento è espresso dai seguenti passi:

- Inizializza tutti i campi vettoriali e scalari per ciascuna delle proprietà interessate.
- Trasforma gli oggetti dell'ambiente in voxel associando a ciascun cubetto ottenuto le relative caratteristiche dell'entità considerata.

Quando la simulazione viene iniziata essa può essere o non essere sincronizzata con la rappresentazione grafica: se non lo è significa che viene creato un thread separato da quello di *cinder* per i calcoli di fluidodinamica. Questa soluzione è stata adottata per evitare che l'applicazione venisse bloccata dall'elevata mole computazionale. La figura 3.3 mostra il diagramma delle classi relativo a `SimulatorSystem` e `RenderSystem`.

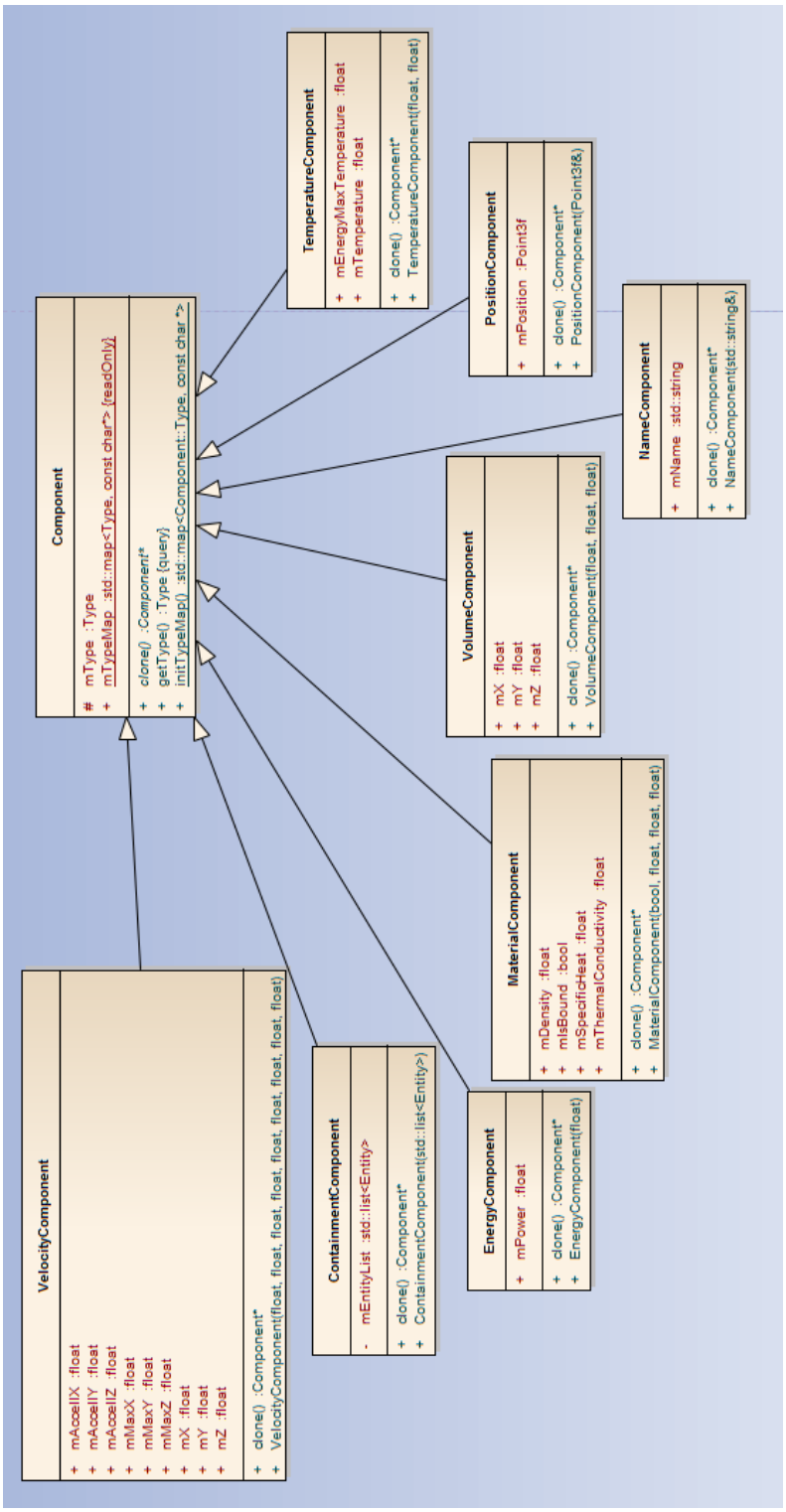

Figura 3.2: Diagramma delle classi inerent ai possibili componenti attribuibili ad un entità.

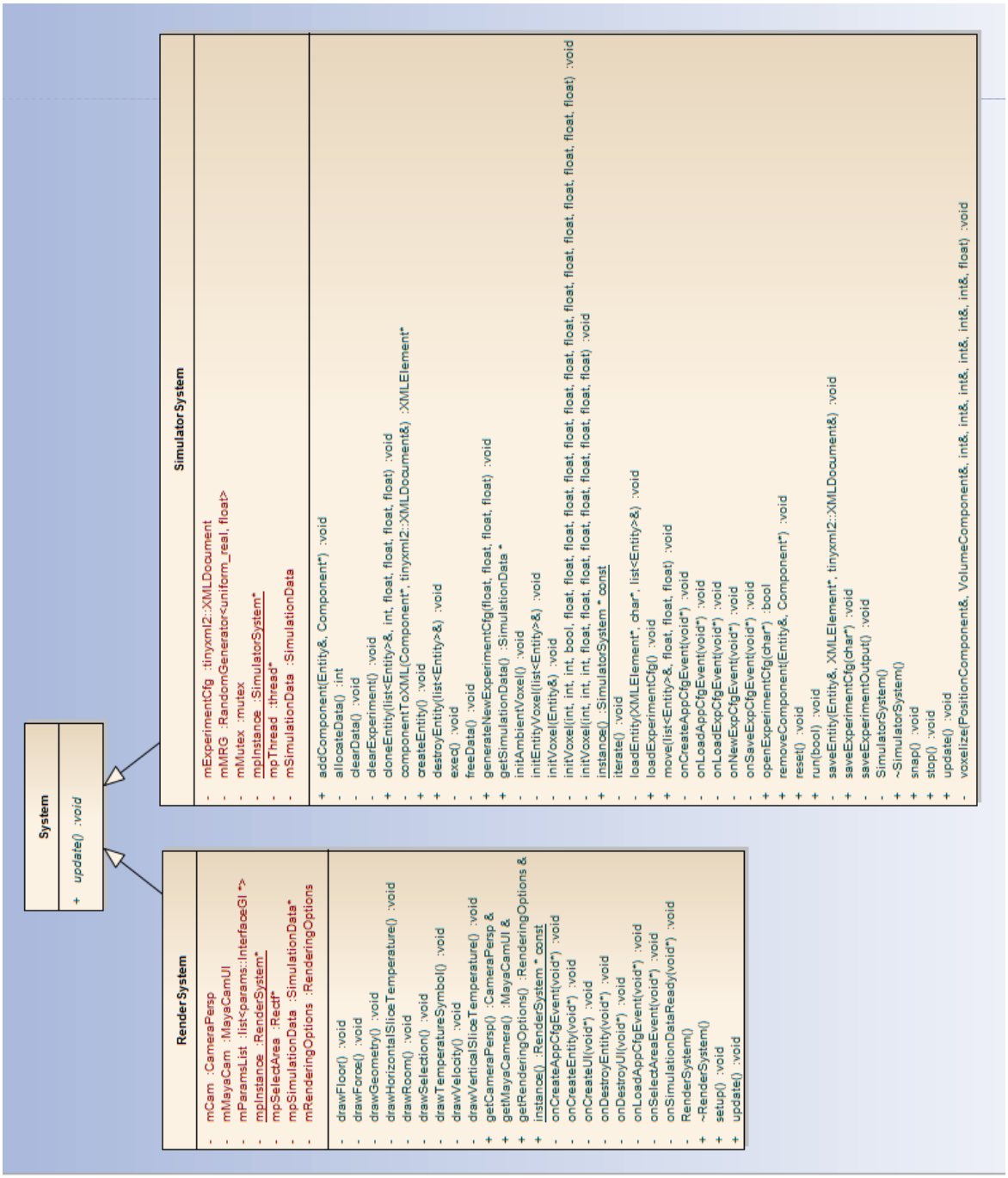

Figura 3.3: Diagramma delle classi che mostra i due sistemi sviluppati.

3.3.7 Formato degli Esperimenti

Un file esperimento deve contenere tutti i parametri della simulazione, gli oggetti presenti nella scena e tutte le proprietà ad essi associate. Per questo motivo *XML* è sembrato la scelta migliore in quanto permette con estrema facilità la descrizione e la definizione di elementi ben strutturati.
I nodi presenti in questi file sono:

- `Configuration`
  è il nodo radice e contiene al suo interno il nodo `Simulation`.

- `Simulation`
  Questo nodo possiede una lunga serie di attributi (il numero di iterazioni, le caratteristiche del fluido, le dimensioni dell'ambiente, ...) e un nodo figlio chiamato `Entities`.

- `Entities`
  Contiene i nodi `Entity` che possono rappresentare una qualunque entità presente nella scena.

- `Entity`
  Per ognuna delle componenti associate ad un entità deve essere presente il relativo nodo nel file esperimento (se ne riportano solo le associazioni):

  - `Name` → `NameComponent`
  - `Volume` → `VolumeComponent`
  - `Position` → `PositionComponent`
  - `Material` → `MaterialComponent`
  - `Temperature` → `TemperatureComponent`
  - `Energy` → `EnergyComponent`
  - `Velocity` → `VelocityComponent`

I file esperimento devono rispettare la struttura definita nel seguente DTD:

```
<!ELEMENT Configuration (Simulation)>
<!ELEMENT Simulation (Entities?)>
<!ATTLIST Simulation
        seed CDATA #REQUIRED
        iterations CDATA #REQUIRED
        voxelSize CDATA #REQUIRED
        areaX CDATA #REQUIRED
        areaY CDATA #REQUIRED
        areaZ CDATA #REQUIRED
        deltaTime CDATA #REQUIRED
        ambientTemperature CDATA #REQUIRED
        ambientInitialMinVelocityX CDATA #REQUIRED
        ambientInitialMaxVelocityX CDATA #REQUIRED
        ambientInitialMinVelocityY CDATA #REQUIRED
        ambientInitialMaxVelocityY CDATA #REQUIRED
        ambientInitialMinVelocityZ CDATA #REQUIRED
        ambientInitialMaxVelocityZ CDATA #REQUIRED
        viscosity CDATA #REQUIRED
        thermalExpansivity CDATA #REQUIRED
        thermalConductivity CDATA #REQUIRED
        horizontalSliceLevel CDATA #REQUIRED
        verticalSliceLevel CDATA #REQUIRED
        density CDATA #REQUIRED
        specificHeat CDATA #REQUIRED
        wallsDensity CDATA #REQUIRED
        wallsSpecificHeat CDATA #REQUIRED
        wallsThermalConductivity CDATA #REQUIRED
        autoSnap CDATA #REQUIRED
        autoSnapInterval CDATA #REQUIRED
```

```
        format CDATA #REQUIRED
        pixelXVoxel CDATA #REQUIRED
        blur CDATA #REQUIRED
        blurSigma CDATA #REQUIRED
        temperatureMinScale CDATA #REQUIRED
        temperatureMaxScale CDATA #REQUIRED>
<!ELEMENT Entities (Entity+)>
<!ELEMENT Entity (Name?, Position+, Volume+, Temperature+, Energy+,
                  Material+, Velocity+)>
<!ATTLIST Name
        id CDATA #REQUIRED>
<!ATTLIST Position
        x CDATA #REQUIRED
        y CDATA #REQUIRED
        z CDATA #REQUIRED>
<!ATTLIST Volume
        volX CDATA #REQUIRED
        volY CDATA #REQUIRED
        volZ CDATA #REQUIRED>
<!ATTLIST Temperature
        temperature CDATA #REQUIRED
        energyMaxTemperature CDATA #REQUIRED>
<!ATTLIST Energy
        power CDATA #REQUIRED>
<!ATTLIST Material
        density CDATA #REQUIRED
        specificHeat CDATA #REQUIRED
        thermalConductivity CDATA #REQUIRED>
<!ATTLIST Velocity
        velX CDATA #REQUIRED
        velY CDATA #REQUIRED
```

```
velZ CDATA #REQUIRED
maxX CDATA #REQUIRED
maxY CDATA #REQUIRED
maxZ CDATA #REQUIRED
accellX CDATA #REQUIRED
accellY CDATA #REQUIRED
accellZ CDATA #REQUIRED>
```

Si riporta una breve descrizione di ciascuno dei campi del file DTD:

- `Simulation`
  - `seed`
    Valore del seme di partenza del generatore di numeri casuali.
  - `iterations`
    Numero di iterazioni da eseguire.
  - `voxelSize`
    Dimensione del lato dei voxel.
  - `areaX`, `areaY`, `areaZ`
    Dimensioni dell' ambiente.
  - `deltaTime`
    Quantità di tempo passata ad ogni iterazione.
  - `ambientTemperature`
    Temperatura iniziale dell'ambiente.
  - `ambientInitialMinVelocityX`, `ambientInitialMinVelocityY`, `ambientInitialMinVelocityZ`
    Quantità che indica la velocità minima iniziale in ciascuna delle tre dimensioni.
  - `ambientInitialMaxVelocityX`, `ambientInitialMaxVelocityY`, `ambientInitialMaxVelocityZ`
    Quantità che indica la velocità massima iniziale in ciascuna delle tre dimensioni.

- `viscosity`
  Viscosità del fluido.
- `thermalExpansivity`
  Espansività termica del fluido.
- `horizontalSliceLevel`
  Valore che indica la posizione dell'area orizzontale di cui si vuole ottenere l'output.
- `verticalSliceLevel`
  Valore che indica la posizione dell'area verticale di cui si vuole ottenere l'output.
- `density`
  Densità del fluido.
- `specificHeat`
  Calore specifico del fluido.
- `wallsDensity`
  Densità dell'area di contorno.
- `wallsSpecificHeat`
  Calore specifico dell'area di contorno.
- `wallsThermalConductivity`
  Conduttività termica dell'area di contorno.
- `autoSnap`
  Booleano che indica se il simulatore deve emettere i risultati a determinati intervalli di tempo.
- `autoSnapInterval`
  Dimensione dell'intervallo di tempo al quale emettere i risultati.
- `format`
  Formato dei risultati.

  - `pixelXVoxel`
    Dimensione che stabilisce quanti pixel si vogliono attribuire ad ogni voxel nei risultati che hanno un formato immagine.
  - `blur`
    Booleano che stabilisce se si vuole applicare un filtro di blur gaussiano ai risultati.
  - `blurSigma`
    Valore del coefficiente di blur del filtro gaussiano.
  - `temperatureMinScale`
    Valore minimo della scala della temperatura.
  - `temperatureMaxScale`
    Valore massimo della scala della temperatura.

- `Name`
  - `id`
    Stringa che indica il nome associato all'entità.

- `Position`
  - `x`, `y`, `z`
    Posizione del centro dell'entità nelle tre dimensioni.

- `Volume`
  - `volX`, `volY`, `volZ`
    Spazio occupato dall'entità nelle tre dimensioni.

- `Temperature`
  - `temperature`
    Temperatura iniziale dell'entità.
  - `energyMaxTemperature`
    Temperatura di regime (massima/minima) che l'entità può raggiungere.

- `Energy`
  - `power`
    Quantità di potenza aggiunta/rimossa all'entità nell'unità di tempo.
- `Material`
  - `density`
    Densità del materiale dell'entità.
  - `specificHeat`
    Calore specifico del materiale dell'entità.
  - `thermalConductivity`
    Conduttività termica del materiale dell'entità.
- `Velocity`
  - `velX`, `velY`, `velZ`,
    Velocità iniziale dell'entità.
  - `maxX`, `maxY`, `maxZ`,
    Velocità massima/minima dell'entità.
  - `accellX`, `accellY`, `accellZ`,
    Accelerazione subita dall'entità nell'unità di tempo.

Capitolo 4

Valutazione sperimentale

La verifica sul corretto funzionamento del simulatore prodotto è stata attuata tramite diversi esperimenti volti a provare ciascuno degli aspetti implementati. La valutazione sperimentale permette di comprendere se i risultati degli scenari riprodotti siano effettivamente coerenti con le previsioni fatte sul loro esito.

4.1 Esperimento 1: Velocità

Lo scenario di questo esperimento prevede una sorgente di aria a temperatura ambiente posta su uno dei lati della stanza. L'obiettivo dell'esperimento è verificare il corretto funzionamento di ciascun termine appartenente alle equazioni di *Navier-Stokes* implementate nel simulatore. La figura 4.1 mostra come la velocità si propaghi lentamente nell'ambiente circostante a partire dalla sorgente in questo caso inserita nel muro. Dal punto di vista grafico è possibile notare dei puntini rossi che in realtà sono dei vettori centrati nel centro di ciascun voxel rivolti nella direzione calcolata dal simulatore. In figura 4.2 si vede come il procedere della simulazione porta a uno stato in cui tutti i cubetti che compongono la stanza saranno influenzati dal flusso del fluido.

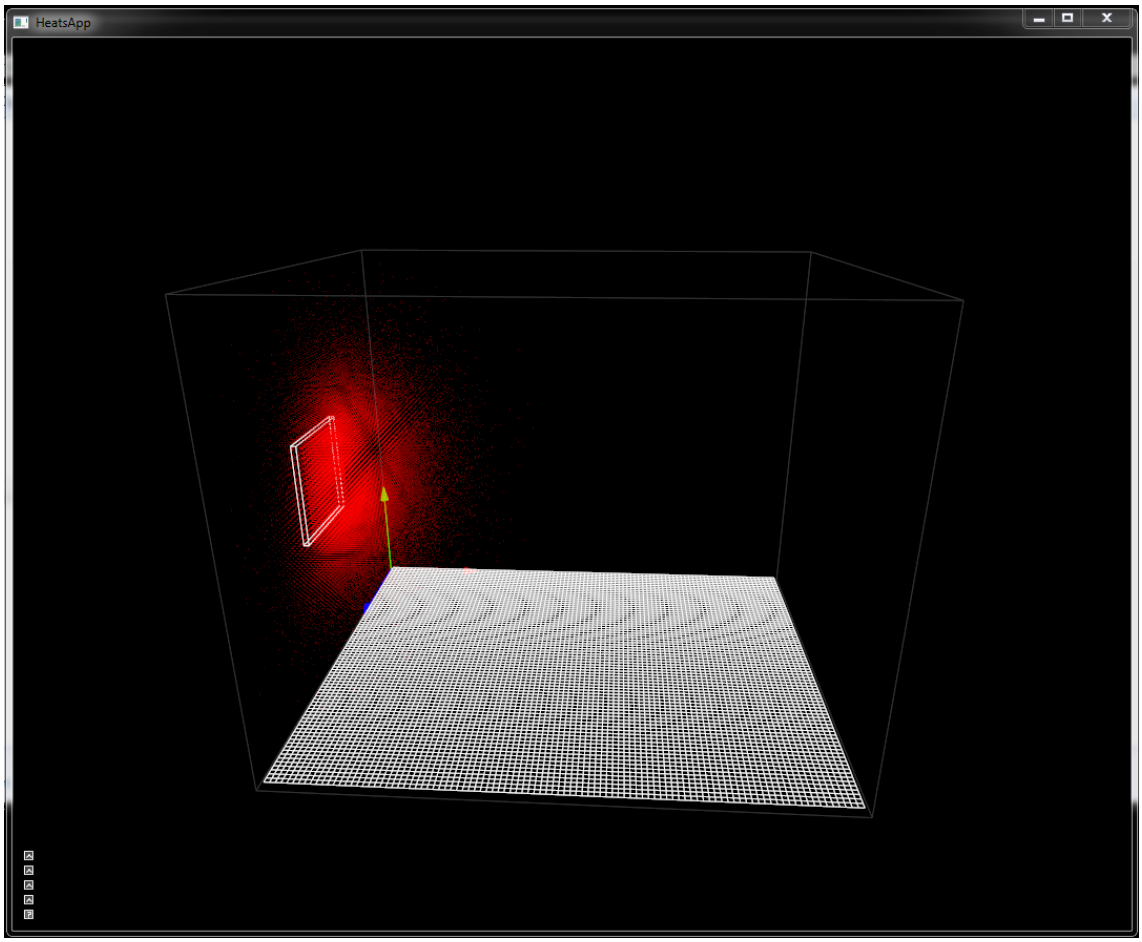

Figura 4.1: La velocità viene aggiunta nei voxel della grata ad ogni iterazione.

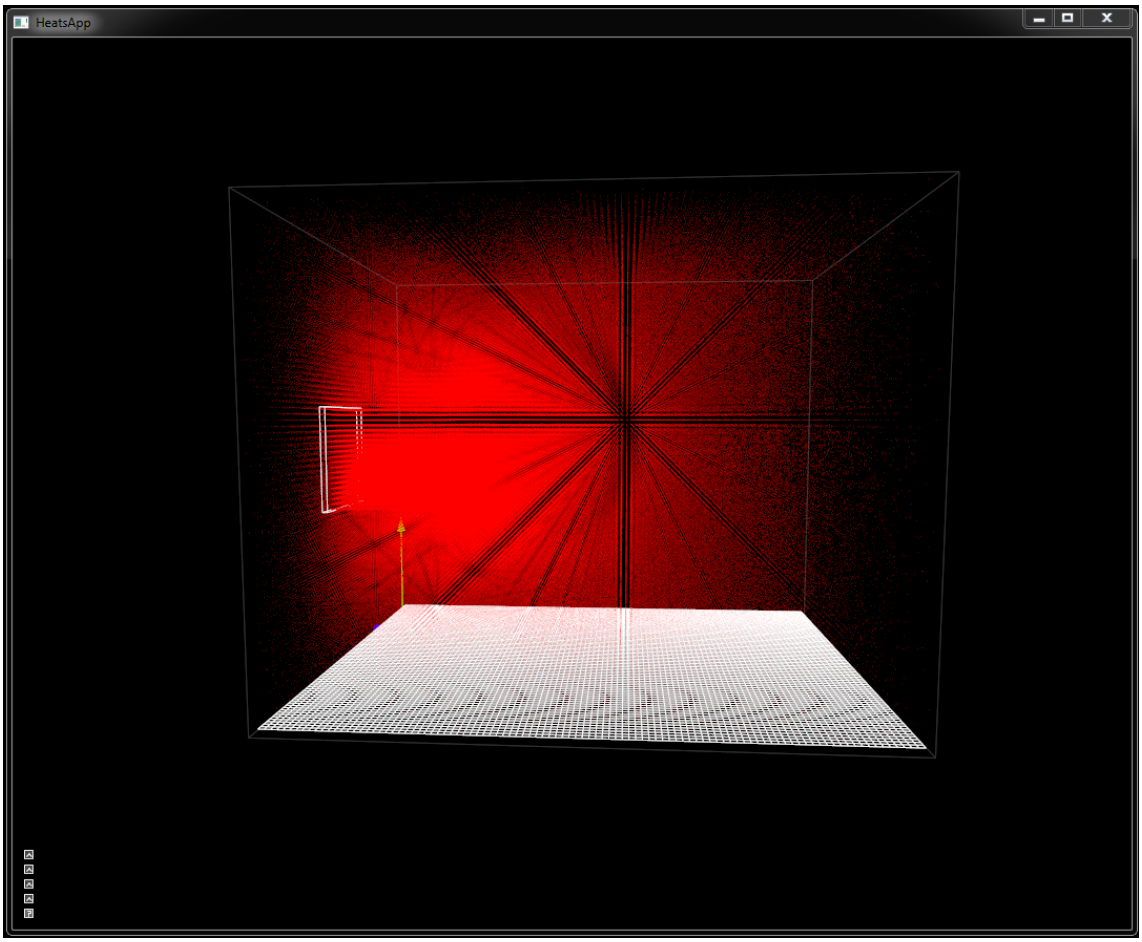

Figura 4.2: La velocità si propaga poi in tutto il fluido.

4.2 Esperimento 2: Velocità e Ostacoli

Il primo esperimento è senza dubbio utile per capire se il simulatore sta calcolando correttamente le equazioni di avvezione e diffusione inerenti alla velocità ma può essere reso più interessante se si aggiungono dei corpi nell'ambiente per vedere come vengono applicate le condizioni di slittamento nullo e di *Neumann* relative alla pressione.

Questo secondo esperimento prevede la stessa sorgente di velocità posta nel muro con l'aggiunta di un pilastro di cemento nel centro della stanza. Poiché la rappresentazione vettoriale non è molto chiara per capire se la velocità è stata deviata in presenza dell'ostacolo, è stata impostata la temperatura iniziale della grata a un valore maggiore di quella dell'ambiente infatti le figure 4.3 e 4.4 mostrano le sezioni verticali e orizzontali del campo scalare della temperatura in prossimità dell'ostacolo.

Analizzando entrambe le figure è possibile notare come il fluido abbia trasportato la temperatura originariamente presente nella grata; inoltre poiché la velocità è annullata in direzione normale alla superficie di contatto si assiste allo scorrimento del fluido sulle facce dell'ostacolo.

4.3 Esperimento 3: Convezione

Anche se il secondo esperimento prevede il passaggio della temperatura dalla grata di areazione al fluido, questo terzo esperimento prevede uno scenario in cui è presente un corpo fatto di rame posto al centro dell'ambiente alla temperatura di 400 Kelvin.

Questo esperimento ha permesso di verificare la correttezza del codice relativo relativo allo scambio di calore per convezione e conduzione perché si assiste al progressivo raffreddamento del corpo che cede calore al fluido. Nella figura 4.5 è rappresentato il corpo in wireframe rosso e il suo stato termico; è interessante notare che i voxel a contatto con la superficie dell'oggetto assumono una colorazione blu che indica una temperatura minore rispetto a

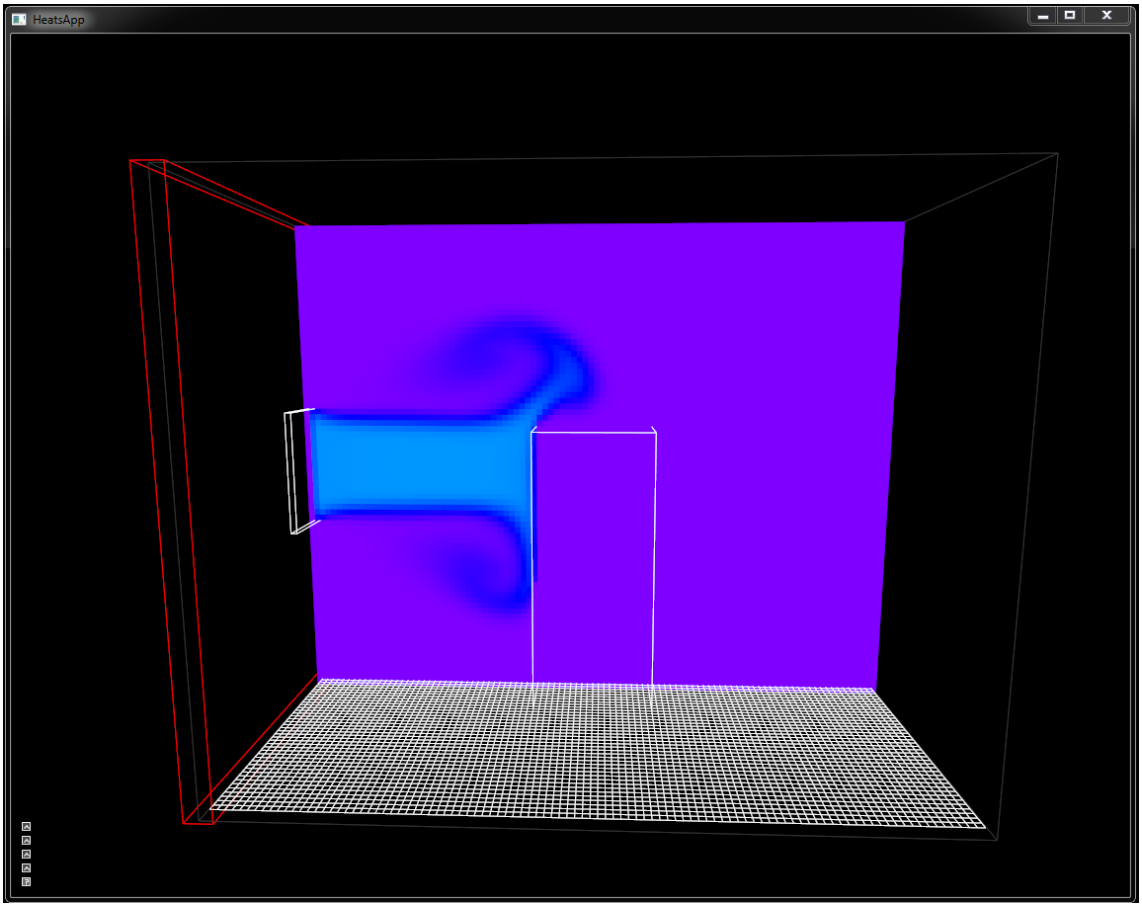

Figura 4.3: Sezione verticale che mostra una netta differenza nel campo di temperatura dovuta alla presenza dell'ostacolo.

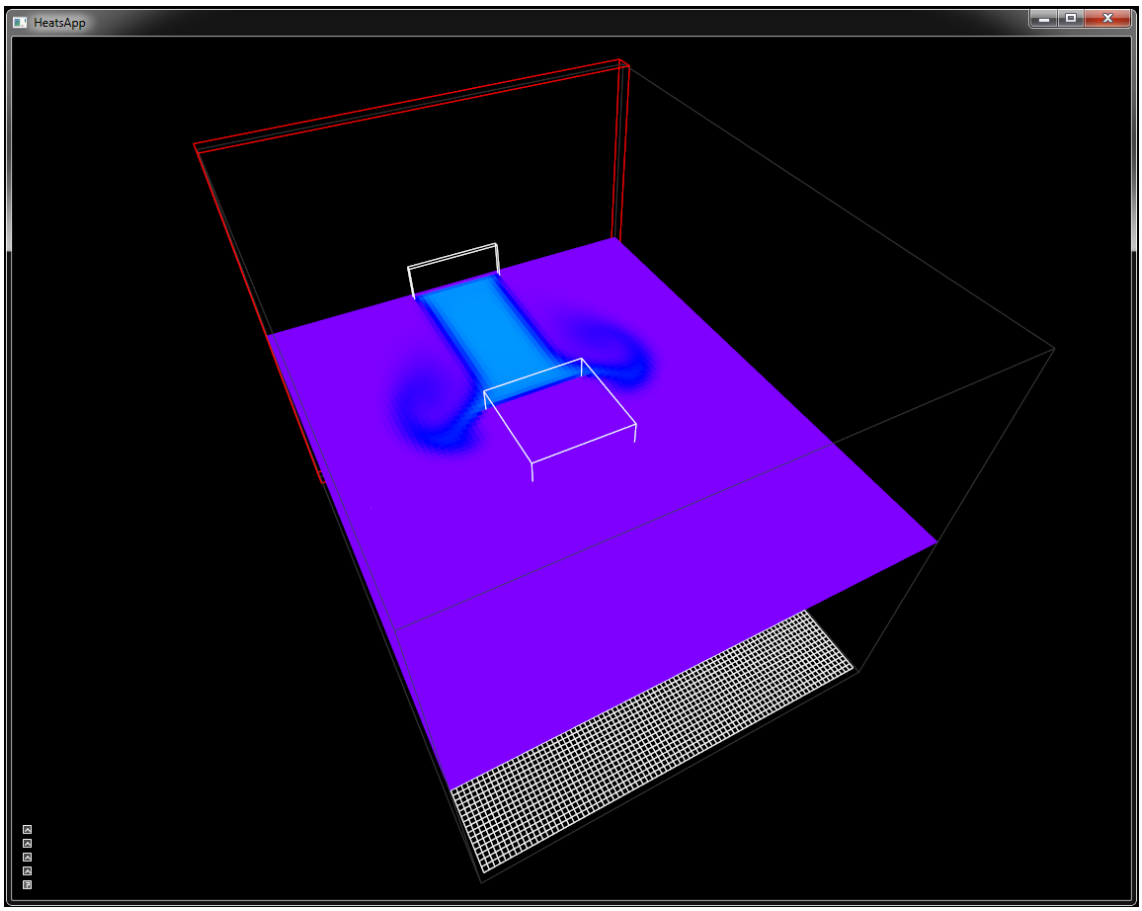

Figura 4.4: Sezione orizzontale dove si nota che il fluido è deviato lateralmente a causa dell'ostacolo.

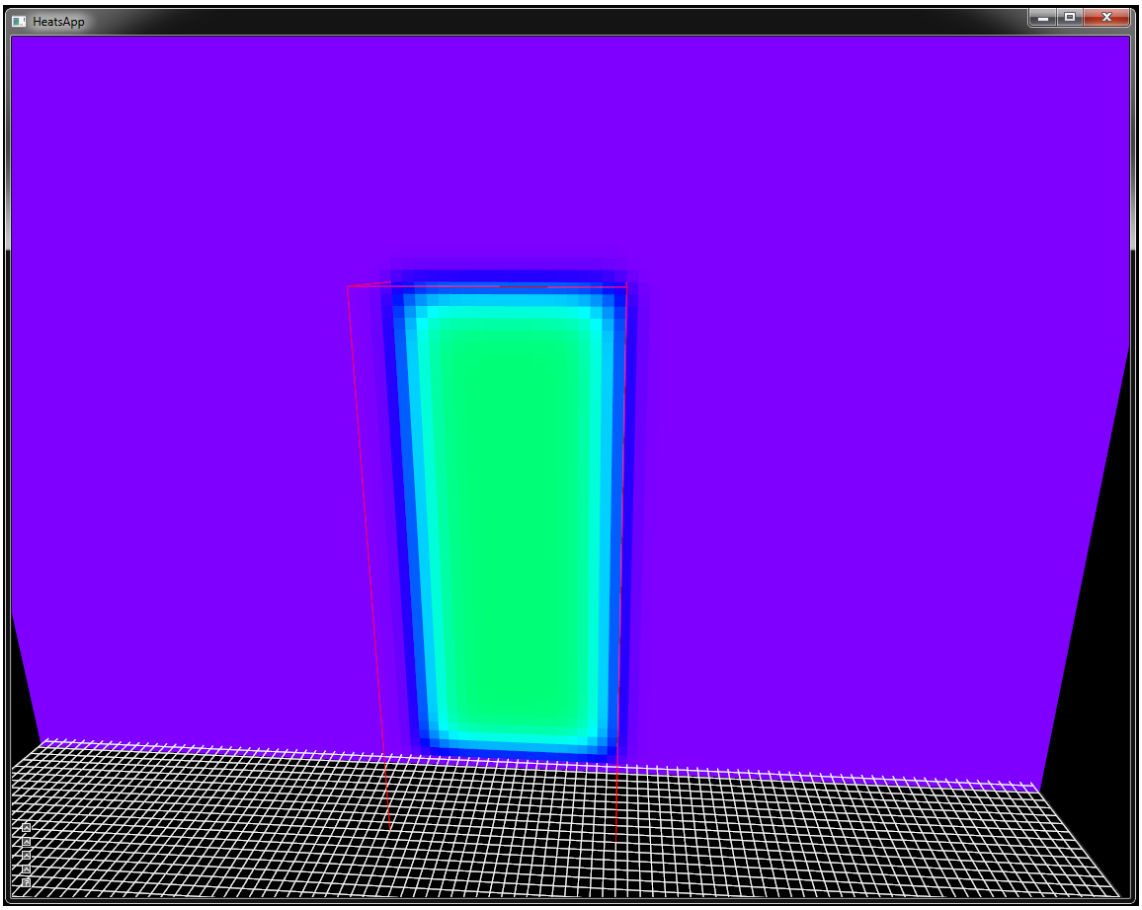

Figura 4.5: Ingrandimento della sezione verticale della temperatura dove è possibile notare lo stato termico del fluido in prossimità del corpo.

quella verde chiaro presente all'interno. Inoltre allontanandosi dal corpo il colore sfuma verso il viola che è la temperatura ambiente.

Questo comportamento sottolinea chiaramente che col passare del tempo il corpo e il fluido avendo temperature diverse tendono a raggiungere lentamente l'equilibrio termico.

4.4 Esperimento 4: Conduzione

Un altro interessante esperimento combina i precedenti prevedendo uno scenario in cui una grata di areazione soffia aria calda in direzione di un corpo composto di rame che è a temperatura ambiente. In questo modo è possibile verificare il corretto passaggio di calore dal fluido al corpo e poi tra i voxel contenuti nell'oggetto per conduzione.

Osservando la figura 4.6 si nota sia l'effetto di scorrimento dovuto alla presenza dell'ostacolo ma anche il passaggio di calore tra fluido e corpo con la conseguente diffusione tra le particelle contenute nell'oggetto. Il processo di simulazione mostra poi che gradualmente l'energia termica si diffonde fino ad arrivare all'estremità opposta a quella di contatto (figura 4.7).

4.5 Esperimento 5: Forze di galleggiamento

Un altro interessante esperimento riguarda le forze di galleggiamento che si vengono a creare a causa delle differenze di temperature tra le particelle del fluido. Lo scenario è lo stesso dell'esperimento tre ma il valore del coefficiente di espansione termica β relativo alla formula (2.17) non è nullo.

Osservando la simulazione si nota un interessante fenomeno che mostra che le particelle più eccitate sono quelle sui lati del corpo; questo comportamento è probabilmente dovuto al fatto che le forze di galleggiamento che si vengono a creare sommandosi generano una forte accelerazione verso l'alto (figura 4.8).

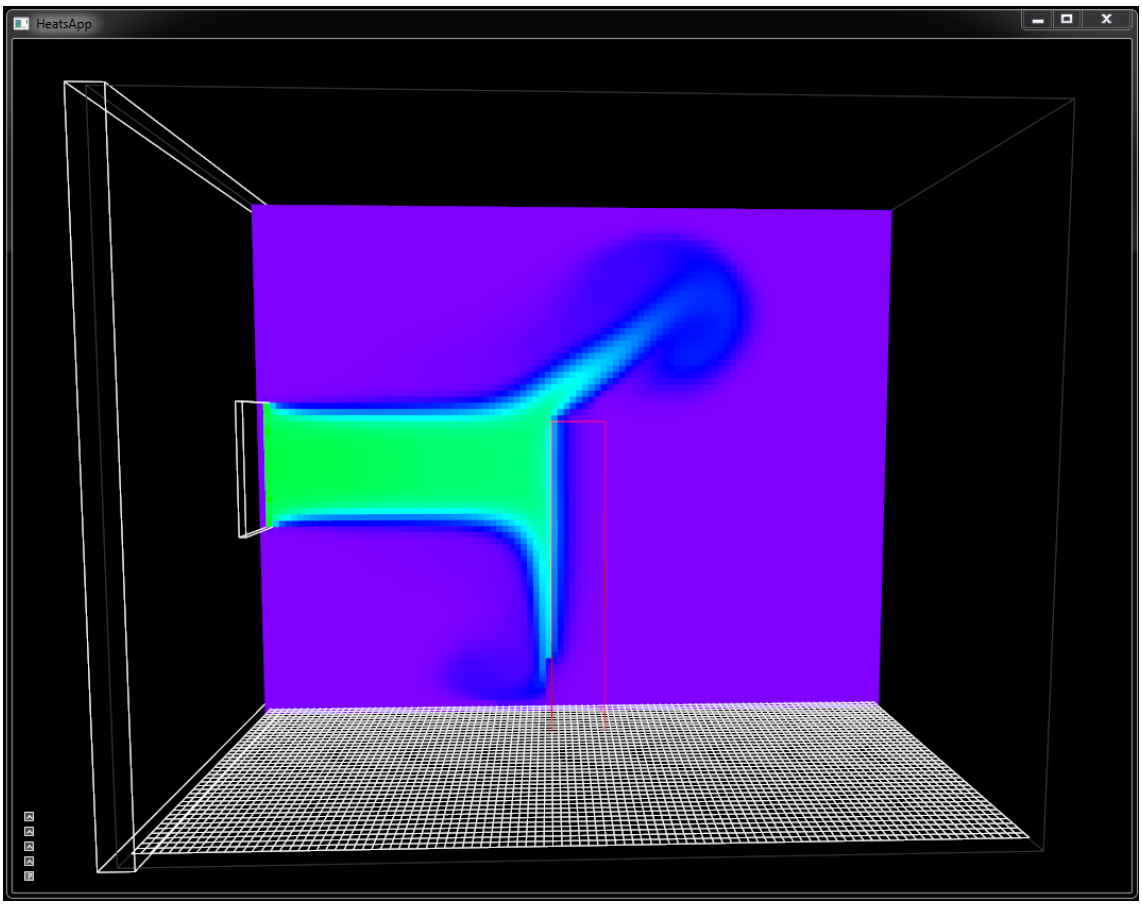

Figura 4.6: Pur essendo deviato il fluido trasferisce calore al corpo che comporta un aumento di temperatura superficialmente.

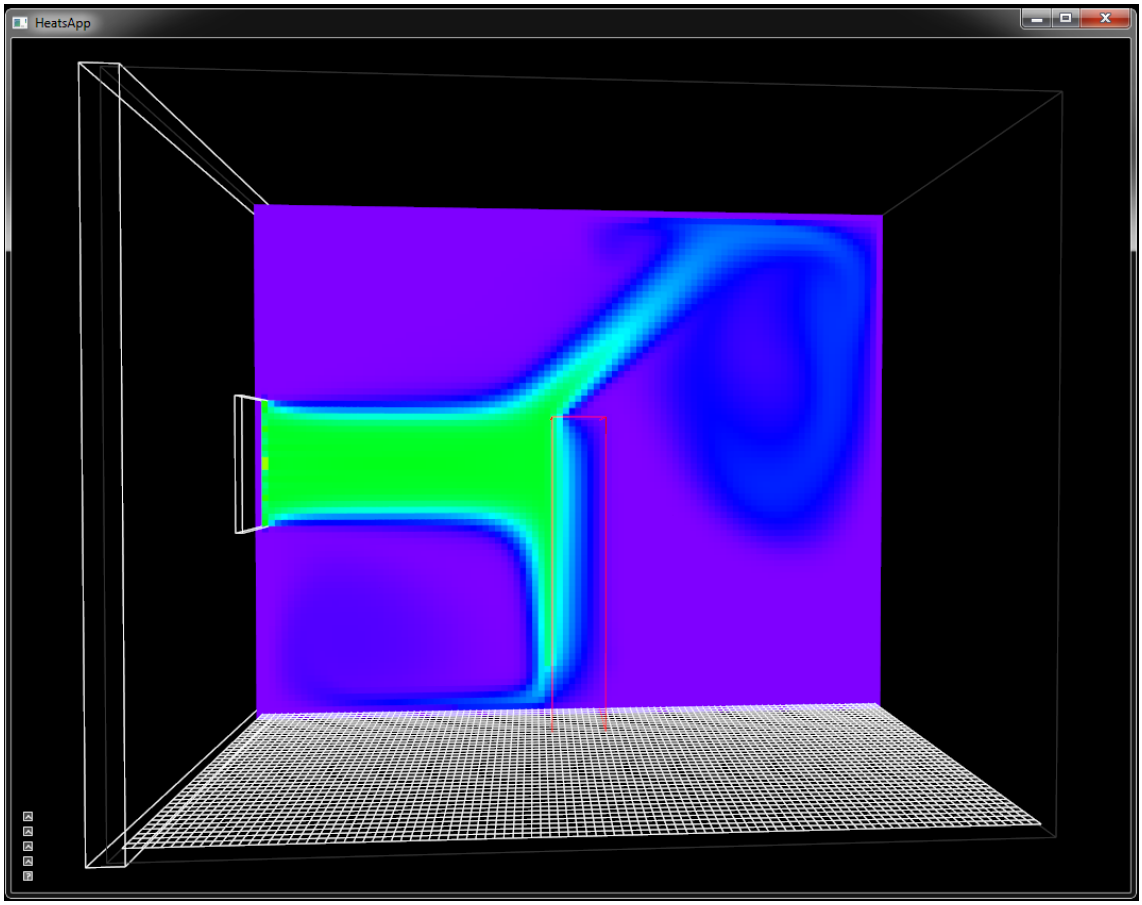

Figura 4.7: Il procedere della simulazione mostra che il calore scambiato inizialmente sulla superficie di contatto giunge all'estremità opposta del corpo.

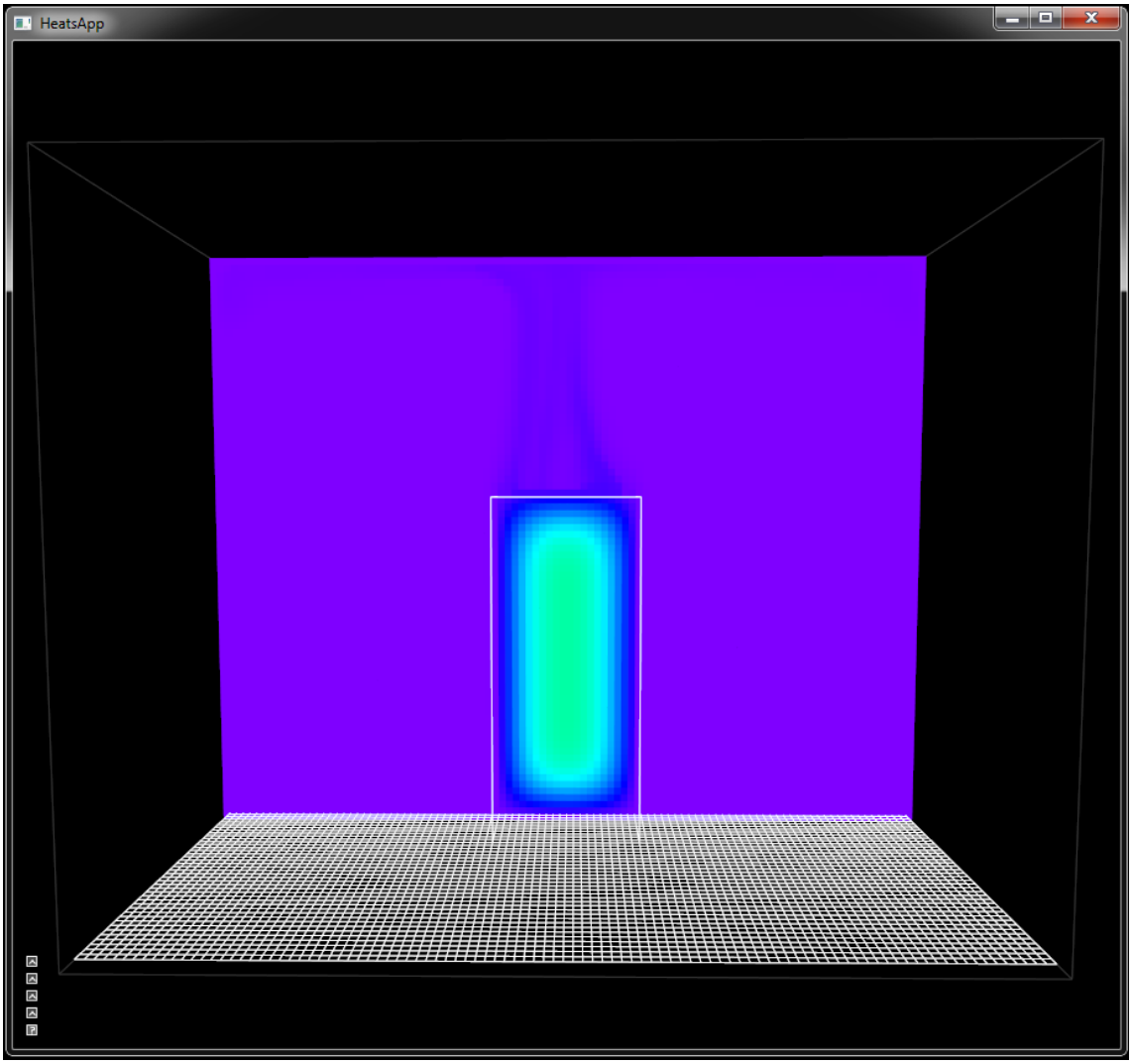

Figura 4.8: Le differenze di temperature tra i voxel generano delle forze che spingono l'aria calda verso l'alto.

4.6 Esperimento 6: Data Center

I precedenti esperimenti considerano degli scenari molto semplici ma sono la base per la costruzione di ambienti più complessi. Modellare un data center infatti non è semplice perché i risultati della simulazione sono fortemente determinati dalle scelte compiute per ricreare virtualmente un elemento del sistema reale in un certo modo piuttosto che in un altro. Si potrebbe pensare di rappresentare ciascuno armadietto rack come un unico parallelepipedo fatto del materiale del case del calcolatore e alimentato con corrente costante, oppure potrebbe essere opportuno avere ogni singolo rack come elemento posizionato dentro l'armadietto, comprendendo le zone di entrata e uscita dell'aria insieme alle ventole di areazione. Il livello di dettaglio che si vuole raggiungere quindi è chiaramente determinato dalle scelte compiute dall'utente che utilizza il simulatore in base alle proprie necessità.

Lo scenario di questo esperimento fa riferimento alla figura 1.3 ed è riprodotto in figura 4.9, dove al centro della stanza sono presenti gli armadietti contenuti in una struttura chiusa che presenta delle grate di areazione al centro del pavimento mentre lateralmente da parte opposta sono situati le unità CRAC. I rack sono fatti di alluminio e sono sottoposti ad un alimentazione costante di 260 Watt. Gli armadietti invece presentano delle grate anteriormente per far uscire l'aria spinta dalle ventole situate posteriormente. Il sistema di areazione contenuto nel CRAC invece è modellato in maniera semplice perché riprodurre tutti i componenti interni (in particolare il condensatore che si occupa di raffreddare l'aria) è troppo complicato, quindi si considera solo una grata di entrata frontale che risucchia l'aria dall'ambiente esterno con l'aggiunta di un volume d'aria che viene costantemente raffreddato e spinto verso il basso per simulare la ventola di areazione.

Uno dei possibili obiettivi della simulazione in questo caso potrebbe essere quello di studiare il posizionamento degli elementi nell'ambiente per evitare che si creino zone a temperatura troppo elevata. Analizzando la figura 4.10 infatti è possibile osservare che probabilmente è necessaria una distanza maggiore tra i CRAC e gli armadietti per evitare che si presenti un eccessivo

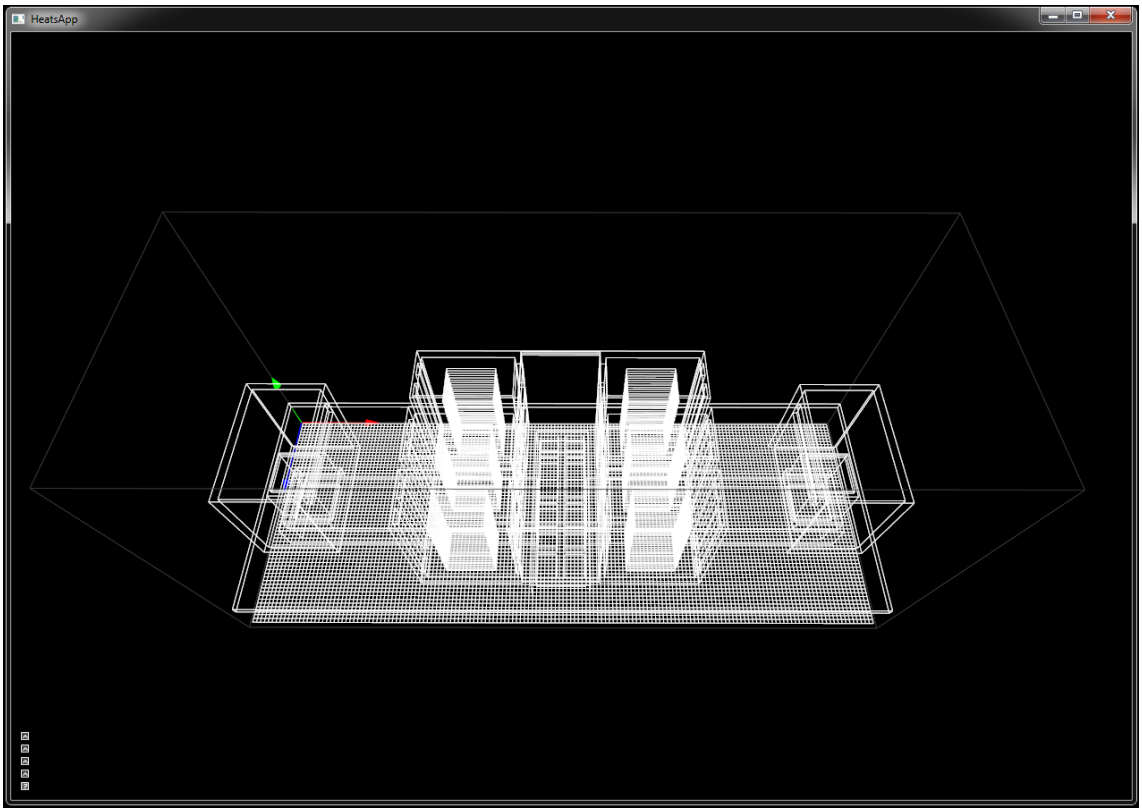

Figura 4.9: Lo scenario, al centro gli armadietti rack isolati lateralmente le unità CRAC.

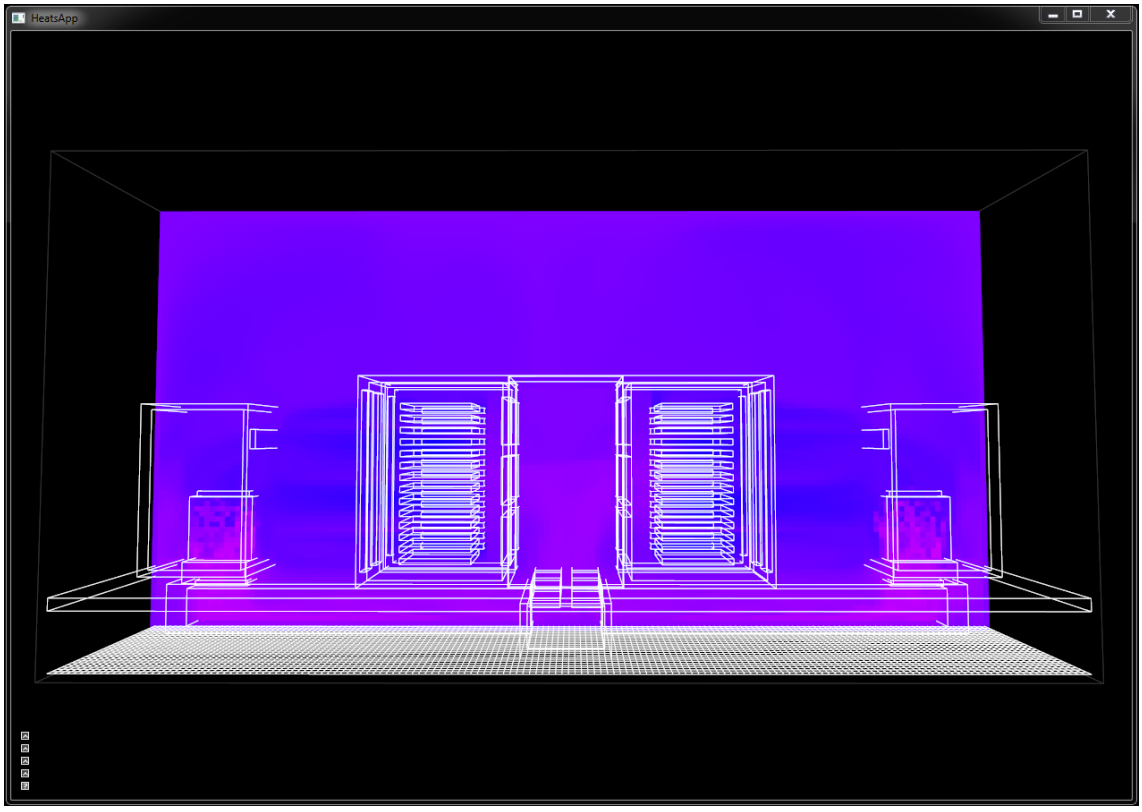

Figura 4.10: Lo stato termico del sistema dopo un'ora di simulazione.

calore tra i due, inoltre ridurre la dimensione delle grate frontali è un parametro che può influenzare questo fenomeno. Un altro dettaglio che emerge dall'analisi riguarda la struttura di contenimento e le ventole di areazione degli armadietti situate posteriormente: il flusso d'aria fredda proveniente dal basso non riesce ad arrivare ai rack in cima perché deviato dalle ventole inferiori; inoltre il volume d'aria sopra le grate poste sul pavimento non è completamente raffreddato, ridurre le velocità delle ventole potrebbe favorire la diffusione dell'aria fredda in tutta la struttura.

Conclusioni e Sviluppi Futuri

In questa tesi si è considerato il problema della propagazione del calore in riferimento ai data center. La teoria matematica sui fluidi e le leggi relative alla propagazione del calore hanno permesso l'effettiva implementazione di un simulatore in grado di riprodurre scenari reali. Le applicazioni di questo strumento sono molteplici e variano in base all'utilizzo che l'utente vuole farne. Tuttavia sussistono delle problematiche insite nel modello matematico scelto per la rappresentazione del campo vettoriale ed esse non vanno sottovalutate se si vuole raggiungere un livello di precisione più elevato in rapporto ai risultati prodotti.

Il lavoro prodotto però è costituisce solo la base del simulatore ed infatti gli sviluppi futuri sono veramente numerosi:

- **Griglia sfalsata**
  Valutare le componenti della velocità sui centri delle facce dei voxel e le relative modifiche delle formule per il calcolo vettoriale possono migliorare la precisione dei calcoli inerenti all'avvezione, diffusione e proiezione.

- **Modello Lagrangiano**
  Un corpo è composto di molecole che in base alla loro energia cinetica, vibrando e urtando tra loro, regolano lo stato termico del sistema. Un alternativa molto interessante è data dal modello lagrangiano che prevede un modello in cui il fluido è rappresentato tramite delle particelle libere di muoversi nello spazio. Questo approccio anche se più

realistico è più complicato da realizzare e probabilmente richiederebbe una completa revisione della teoria dei fluidi.

- **Calcolo su scheda grafica**
  L'implementazione attuale esegue i calcoli relativi alla simulazione solo sul processore ma in realtà è possibile parallelizzare gli algoritmi e sfruttare la scheda grafica. Questo approccio notevolmente differente prevede l'utilizzo di *shaders* che contengono del codice eseguibile solo sul processore della scheda grafica (GPU) che attualmente contiene un numero di unità di calcolo (core) nell'ordine delle migliaia. Ogni *shader* quindi verrebbe eseguito in maniera parallela e indipendente su ciascun core raggiungendo cosi un notevole miglioramento delle prestazioni che permetterebbe di simulare molto più velocemente gli scenari. Questa implementazione però sostituirebbe gli array utilizzati per i campi con delle texture tridimensionali che una volta calcolate possono essere rappresentate direttamente.

- **Strumenti di modellazione**
  Attualmente il simulatore prevede solo dei parallelepipedi rettangoli per rappresentare gli elementi presenti nell'ambiente. Un possibile sviluppo futuro potrebbe considerare inizialmente delle forme primitive semplici (es. sfera, piramide, toro, ...) e in seguito strutture più articolate come le mesh. In questo modo però la fase che trasforma le forme in voxel può diventare più complessa.

- **Versione plugin**
  Attualmente i programmi per la modellazione tridimensionale possono essere migliorati tramite delle estensioni (plugin) che aggiungono delle nuove funzionalità. Un possibile sviluppo futuro potrebbe essere quello di sviluppare una versione plugin del simulatore per inserirlo effettivamente in un'applicazione che mette già a disposizione gli strumenti per modellare con semplicità l'ambiente e gli elementi presenti nello scenario.

- **Versione mobile**

  La forte diffusione di smartphone e dispositivi mobili dotati di processori multi-core e memoria nell'ordine dei Gigabyte indica che potrebbe essere possibile sviluppare una versione mobile del simulatore anche su questo genere di calcolatori, infatti non è da sottovalutare la capacità di poter utilizzare il simulatore in qualsiasi occasione.

- **Fisica**

  Gli oggetti rappresentabili nel simulatore sono tutti statici, non possono muoversi durante la simulazione. La capacita di poter rappresentare dei corpi in grado di muoversi liberamente durante la simulazione potrebbe aggiungere un grado di realismo e dinamicità anche se ciò complicherebbe sicuramente la simulazione.

Bibliografia

[1] Jos Stam. *Real-Time Fluid Dynamics for Games.*

[2] Jos Stam. *Stable Fluids*, 1999.

[3] Ronald Fedkiw, Jos Stam, Henrik Wann Jensen. *Visual Simulation of Smoke.*

[4] Nick Foster, Dimitris Metaxas. *Modeling the Motion of a Hot, Turbulent Gas.*

[5] Mark J. Harris. *Fast Fluid Dynamics Simulation on the GPU*, 2004.

[6] Keenan Crane, Ignacio Llamas, Sarah Tariq. *Real-Time Simulation and Rendering of 3D Fluids.*

[7] Mark J. Harris, William V. Baxter III, Thorsten Scheuermann, Anselmo Lastra. *Simulation of Cloud Dynamics on Graphics Hardware*, 2003.

[8] Jonas Meyer. *Interactive Real-Time Simulation of Smoke*, Master Thesis 2005.

[9] Michael Ash. *Simulation and Visualization of a 3D Fluid*, Master Thesis 2005.

[10] Marinus Rorbech. *Real-Time Simulation of 3D Fluid Using Graphics Hardware*, Master Thesis 2005.

[11] Michael Griebel, Thomas Dornseifer, Tilman Neunhoeffer. *Numerical Simulation in Fluid Dynamics*, 1998.

[12] Chandrakant D. Patel, Amip J. Shah. *Cost Model for Planning, Development and Operation of a Data Center*, 2005.

[13] Kailash C. Karki, Amir Radmehr, and Suhas V. Patankar. *Use of Computational Fluid Dynamics for Calculating Flow Rates Through Perforated Tiles in Raised-Floor Data Centers.*

[14] Jeffrey Rambo and Yogendra Joshi. *Modeling of data center airflow and heat transfer: State of the art and future trends*, 2007.

[15] Richard N. Taylor, Nenad Medvidovi?, Eric M. Dashofy. *Software Architecture: Foundations, Theory, and Practice.* John Wiley & Sons, Inc., 2009.

[16] Andrew Bell, Hai Nguyen. *Libreria Cinder*, http://libcinder.org/

[17] Lee Thomason. *Libreria tinyxml2*, https://github.com/leethomason/tinyxml2

[18] David Tschumperle. *Libreria CImg*, http://cimg.sourceforge.net/

[19] Leslie Lamport. *LaTeX: a document preparation system.* Addison-Wesley, 1994.

[20] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, 1994.

Ringraziamenti

Ringrazio la mia famiglia che mi ha supportato in tutto e per tutto e le persone che ho conosciuto in questo periodo di duro studio e con cui ho condiviso grandi emozioni.