Costruzione di Interfacce

Scene dinamiche e Collision detection

Fabio Ganovelli
fabio.ganovelli@isti.cnr.it

Scene dinamiche

- Scene in cui gli oggetti si possono muovere
- Molte applicazioni:
  - chirurgia virtuale
  - giochi
  - animazione
  - fenomeni naturali
  - progetti di C.I.
- Ma è grafica?
  - Non proprio, è un campo interdisciplinare (fisica, robotica, ingegneria, informatica)...
  - ...ma i grafici se ne occupano parecchio

Scene dinamiche

- Due problemi:
  - modellare il comportamento di un oggetto sottoposto a forze
  - modellare l’interazione tra più oggetti
- Cosa affrontiamo oggi:
  - simulazione interattiva: come è organizzato il codice
  - Il minimo indispensabile di fisica
  - Soluzioni fondamentali per modellare l’interazione tra oggetti

Il codice

```
init(){ t = 0;// il tempo corrente}
void main_cycle(){
   simulate(dt); /* una funzione che  calcola tutto quello
   che succede nei prossimo dt di tempo (da t a t+dt) */
   render(); /* la funzione che fa il rendering della
                    scena */

   t=t+dt;  /* avanzamento del tempo */
}
```

- brutto! Il tempo avanza con velocità dipendente dalla CPU

Il codice: temporizzazione

```
init(){ t = 0;}

void main_cycle(){
   dt=clock() - t;
   simulate(dt);
   render();
   t=t+dt;
}
```

- bene, purchè l'esecuzione di simulate(dt) non duri più di dt !

! Il Frame Rate è sacro !

Lo stato

- Stato di un oggetto:
  - il minimo:
    - posizione p = (x,y,z)
    - velocità v = (vx,vy,vz)
  - qualcosa in più:
    - accelerazione a = (ax,ay,az)
  - un po' più completo:
    - massa m
    - forza agente f = (fx,fy,fz)

Un passo di simulazione

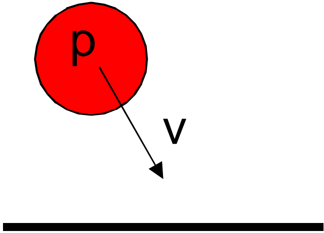
```
void simulate(dt){
for (ogni oggetto "obj" nella scena)
   {
      obj.p = obj.p + dt * obj.v;
      obj.v = obj.v + dt * obj.a;
   }
.....
// >> qui cosa ci mettiamo? <<
}
```

Es: obj.a costante per tutti gli oggetti obj.a =(0,0,-9.8)

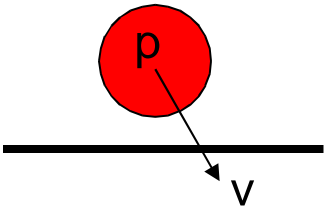
Interazione tra oggetti

- Se un oggetto si muove può andare a sbattere
  - contro parti statiche della scena (es: un muro)
  - contro parti dinamiche della scena (es: un altro oggetto)
- Il disegno di algoritmi e strutture dati per rilevare quando questo accade si chiama Collision Detection
- E quando l'abbiamo rilevato?
- lo stato dell'oggetto deve essere modificato di conseguenza: Collision Response

Esempio:



La palla si trova in posizione p
e ha velocità v



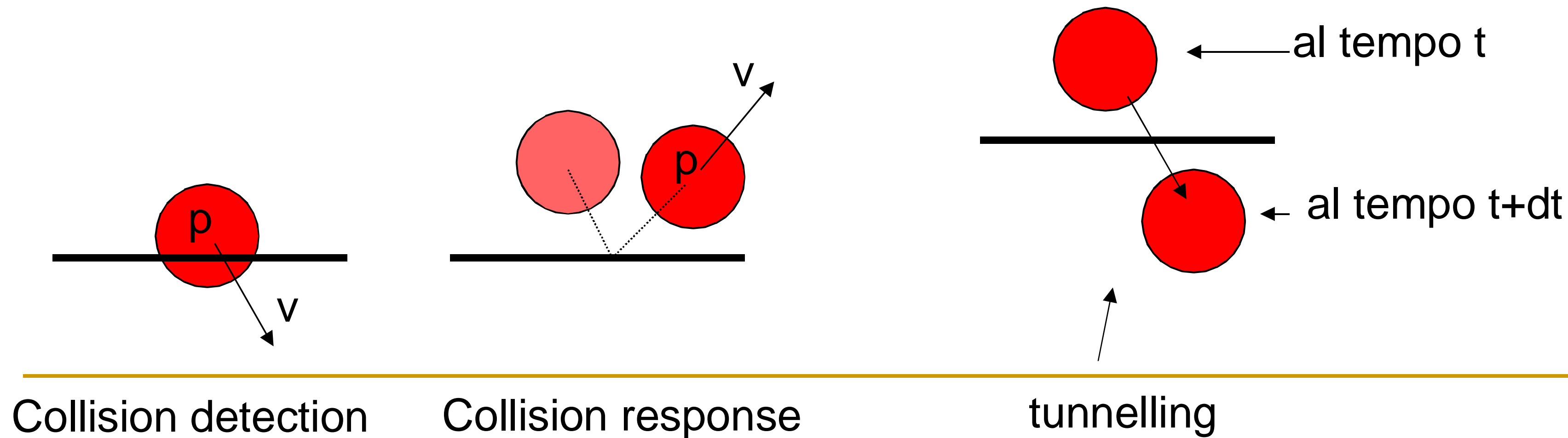
Dopo un passo di simulazione la
palla è più vicina al terreno

La palla raggiunge il terreno.
Ce ne accorgiamo (CD)
Cambiamo lo stato “riflettendo”
la velocità (CR)

Collision detection Collision response

Così sarebbe bello!

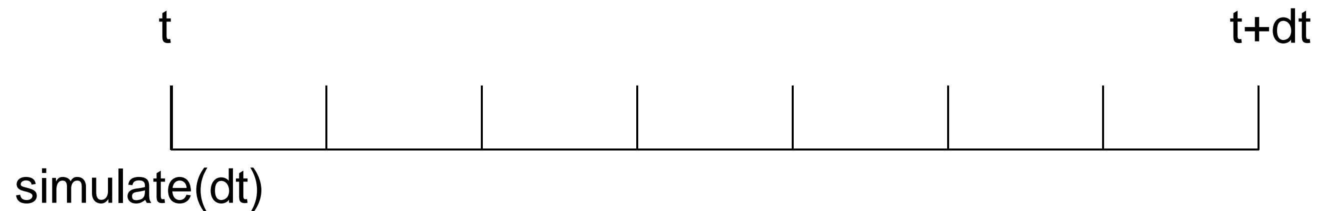
- La simulazione procede per intervalli discreti
- Le collisioni avvengono in un momento qualsiasi
- Quindi:
  - gestire la collision response diventa più complicato
  - rilevare le collisioni diventa più difficile



Collision detection   Collision response   tunnelling

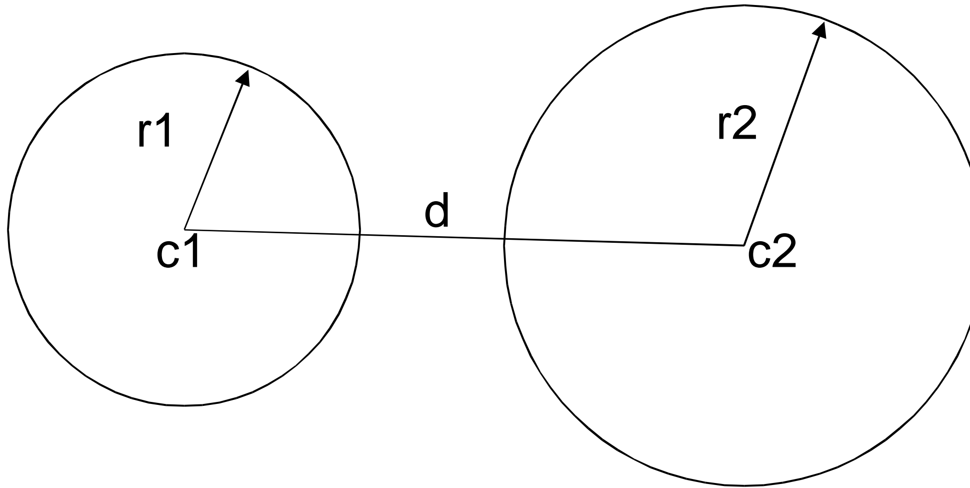
Soluzione brute force

- dividiamo l’intervallo dt in parti più piccole e ignoriamo il problema



Collision Detection tra due oggetti

- caso facile: i due oggetti sono sfere



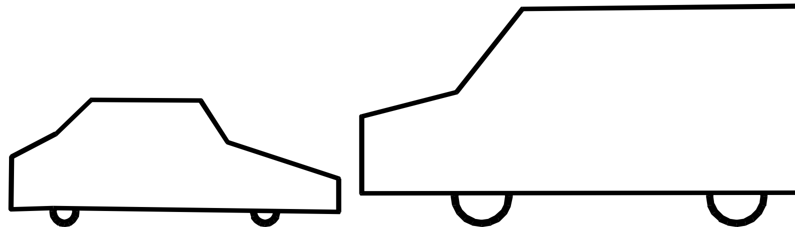
se $d < r1+r2$ c'è intersezione tra le sfere

$(c2-c1)*(c2-c1) < (r1 + r2)^2$

- Più in generale i casi in cui il bordo degli oggetti si esprime analiticamente in forma chiusa con una funzione semplice (piani, sfere, ellissoidi..)

Collision Detection tra due oggetti

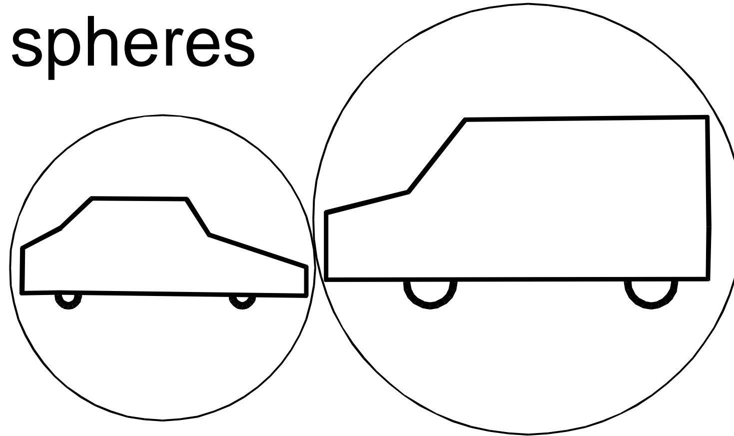
- Gli oggetti sono descritti da poligoni



- Brute force: si testano tutte le coppie di poligoni
- costo = n1*n2*c
  - n1 numero di poligoni del primo oggetto
  - n2 numero di poligoni del secondo oggetto
  - costo di un test di intersezione tra due poligoni
- male! Il costo è quadratico nel numero di poligoni!

Bounding Volumes

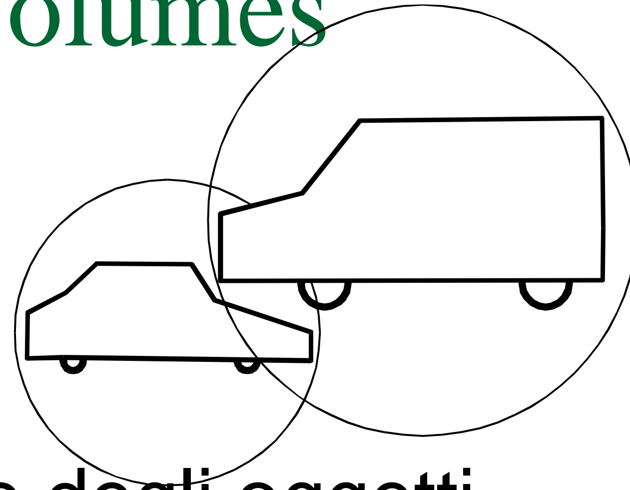
- Un concetto fondamentale: Bounding Volume
  - Il BV di un oggetto è una primitiva geometrica per cui il test di intersezione sia efficiente (tempo costante) e contenente l’oggetto
- ES: bounding spheres



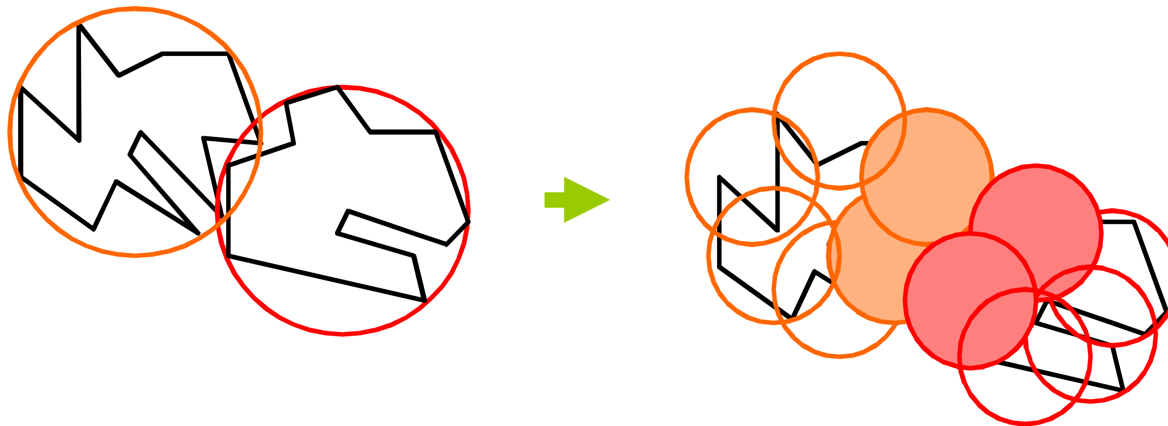
- L’idea: se i bounding volumes di due oggetti non si intersecano, nemmeno gli oggetti si intersecano

Gerarchie di bounding volumes

- ...e se i BV si intersecano?

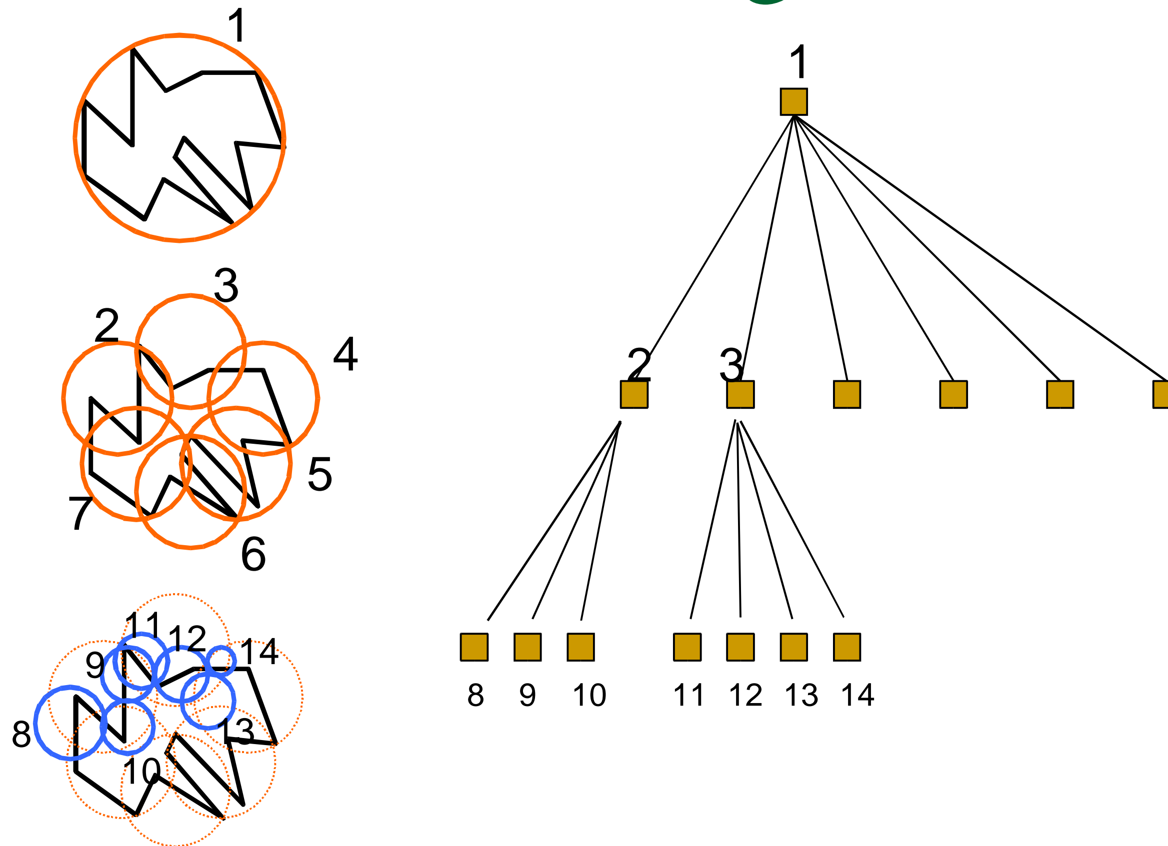


- usiamo i BV di una partizione degli oggetti



- ...ricorsivamente.

Gerarchie di bounding volumes



- i poligoni vengono testati solo quando in bounding volumes “foglia” si intersecano

Gerarchie di Bounding Volumes

- Molte varianti:
  - Che tipo di Bounding Volumes?
    - sfere, axis aligned bounding boxes (AABB), object oriented bounding boxes (OOBB)....
  - Che tipo di gerarchia?
    - bin-tree, oct-tree, K-DOP...
  - Come si costruisce?
  - Come si visita l'albero durante la CD?
    - depth first, breadth first, priority scheduling

Costo

$$T_c = N_v*C_v + N_n*C_n + N_s*C_s$$

v : nodi visitati

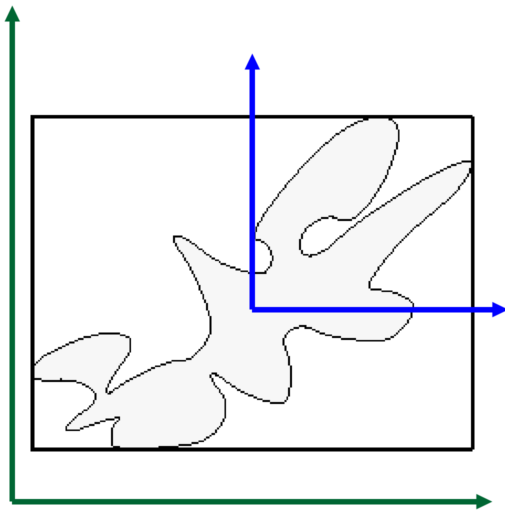
n : coppie di bounding volumes testati

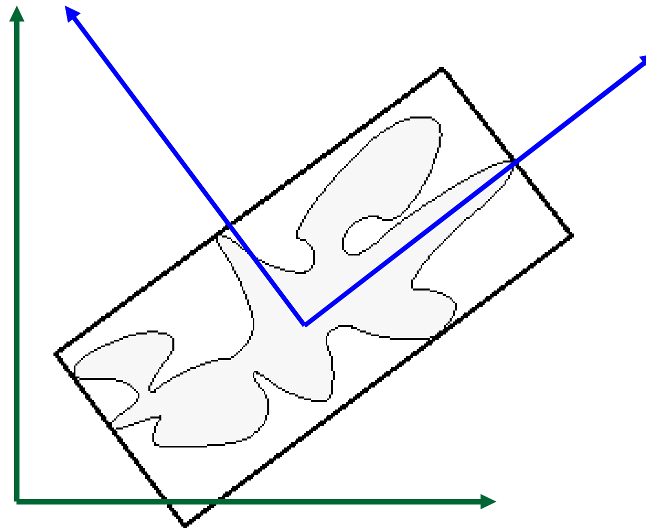
s : coppie di poligoni testati

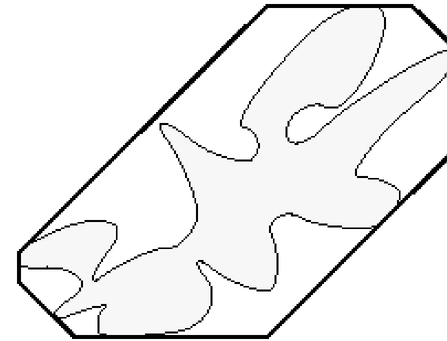
N: numero di

C: costo di

BV: non solo sfere







AABB

il BV è un box con le facce parallele ai piani nel sistema di riferimento dell'oggeto

OBB

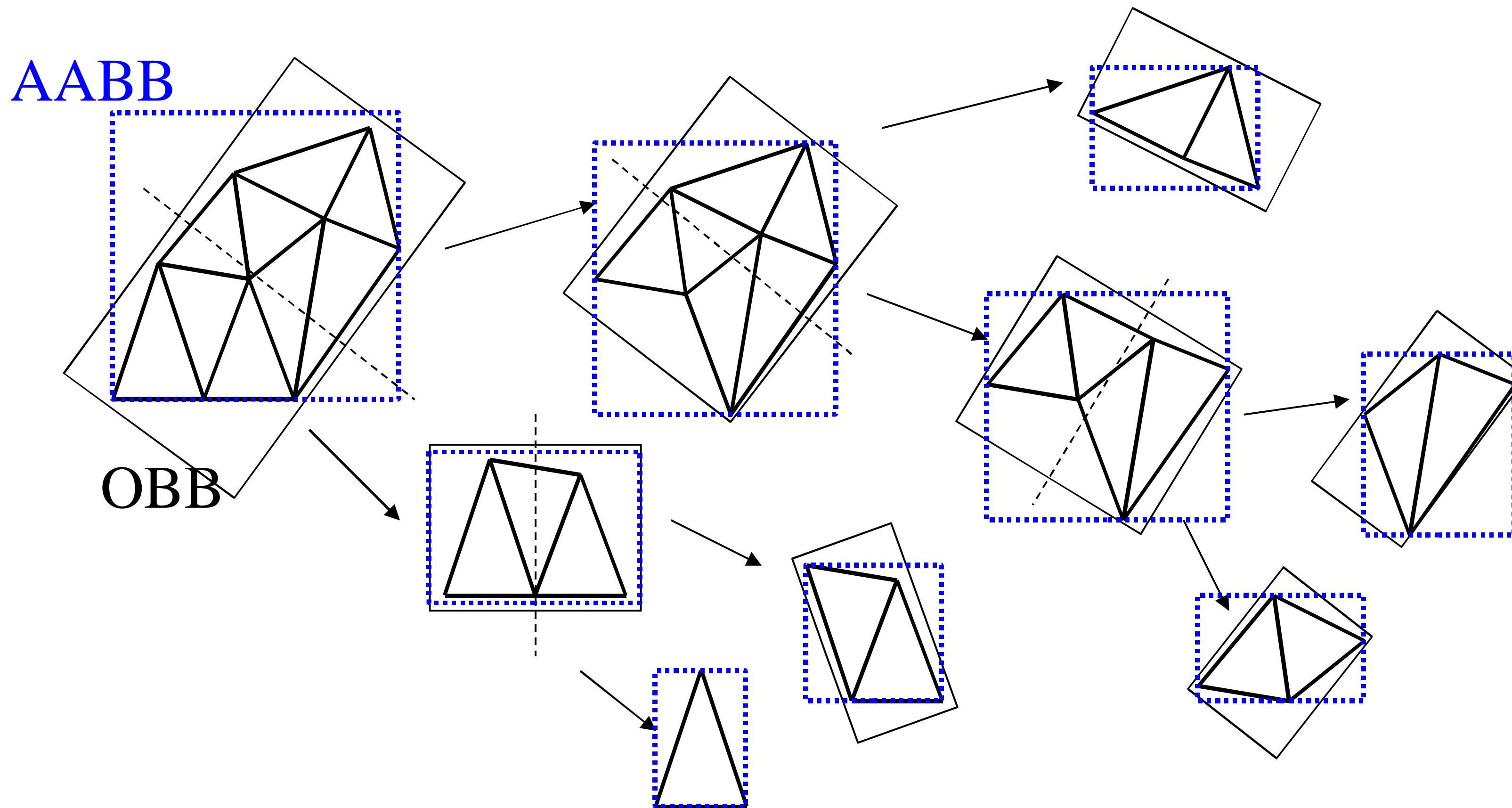
il BV è un box con le facce parallele a un sistema di riferimento ottimale

k-DOP

il BV è un poliedro convesso con le facce parallele a un insieme di k piani prefissato

Cosa scegliereste in questo caso?

Esempio: AABB vs OBB



Uso della gerarchia

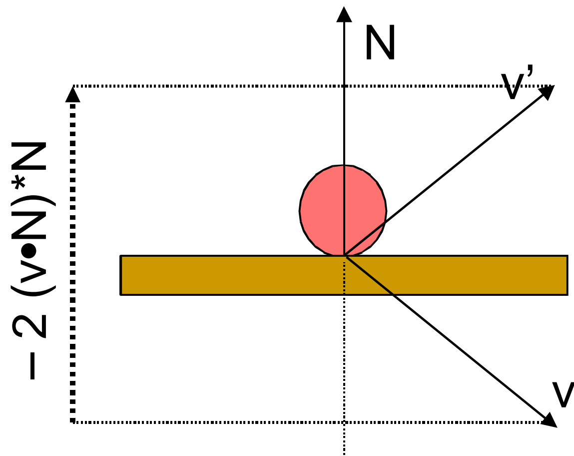
```
deque< pair<Node,Node> > pairs_to_test;
// coda delle coppie di nodi da testare. All'inizio contiene solo la
// coppia dei nodi radice delle due gerarchie

bool collision_detection(){
while( ! pairs_to_test.empty() ) {
pairs_to_test.pop_front();
...// assegna la coppia di valori a n1 e n2
if( n1 e n2 sono nodi foglia)
   return test_all_pairs( polygons_of(n1),polygons_of(n2));
   // test_all_pair testa "brute force" tutte le coppie di poligoni
   contenute nei nodi n1 e n2
else
    {
         if(BV(n1) > BV(n2) )
           for( tutti  n_i   figli di n1)
               pairs_to_test.insert( pair(n_i,n2) );
         else
           for( tutti  n_i   figli di n2)
               pairs_to_test.insert( pair(n1,n_i) );
   }
}
return false;
}
```

Ingredienti per implentare uno schema

- Saper calcolare il BV di un insieme (di poligoni, di altri BV)
- Saper calcolare se due BV si intersecano (già visto per le sfere)
- Saper visitare la gerarchia

Collision response

- La risposta dipende dal modello fisico che si sta usando.
- Caso più semplice:



v' = v – 2 (v•N)*N

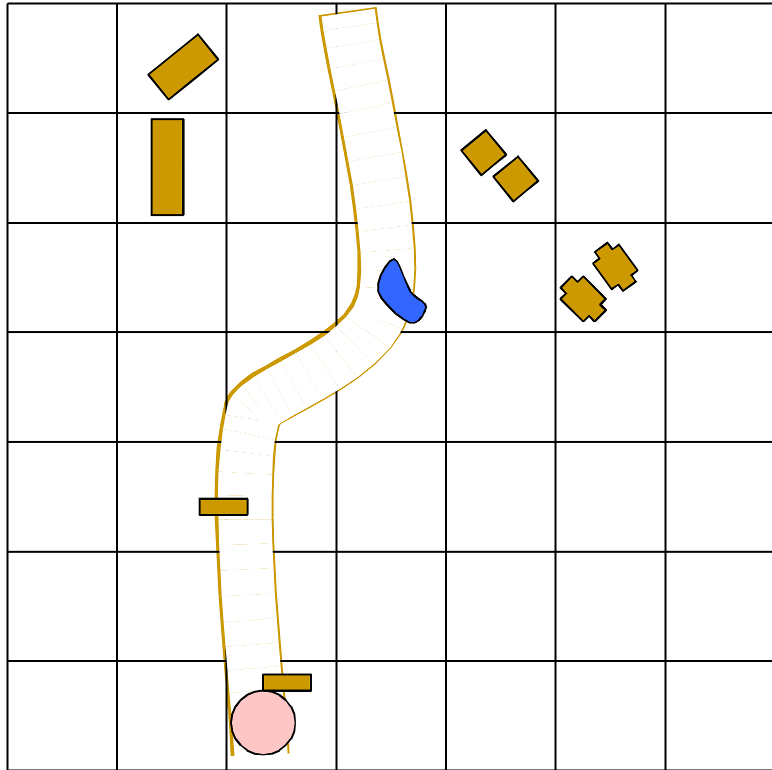
v: velocità prima dell'urto
v': velocità dopo l'urto
N: normale della superficie nel punto di contatto

- ci fermiamo qui..

CD e CR tra un oggetto e la scena statica

- Oss 1: la scena non è altro che un oggetto fermo!
- Oss 2: il BV della scena contiene tutto
- possiamo indicizzare lo spazio e risparmiarci un bel pò di conti

CD e CR tra un oggetto e la scena statica



- lo spazio è suddiviso in una griglia uniforme
- ogni cella contiene una lista di puntatori agli oggetti (della scena) che la intersecano
- data la posizione dell'oggetto in movimento (es: la macchinina) è facile calcolare le celle che interseca
- si testa l'intersezione solo con gli oggetti contenuti in tali celle

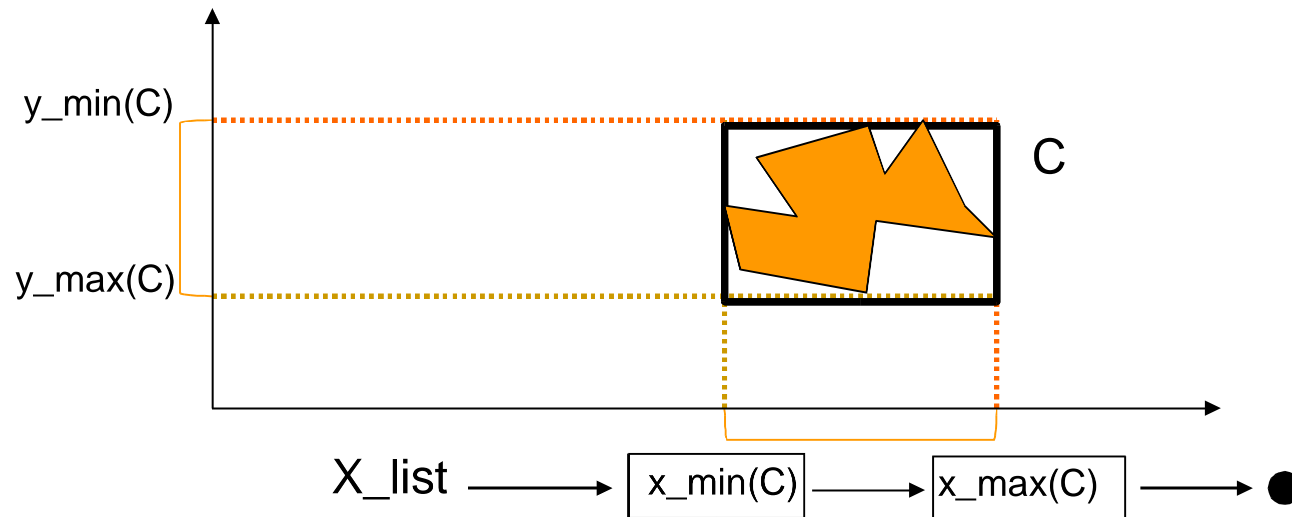
Collision Detection tra molti oggetti

- ..fin qui sappiamo trattare in modo efficiente la CD tra due oggetti e tra un oggetto e la scena
- ..e se ho tanti oggetti che si muovono?
- brute force: proviamo a testare tutte le coppie di oggetti
- costo : n * n, n numero di oggetti
- Doh! siamo daccapo..

Sweep and Prune

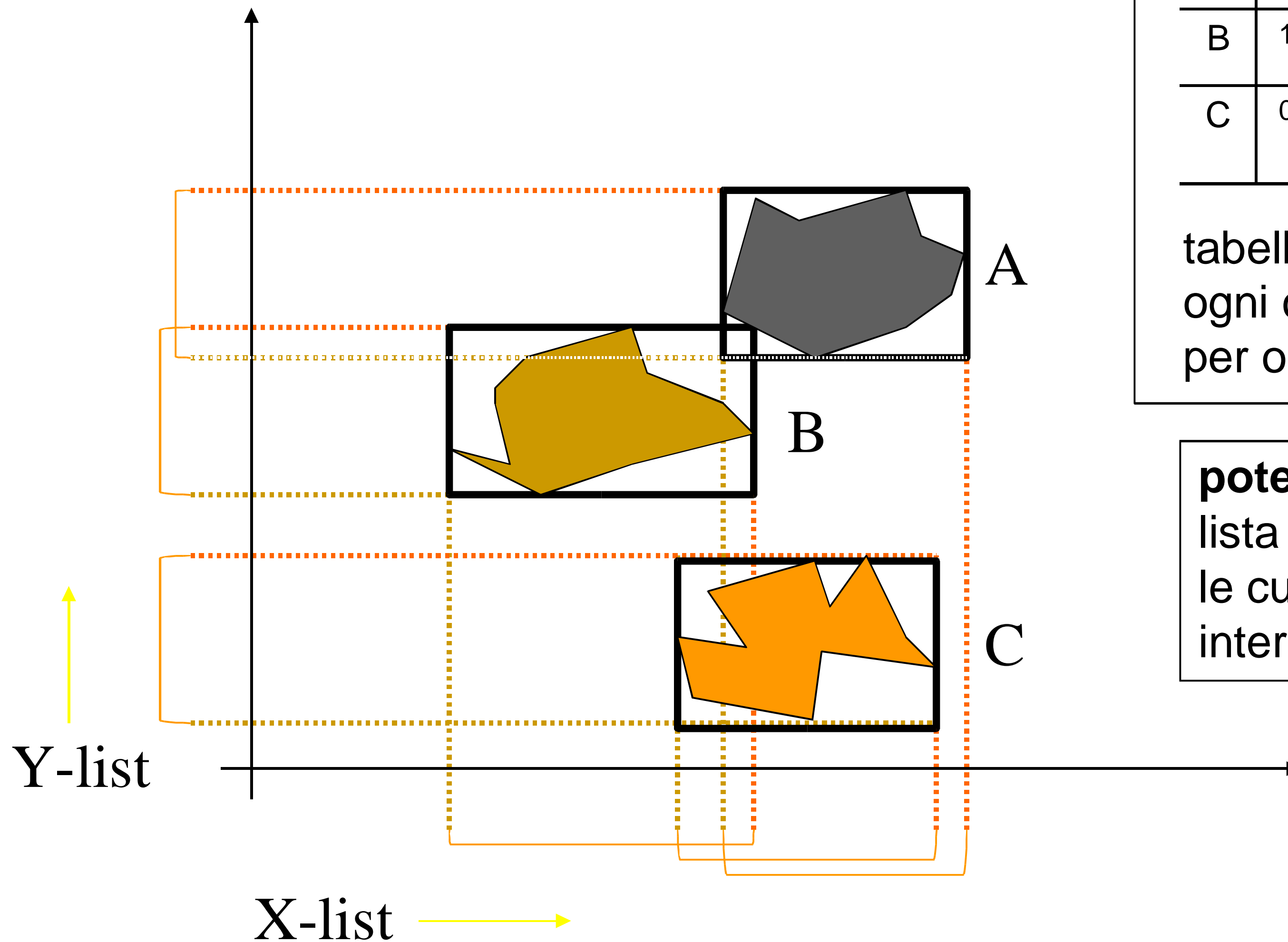
- Oss 1: due box allineati con gli assi si intersecano sse le loro proiezioni sugli assi x,y e z si intersecano
- L'idea:
  - Usiamo dei box allineati con gli assi come BV
  - Proiettiamo i BV sugli assi
  - Usiamo Oss 1 per decidere quali sono le coppie da testare
- fin qui niente di nuovo..

Sweep and Prune



- ogni proiezione su un asse è un intervallo
- teniamo una lista per ogni asse
- la lista contiene tutti gli estremi degli intervalli
- ad ogni passo la lista viene ordinata

Sweep and Prune



tabella

xy	A	B	C
A	-		
B	11	-	
C	01	01	-

tabella di grandezza n*n
ogni cella contiene un bit
per ogni asse

potential_collision
lista di coppie
le cui proiezioni si
intersecano in tutti gli assi

S&P: Insertion Sort

```
void InsertionSort_X(){
for( i = 1; i < N;++i)
 {
    if( x[i] < x[i-1])
     {
        i1=i-1;
          while( (i1>0) and (x[i1]<x[i1-1])
         {
           swap(i1,i1-1);
           i1--;
         }
     }
 }
}
```

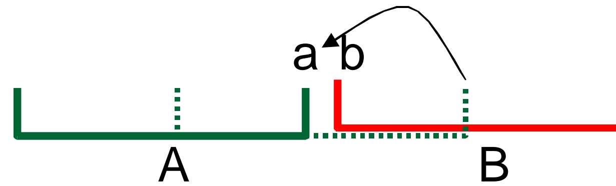
Esempio:

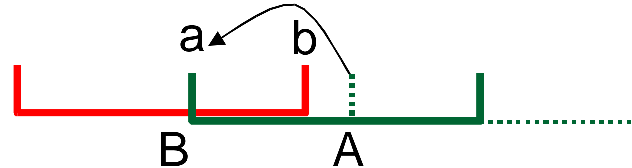
X = 3,5,7,13,2 i=4
X = 3,5,7,2,13 i1= 3
X = 3,5,2,7,13 i1= 2
X = 3,2,5,7,13 i1= 1
X = 2,3,5,7,13

- La complessità è O(n^2) nel caso pessimo e nel caso medio
- Però è O(n) se sussiste Frame to Frame Coherency
- ....che è proprio il nostro caso...
- la tabella viene aggiornata durante l'insertion sort, nella funzione swap(i1,i1-1)

S&P: tabella e potential_collision

```
void swapX(a,b){
// A è il box che proietta a
// B è il box che proietta b
if(a è un massimo e b è un minimo)
{
  tabella(A,B).bit_x = 0;
  remove_from_potential_collision(A,B);
}

else

if(a è un minimo e b è un massimo)
  tabella(A,B).bit_x = 1;
    if(tabella(A,B).bit_x && tabella(A,B).bit_y))
      add_to_potential_collision (A,B);
 }
}
```



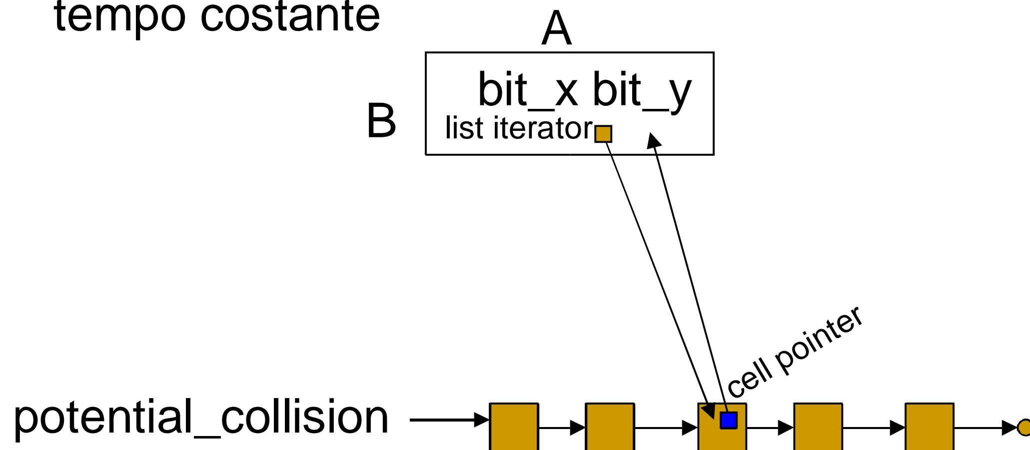


- `add_to_potential_collision` e `remove_from_potential_collision` operano in tempo costante
- `potential_collision` NON viene azzerata da uno step all'altro
- stessa implementazione per tutti gli assi

S&P: gestione di potential_collision

- `add_to_potential_collision` aggiunge un elemento `(A,B)` sse non è già presente
- `remove_to_potential_collision` rimuove un elemento `(A,B)` sse è presente

- possibile implementazione:
  - `potential_collision` è una lista di puntatori a cella di tabella (= coppia)
  - ogni cella di tabella contiene un iteratore di tale lista. Vuoto se non è presente (o usate un valore booleano, come credete)
- Così inserzione (che va bene in cima alla lista) e rimozione si fanno in tempo costante



Riassumendo

- S&P ha un tempo lineare nel caso di Frame to Frame coherency
- restituisce la lista di oggetti i cui bounding box si intersecano
- da lì si può procedere ai test per coppie
- Questa fase di "scrematura" viene riferita come Broad Phase Collision Detection, mentre quella tra coppie Narrow Phase CD

Conclusioni

- nella vita reale (il progettino) vi servirà molto meno di quello che si è visto
- Una Bounding Sphere intorno al vostro omino, macchinina, palla basta e avanza
- Se avete uno scene graph potete usare direttamente quello come gerarchia e associate il BV come membro del nodo
- se volete cimentarvi in qualcosa di più chiedete pure aiuto

Riferimenti bibliografici

- [AABB TREE} *Efficient Collision Detection of Complex Deformable Models using AABB Trees* Gino van den Bergen Journal of Graphics Tools: JGT , 1998 http://citeseer.ist.psu.edu/vandenbergen98efficient.html
- [OBB TREE]*OBB-Tree: A Hierarchical Structure for Rapid Interference Detection ,* S. Gottschalk, M.C. Lin, D. Manocha , Computer Graphics, 1996 , http://citeseer.ist.psu.edu/gottschalk96obbtree.html
- [Sweep and Prune] *I-{COLLIDE}: An Interactive and* Exact *Collision Detection System for Large-Scale Environments, Jonathan* D. Cohen and Ming C. Lin and Dinesh Manocha and Madhav K. Ponamgi, Symposium on Interactive 3D Graphics , http://citeseer.ist.psu.edu/cohen95icollide.html
- [Sphere tree] *Collision detection for animation using sphere-trees I.J.Palmer* & R.L.Grimsdale, Computer Graphics Forum, 14(2), 1995, pp105-116
- altre slides su CD: http://flender.netfarm.it/~guyver1/Didattica/RTM/Lezioni/RTM20020419.zip