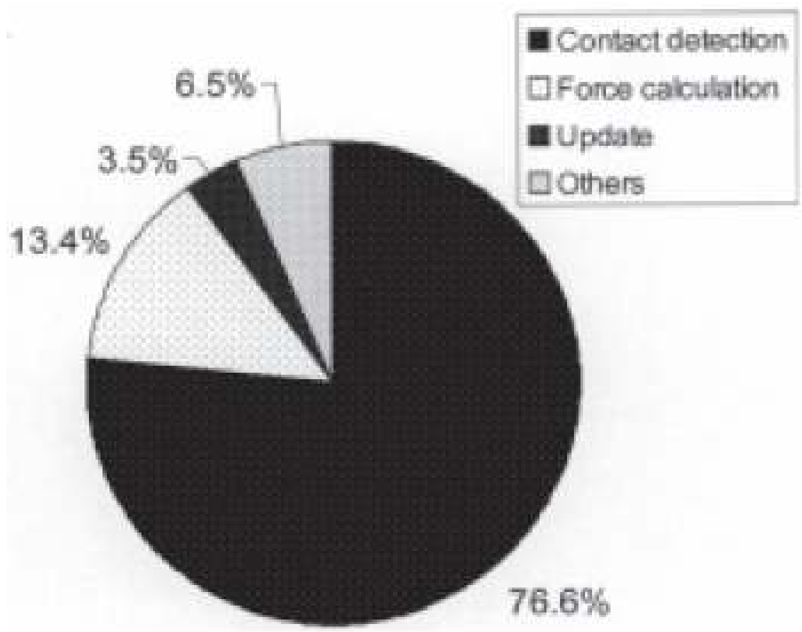
DEM Modeling: Lecture 11

Coarse Contact Detection

Coarse Contact Detection

- Contact detection is typically the most time consuming part of a soft-particle DEM simulation
- Contact between particles is often divided into two steps:
  - coarse contact detection (aka neighbor search)
  - fine contact detection

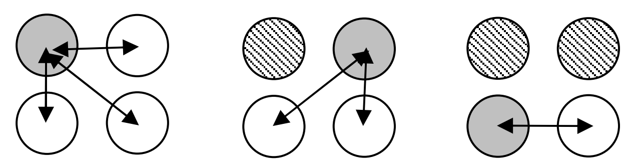


From Mio *et al*. (1995)

Brute Force

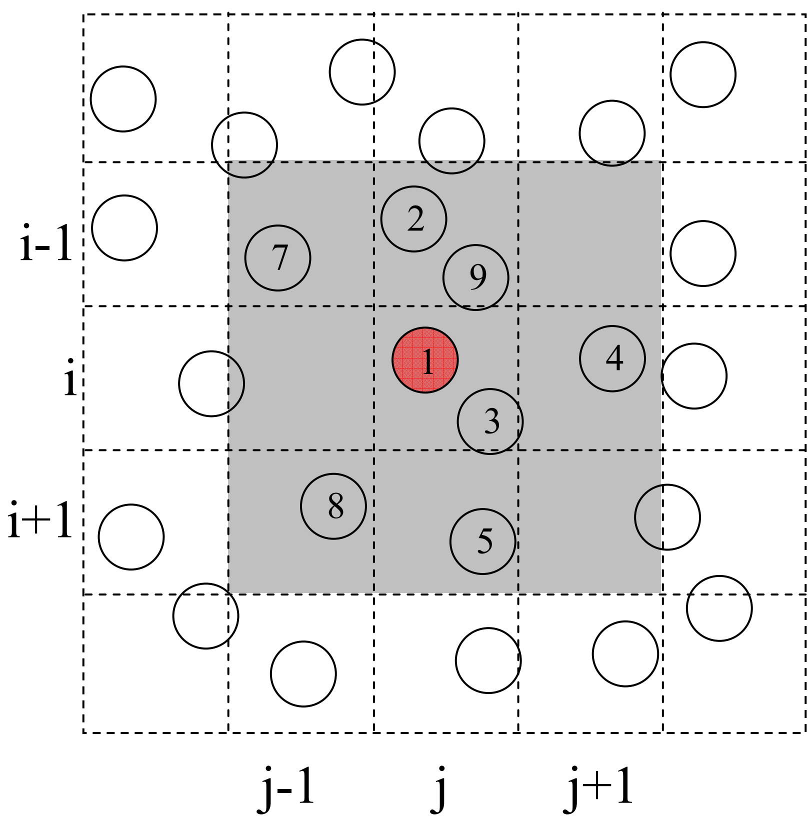
- Assume a system contains N particles
- To determine if contact occurs between any two particles
  - could check for contacts between all possible particle pairs:
    - particle 1: N-1 contact checks
    - particle 2: N-2 contact checks
    - particle N-1: 1 contact check
    - particle N: 0 contact checks
    - total # of contact checks:

$$(N-1)+(N-2)+\cdots+1 = N(N-1)/2 \sim \mathbf{O}(\boldsymbol{N}^{2})$$

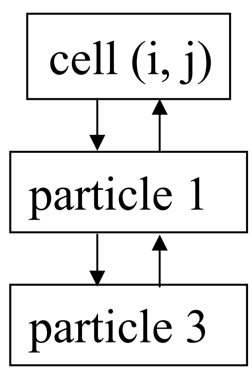


  - aka “naïve” contact detection
- There are more efficient ways of checking for contacts!
  - neighboring-cell contact detection scheme
  - nearest-neighbor contact detection scheme
  - sweep and prune

Neighboring Cell



- divide the workspace into a grid of cells
- for each cell, maintain a list of the particles contained within that cell
- for a given particle, only check for contact with other particles in its own cell and neighboring cells
- cell size may be smaller than particle size, a single particle may occupy multiple cells



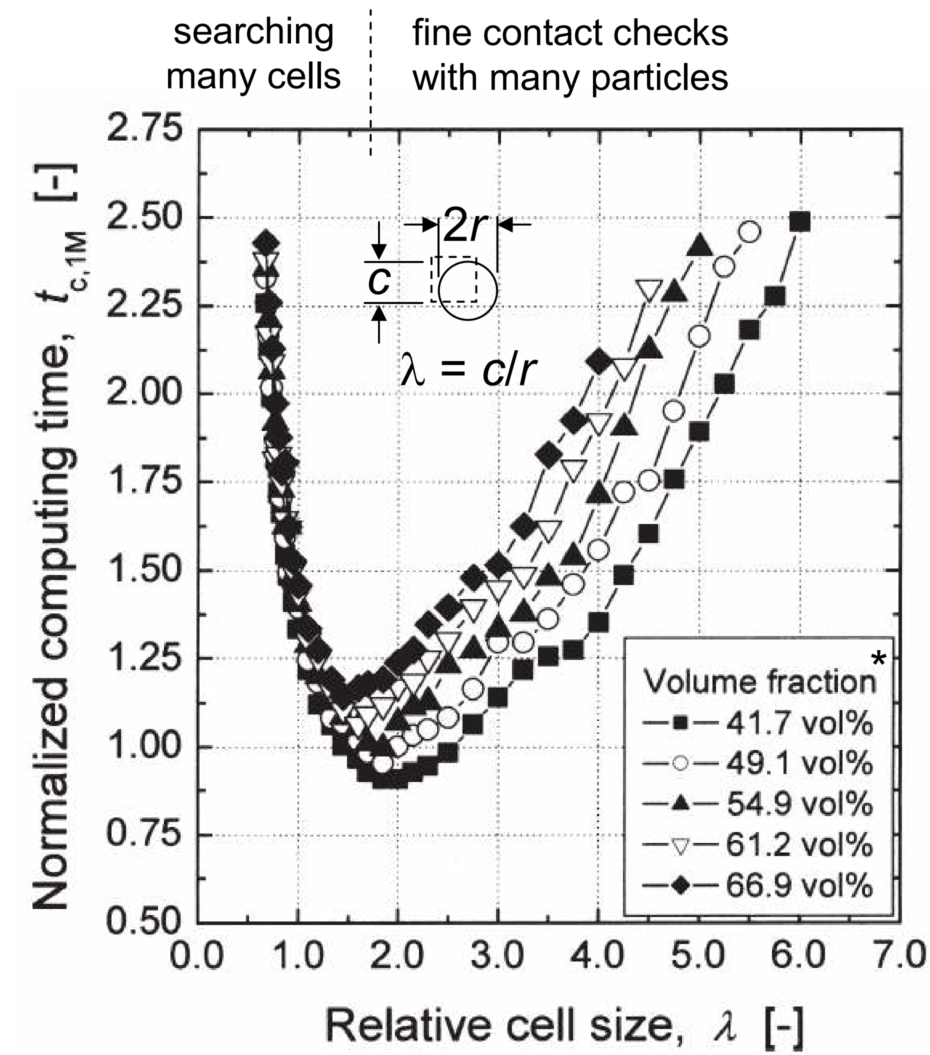
double linked lists are often used to maintain the cell lists

For particle 1, in cell (i, j), check for contact against:

cell (i-1, j-1):	particle 7
cell (i-1, j):	particles 2 and 9
cell (i-1, j+1):	-
cell (i, j-1):	-
cell (i, j):	particle 3
cell (i, j+1):	particle 4
cell (i+1, j-1):	particle 8
cell (i+1, j):	particle 5
cell (i+1, j+1):	-

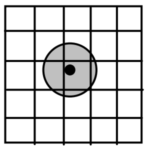
Neighboring Cell…

- Cell size optimization
  - Mio *et al*. (2005)
  - optimal $\lambda \equiv c/r \approx 1.5$ with $\lambda\uparrow$ as solid fraction $\downarrow$
  - optimal cell size has 0.7 – 0.8 particles per cell
  - optimum is insensitive even when a range of particle sizes is used
  - analytical derivation is presented supporting numerical findings

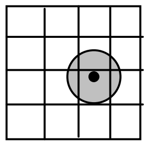


*Here, volume fraction means solid fraction.

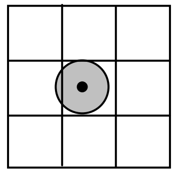
Neighboring Cell…



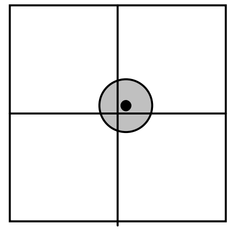
λ = 1.0
n_S = 5



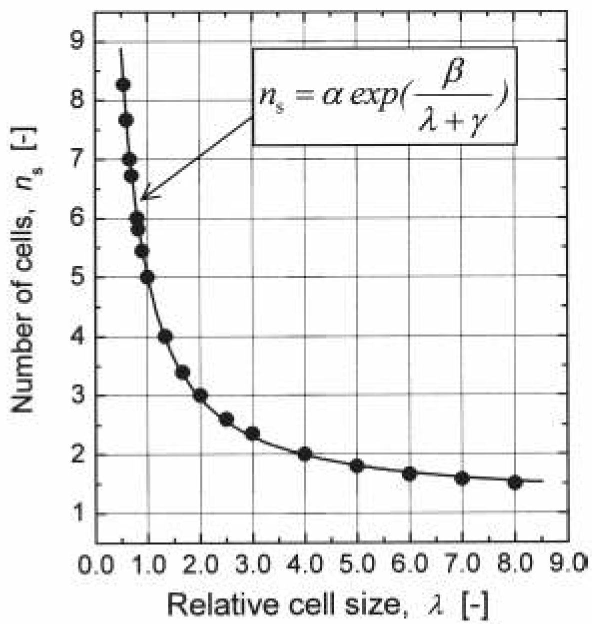
λ = 1.33
n_S = 4



λ = 2.0
n_S = 3



λ = 4.0
n_S = 2



(α, β, γ) = 1.14, 2.65, 0.80)

$$n_{SC} = n_S^3$$ # of searched cells

$$V_C = n_{SC}\left(\lambda r\right)^3$$ volume of searched cells

$$n_C = \frac{N}{V_{WS}} V_C$$ volume of particles in searched cells (N = total # of particles, V_{WS} = volume of workspace)

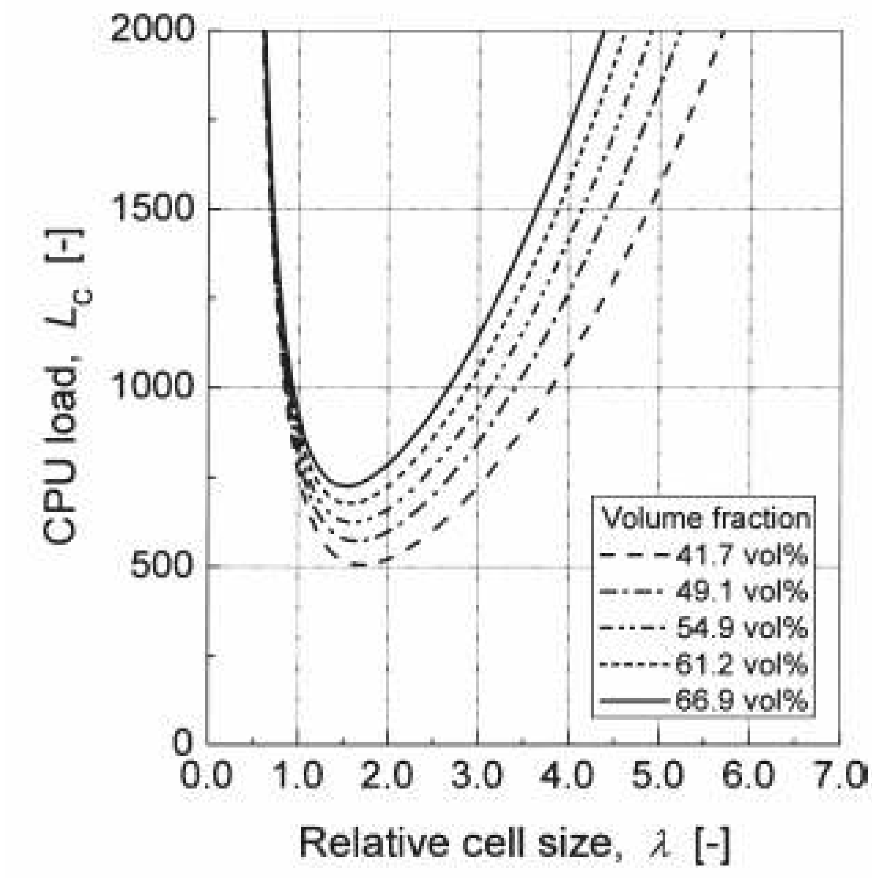
$$n_{CC} = \frac{n_C - 1}{2}$$ avg. # of fine contact checks for a particle

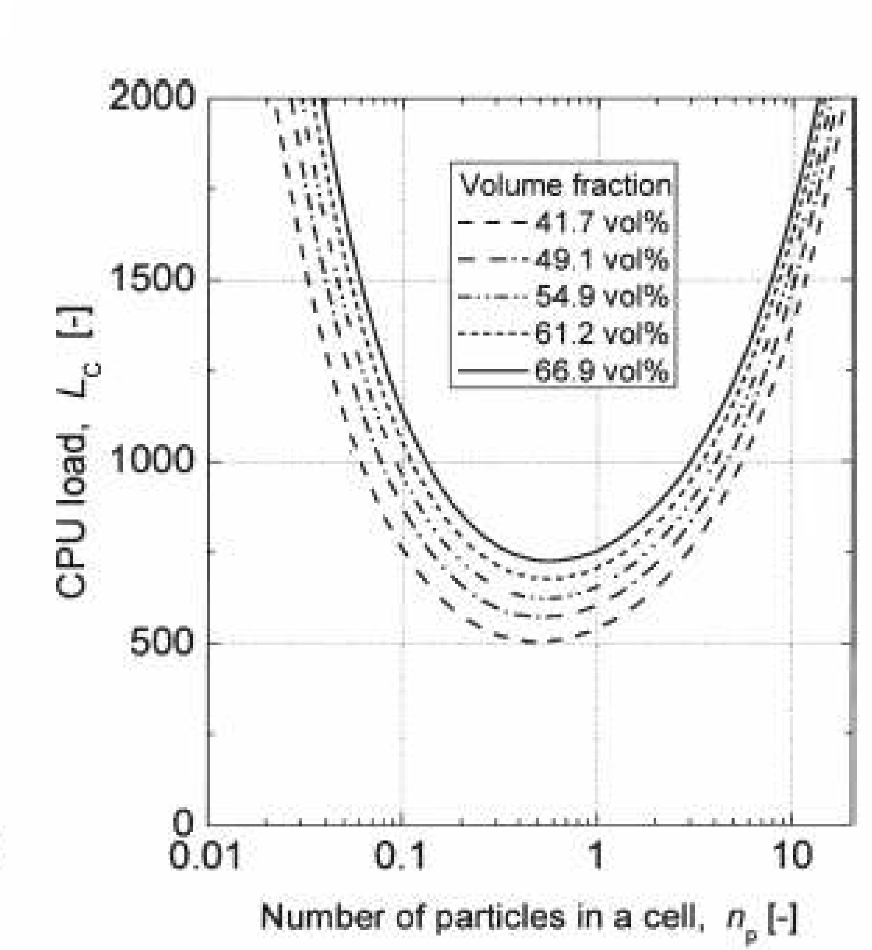
$$L_{CPU} = \kappa_{SC} n_{SC} + \kappa_{CC} n_{CC}$$ CPU load

κ_{SC} = CPU load for searching cells
κ_{CC} = CPU load for fine contact checks
These CPU loads will vary depending upon algorithm and implementation specifics. Mio *et al*. (2005) found that κ_{CC}/κ_{SC} = 10.47

From Mio *et al*. (2005)

Neighboring Cell…



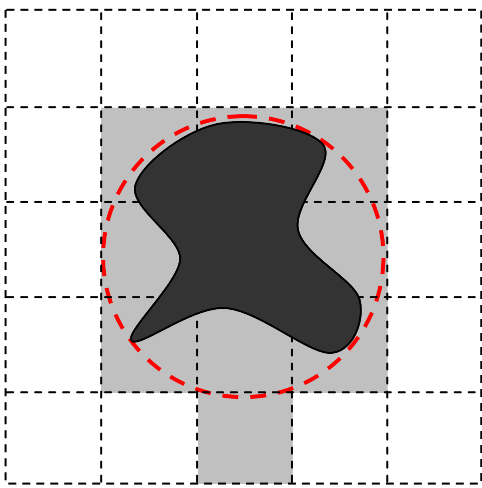


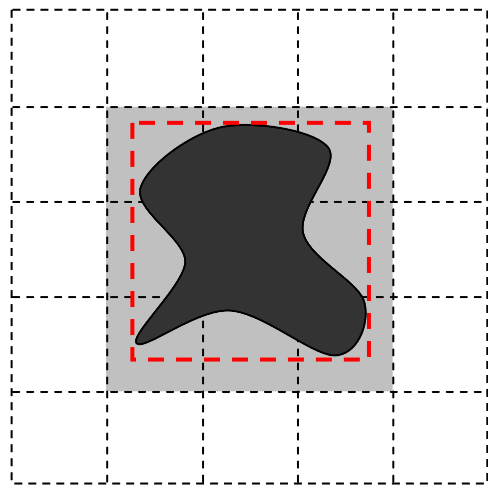
$$L_{CPU} = \kappa_{SC} n_S^3 + \tfrac{1}{2}\kappa_{CC}\left[\frac{N}{V_{WS}}\left(n_S \lambda r\right)^3 - 1\right] \quad \text{where} \quad n_S = \alpha \exp\left(\frac{\beta}{\lambda + \gamma}\right)$$

From Mio *et al*. (2005)

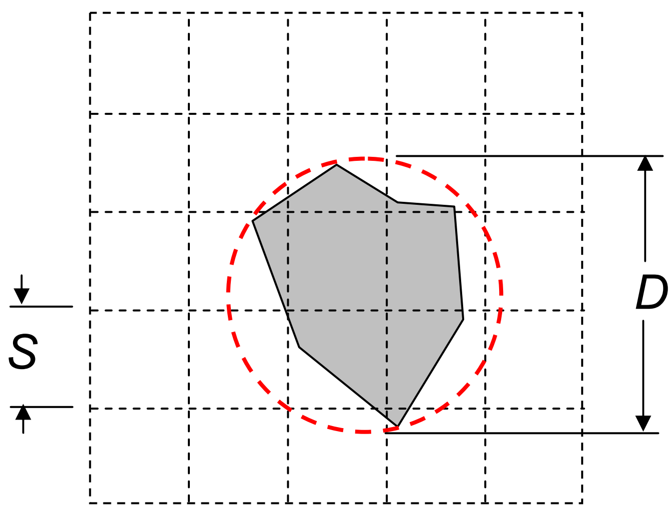
Neighboring Cell…

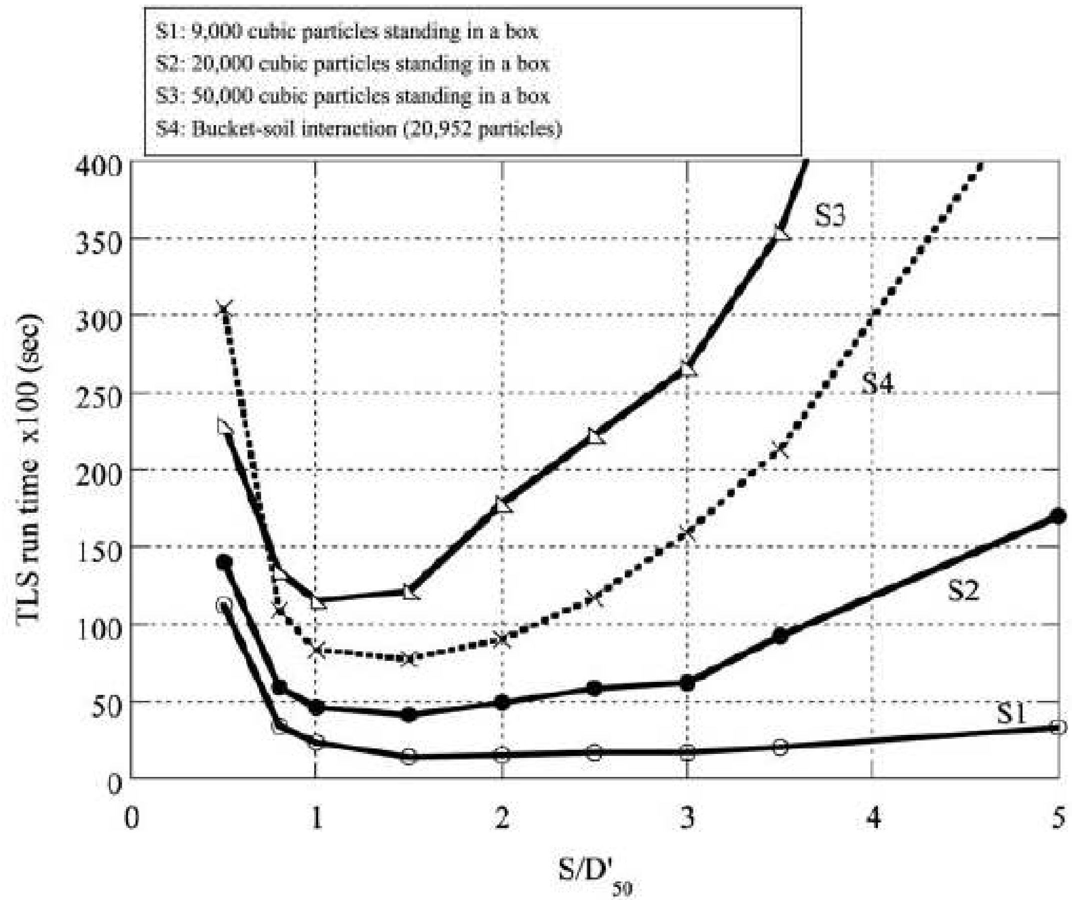
- Can also use bounding spheres or bounding boxes for non-spherical particles and then implement the neighboring cell algorithm





Neighboring Cell…





- Zhao *et al*. (2006) empirically examined optimal cell size using polygonal particles.
- The optimum ratio of cell size to median bounding sphere diameter by volume is: $S/D'_{50} \approx 1.5$.
- Twice the optimal size as what was found by Mio *et al*. (2005)!

Neighboring Cell…

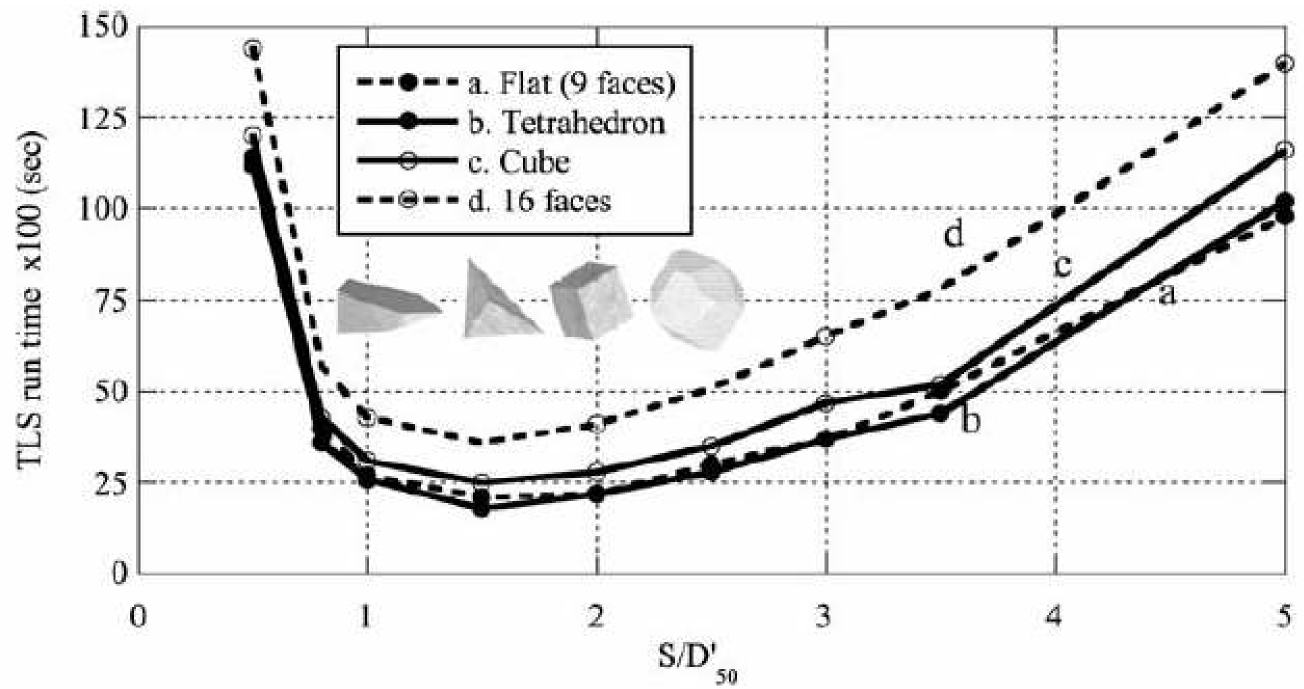


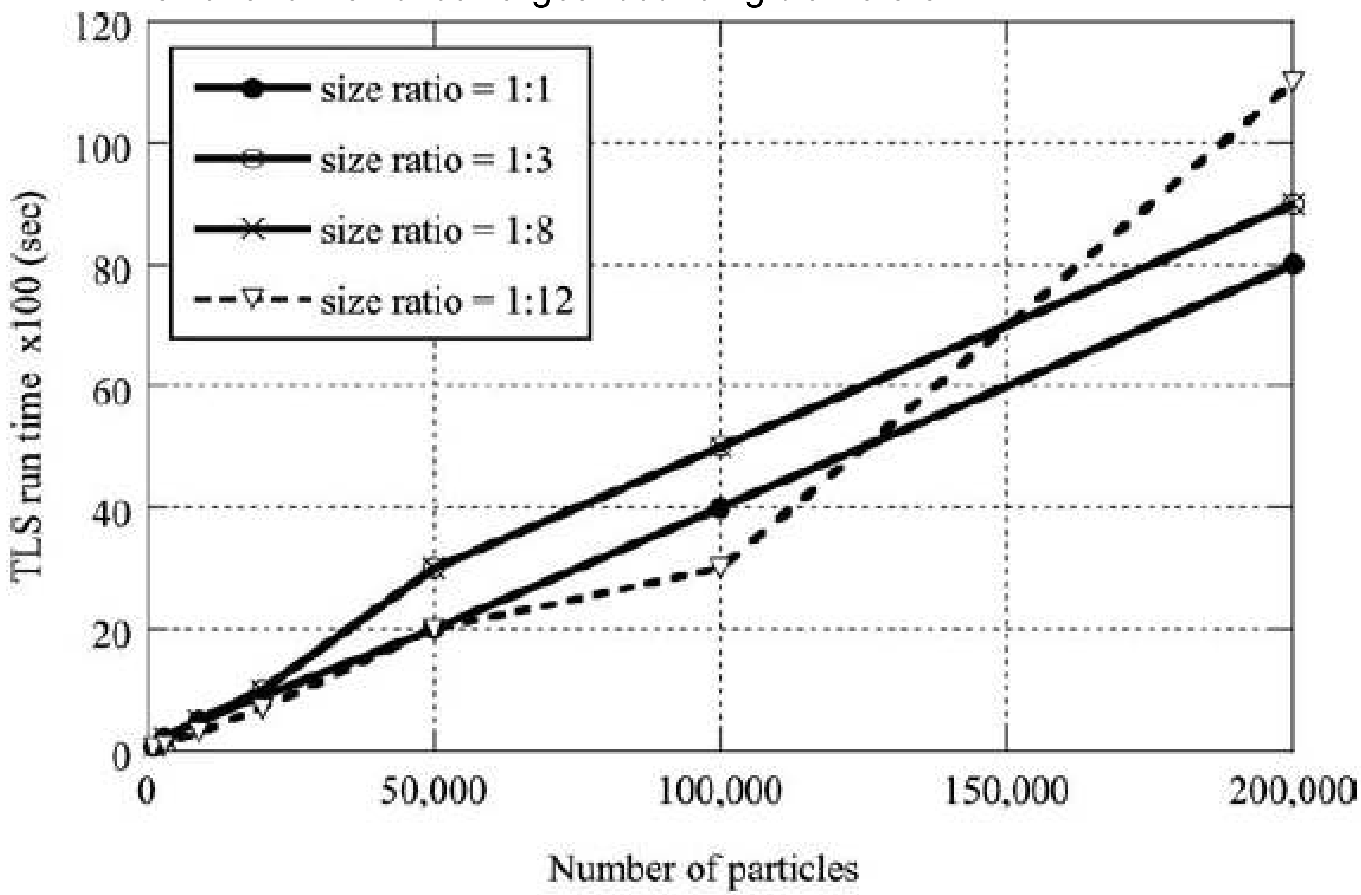
Figure 7. TLS performance: effect of particle shape (50,000 uniform particles standing in a box)

- Shape has little effect on optimal cell size.
- (Increasing shape complexity results in increasing run time.)

From Zhao *et al*. (2006)

Neighboring Cell…

size ratio ≡ smallest:largest bounding diameters



neighboring cell contact detection algorithm is insensitive to particle size ratio

From Zhao *et al*. (2006)

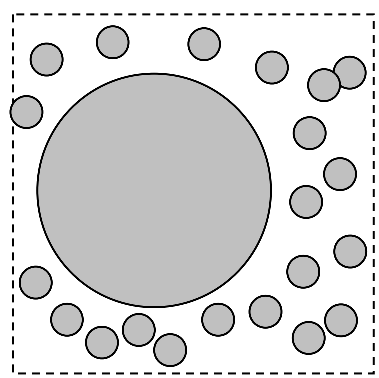
Neighboring Cell…

- Assuming a constant number of particles per cell, c (in 3D with one particle per cell, $c = 26$)
  - particle 1: c checks
  - particle 2: c checks
  - particle N-1: 1 check, at most
  - particle N: 0 checks
  - total # of contact checks:

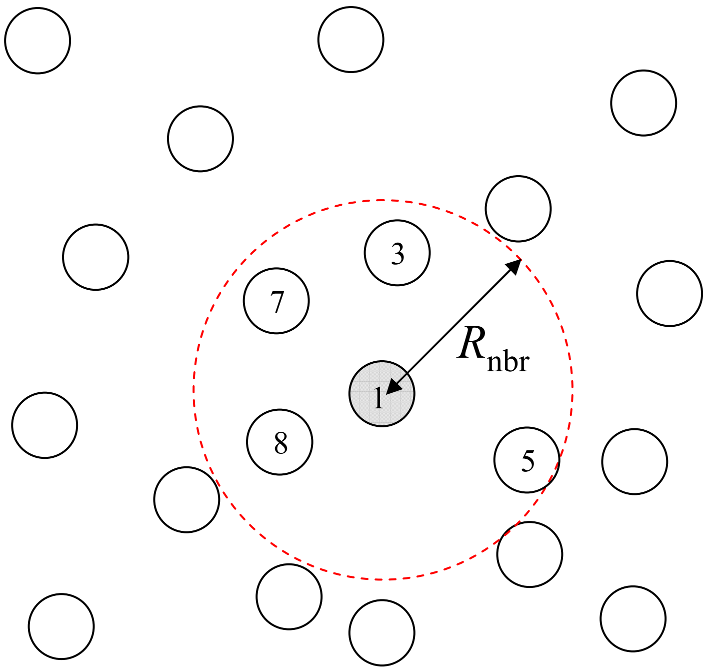
$$(N - c)c + (c - 1) + (c - 2) + \ldots + 1 \sim \mathbf{O}(\boldsymbol{N})$$

Neighboring Cell…

- The additional bookkeeping of maintaining neighbor lists is computationally less costly than performing an N^2 brute force check
- For large particle size differences, the neighboring cell algorithm degenerates to the brute force method if the cell size is chosen to be ≥ particle size
  - for cell sizes < particle size, the algorithm is slow since many cells need to be checked, but it's not as bad as an N^2 check



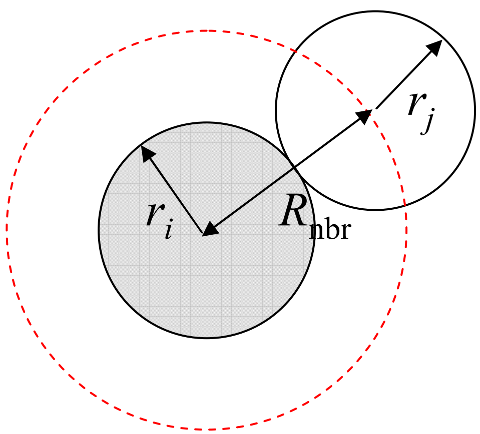
Nearest Neighbor



neighbor list for particle 1:
- particle 3
- particle 5
- particle 7
- particle 8

- define a neighborhood for each particle
- maintain a list of each particle's neighbors
- only check for contacts between neighbors
- periodically update particle neighbor lists so that particles outside the neighborhood will not contact the target particle without first becoming a neighbor
  - periodic updates are typically an N^2 brute force contact search

Nearest Neighbor…



- neighborhood radius needs to be large enough so that a particle moves into the neighborhood before contacting the target particle

$$R_{\text{nbr}} > \left(r_i + r_j\right)$$

- as $R_{\text{nbr}}\uparrow \Rightarrow$ update frequency $\downarrow$, but # neighbors $\uparrow$
- update the neighbor lists when the total distance any one particle could move relative to any other particle is equal to the neighborhood radius

$$\sum_{t_{\text{prev}}}^{t}\left(2\left|\dot{\mathbf{x}}(t)\right|_{\max}\Delta t\right) \geq R_{\text{nbr}}$$

where $R_{\text{nbr}} \equiv$ neighborhood radius

$t_{\text{prev}} \equiv$ time of the previous neighborhood update

$\left|\dot{\mathbf{x}}(t)\right|_{\max} \equiv$ max speed of any particle at the given time

$\Delta t \equiv$ simulation time step

Nearest Neighbor…

- Assuming a constant number of particles per neighborhood, c
  - particle 1: c checks
  - particle 2: c checks
  - particle N-1: 1 check, at most
  - particle N: 0 checks
  - total # of contact checks:

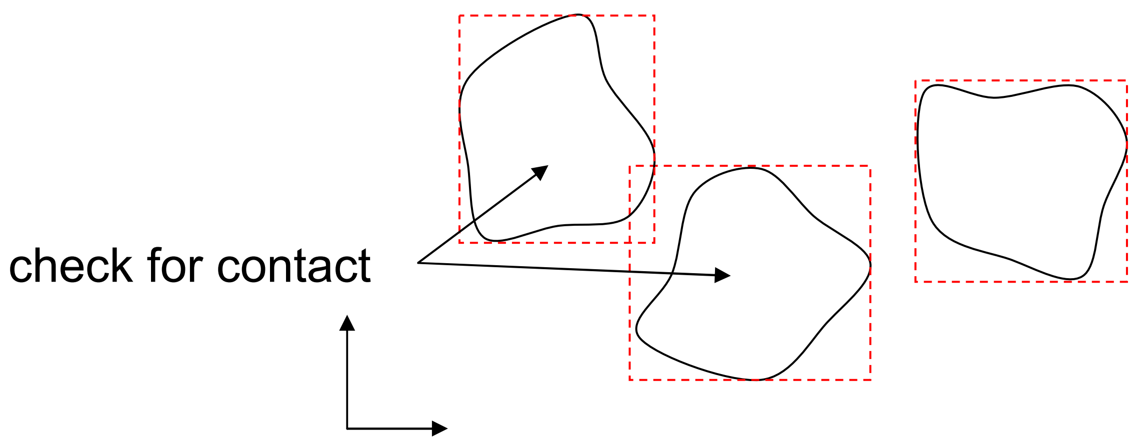
    $(N - c)c + (c - 1) + (c - 2) + \ldots + 1 \sim \mathbf{O}(\boldsymbol{N})$

Nearest Neighbor…

- The additional bookkeeping of maintaining neighbor lists is computationally less costly than performing an N^2 brute force check
- Nearest-neighbor becomes less efficient as the frequency of updating the neighbor lists increases
  - *e.g.,* when particles move at large speeds
  - $\Rightarrow$ nearest-neighbor technique is most efficient for quasi-static assemblies
- Optimal cell size and neighborhood size have not been studied
  - (left as an exercise)

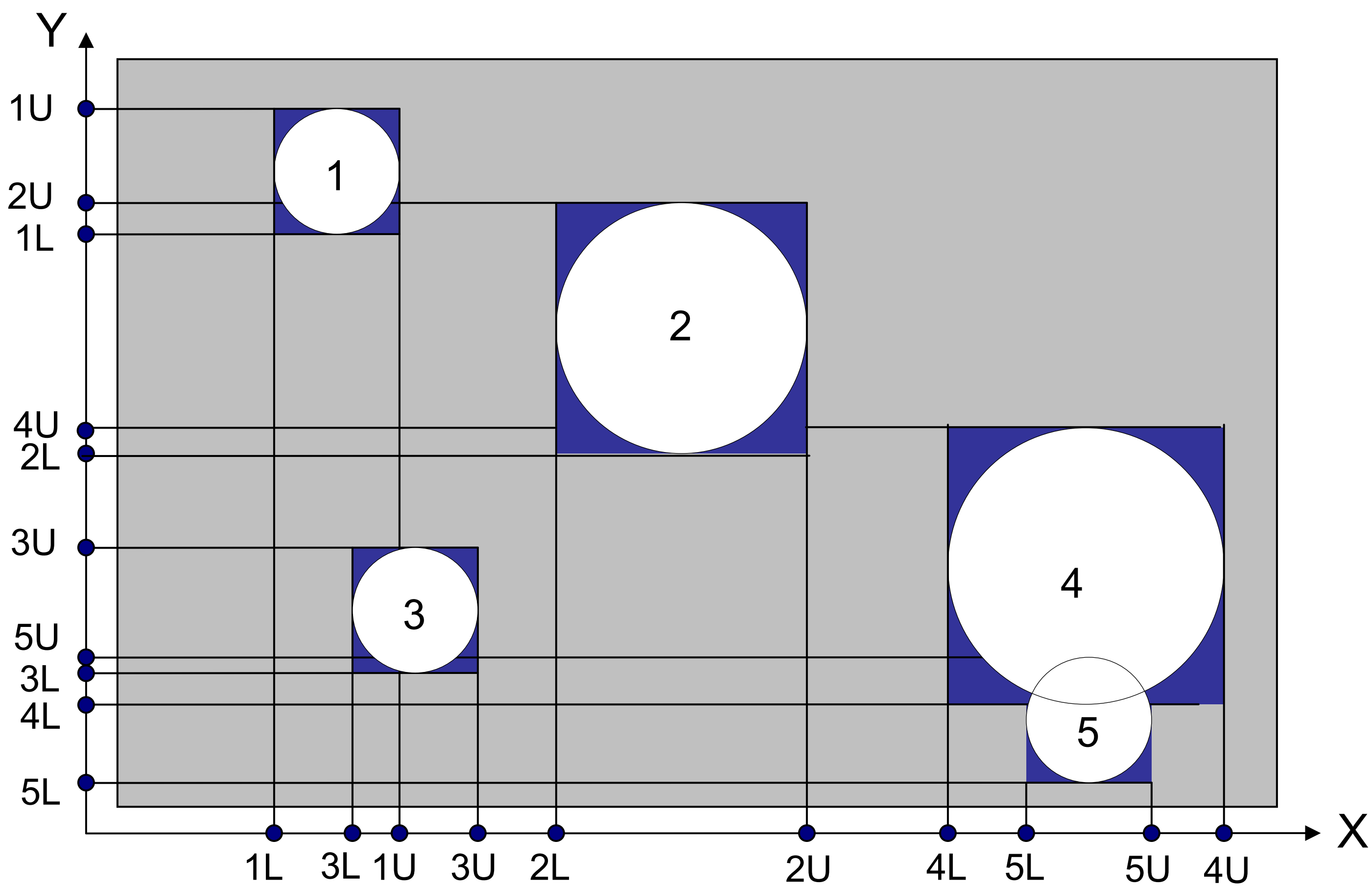
Sweep and Prune

- aka bounding box method, spatial sort
- each particle has a bounding box with edges aligned with the global axes
- if the bounding boxes don't overlap in all three coordinate directions, then the particles will not overlap
  - only check for contact between particles with overlapping bounding boxes



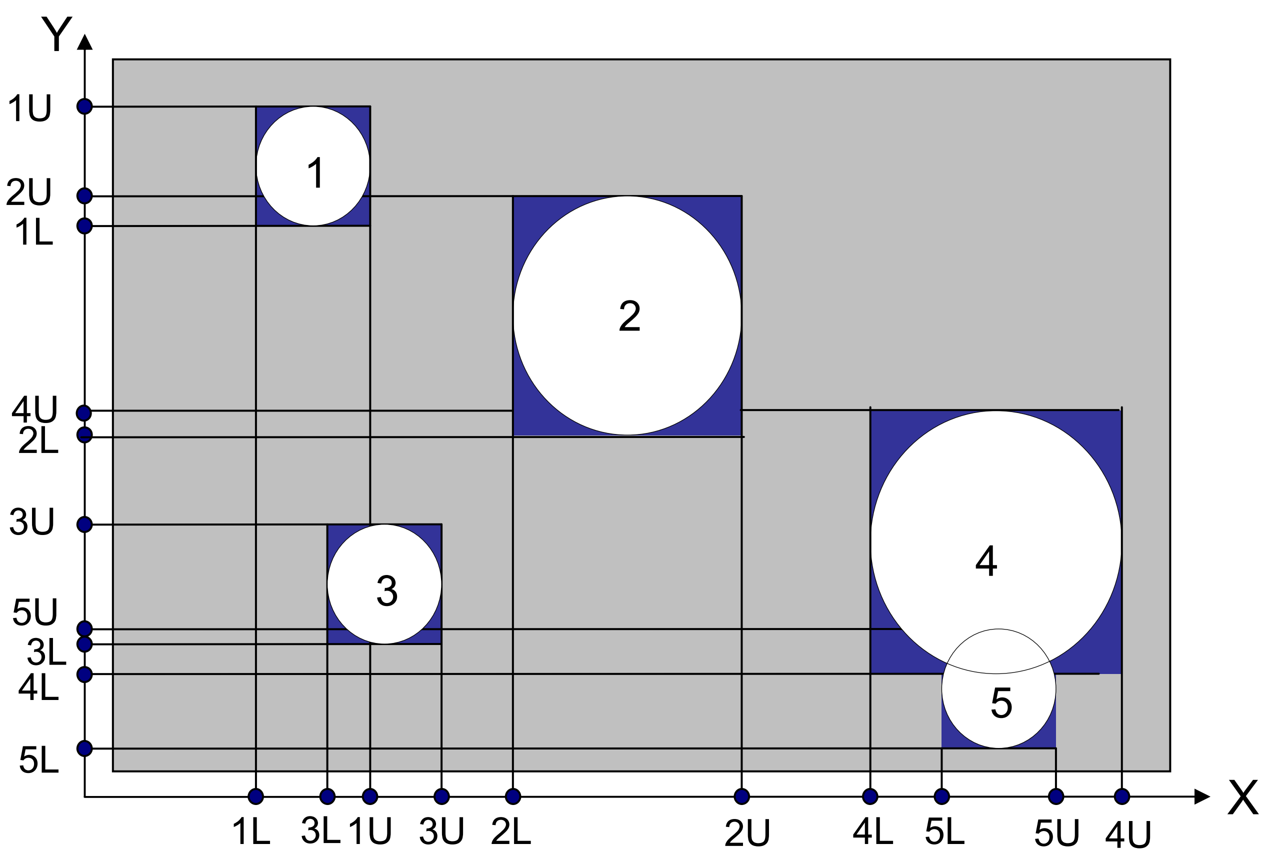
Sweep and Prune…

Step 1: Create (axis aligned) bounding boxes for each particle. Note the projected coordinates of the box extrema on each axis.



Sweep and Prune…

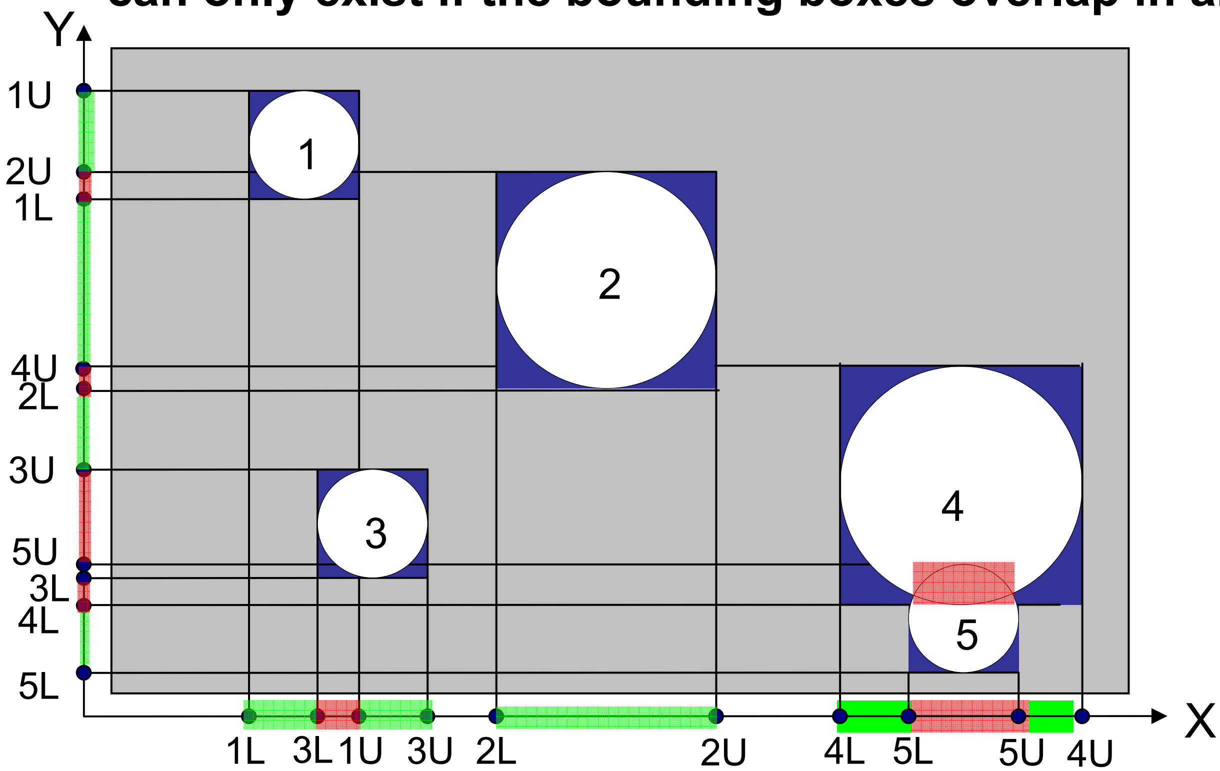
Step 2: Create sorted lists of the bounding box segments (endpoint pairs) in each projected dimension. Retaining the ordering from the last frame makes this a fast process since the ordering generally won't change much between simulation time steps (known as "coherence").



***x* list**	***y* list**
1 L	5 L
3 L	4 L
1 U	3 L
3 U	5 U
2 L	3 U
2 U	2 L
4 L	4 U
5 L	1 L
5 U	2 U
4 U	1 U

Sweep and Prune…

Step 3: Sweep through each list, tracking which boxes overlap. Contacts can only exist if the bounding boxes overlap in all axis directions.

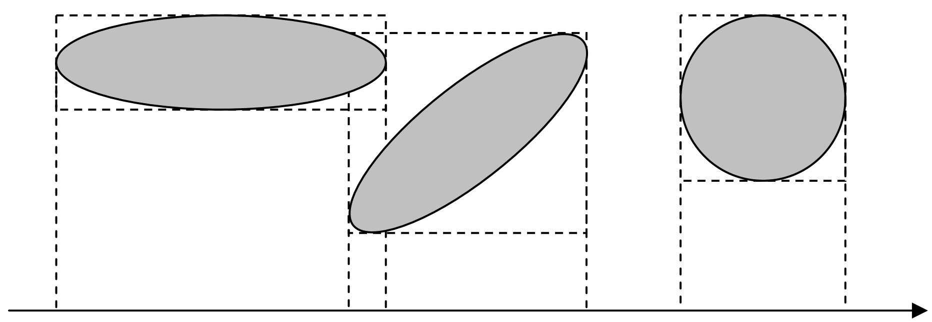


***x* list**	***y* list**
1 L	5 L
3 L	4 L
1 U	3 L
3 U	5 U
2 L	3 U
2 U	2 L
4 L	4 U
5 L	1 L
5 U	2 U
4 U	1 U
BB overlap	*BB overlap*
1 & 3	*1 & 2*
4 & 5	*2 & 4*
	3 & 4
	3 & 5
	4 & 5

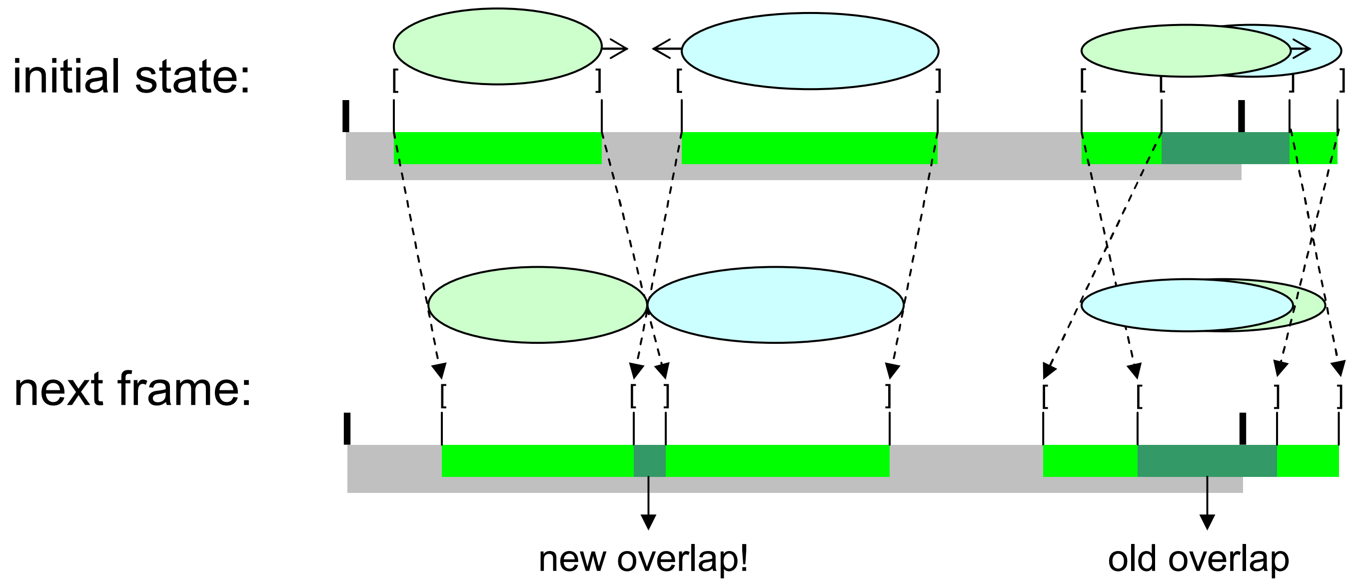
check for contact: 4 & 5

Sweep and Prune…

- O(N) achieved by assuming spatial coherence: since the segments move very little, the lists are always “almost-sorted,” and only linear time is required to update them.
- Optimization: use insertion sort for the lists, “sweep” segments as they are sorted (combine steps 2 and 3).
- Any object shape can be used, as long as a bounding box encloses it.



Sweep and Prune…



- All segment list sorting involves swaps of adjacent endpoints (step 2). Use this information to accomplish step 3.
- Test for swap type to update collision table:
  – Swap] [to [ ] : new overlap for pair
  – Swap [ ] to] [: end overlap for pair
  – Otherwise do nothing ([[or]])

Sweep and Prune…

- Performs well when compared with neighboring cell, but is more complex to implement
- Advantages
  - handles poly-sized particle distributions without degenerating
  - deals with high density systems more efficiently
- Disadvantages
  - no advantage for simple systems (low density, mono-sized spheres)
  - more difficult to implement, especially for moving periodic boundaries
  - oblong particles difficult to bound efficiently
- DESS (Perkins and Williams, 2001)
  - aka sweep and prune
  - $O(N^2)$ for an insertion sort, $O(N \ln N)$ for heap sort

Summary

- Most computation time is spent in coarse contact detection
- Do NOT use brute force except for systems containing only tens of particles at most
  - (unless you like to waste time and electricity)
- Three common methods: neighboring cell, nearest neighbor, sweep and prune
- The “best” method depends upon the system under investigation
  - quasi-static systems: nearest neighbor
  - systems with large size differences: sweep and prune
  - the “work horse”: neighboring cell

References

- Allen, M.P. and Tildesley, 1989, *Computer Simulation of Liquids*, Oxford University Press, New York, pp. 149 – 152.
- Cohen, Lin, Manocha, Ponamgi, 1995, "I-COLLIDE: An interactive and exact collision detection system for large-scale environments," in *Proceedings of the 1995 Symposium on interactive 3D Graphics,* SI3D '95.
- Gavrilova, Rokne, Gavrilov, Vinogradov, 2002, "Optimization techniques in an event-driven simulation of a shaker ball mill," *Lecture Notes in Computer Science*, Vol. 2331.
- Mio, H., Shimosaka, A., Shirakawa, Y., and Hidaka, J., 2005, "Optimum cell size for contact detection in the algorithm of the discrete element method," *Journal of Chemical Engineering of Japan*, Vol. 38, No. 12, pp. 969 – 975.
- Munjiza, A. and Andrews, K.R.F., 1998, "NBS contact detection algorithm for bodies of similar size," *International Journal for Numerical Methods in Engineering*, Vol. 43, pp. 131 – 149.
- Perkins, E. and Williams, J.R., 2001, "A fast contact detection algorithm insensitive to object sizes," *Engineering Computations*, Vol. 18, No. 1/2, pp. 48 – 61.
- Vemuri, B.C., Chen, L., Vu-Quoc, L., Zhang, X., and Walton, O., 1998, "Efficient and accurate collision detection for granular flow simulation," *Graphical Models and Image Processing*, Vol. 60, pp. 403 – 422.
- Williams, J.R. and O'Connor R., 1995, "A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries," *Engineering Computations*, Vol. 12, pp. 185 – 201.
- Zhao, D., Nezami, E.G., Hashash, Y.M.A., and Ghaboussi, J., 2006, "Three-dimensional discrete element simulation for granular materials," *Engineering Computations*, Vol. 23, No. 7, pp. 749 – 770.