Cell Size issues

The number of cells an object overlaps is guaranteed to be no more than four cells (for a 2D gird; grid for a 3D grid)

Use all-pairs approach

In ray tracing, a popular method for determining the grid dimensions has been the n^(1/3) rule: given n objects, divide space into a k x k x k grid, with k = n^(1/3). The ratio of cells to objects is referred to as the grid density.

**1. Grids as arrays of linked lists, objects in the cell gets inserted into the linked list**

The natural way of storing objects in a grid is to allocate an array of linked list.

single-liked list, updating and deletion of an arbitrary object is O(n). But updating and deletion can also be made O(1) by using a double-linked list and having the objects maintain direct pointers to the linked-list entries of all cells the objects are in

**Embedded linked list**

P287 "The easiest way of maintaining these linked lists is to embed the links within the objects themselves

**2. Hierarchical Grids**

**A grid structure particularly well suited to holding dynamically moving objects**

**Map objects and cells into a hash table**

to handle collisions, the mapping of two keys to the same bucket

**Open hashing**

Linked list hash table or separate chaining

Each bucket is allowed to contain a linked list of records

Objects is inserted into the cell in which its center (based on the object bounding sphere) or top left-hand position(based on the object AABB) is in.

Level 1 encompass the smallest objects

Level 2 double the size

...

...

Testing whether an arbitrary object A is in collision with objects stored in the hgrid is done by traversing through all Hgrid levels.

To speed up the collision testing further, it is worthwhile keeping track of the total number of objects on each grid level. Levels that contain no objects can then be excluded from testing.

Hgrid can be very memory expensive when each grid level is allocated as a dense array. For this reason they are best implemented using hashed storage, as described in section 7.1.3

**Compare map vs hash\_map**

map is implemented using some kind of balance tree, so lookup time is O(log n)

**How does hash Table have O(1) look up time?**

http://stackoverflow.com/questions/4363539/how-does-hashing-have-an-o1-search-time

a hash table is an array containing all of the keys to search on.

**Imagine:**

hash(x) = 3 (x is the key), and so

array[3] = your value

Implicit Grids

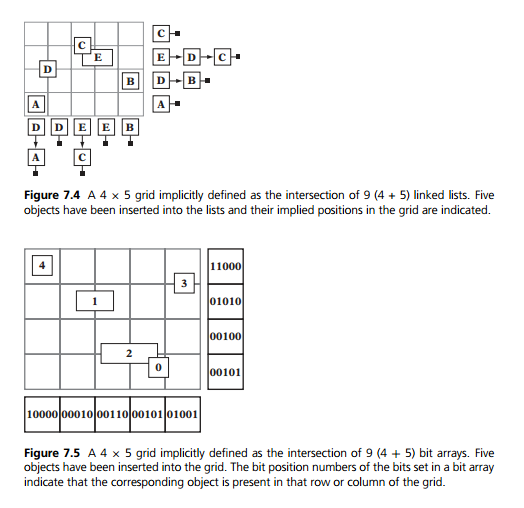
3 x one array elements, each array element points to a linked list of objects.

an object is inserted into the grid by adding it to the lists of the grid cells it overlaps.

overlap testing for an object is now performed by checking to see if the object overlaps another object in both the row cells and the column cells

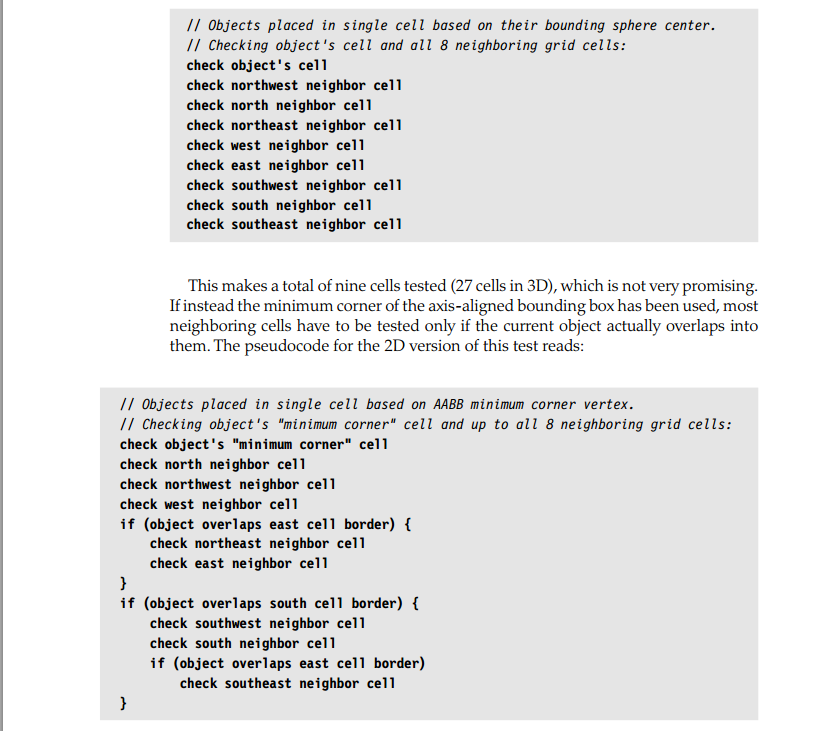
if object is fully contained within a single grid cell of the grid, it is inserted into two lists (one row and one column list),

but if a object occupies 4 x 4 grid cells, it would be inserted into eight lists for the implicit grid (four row lists and four column lists), instead of 16.



For any object, all neighboring cells and their contents must be tested for collision

in 2D



AABB minimum corner vertex

Number of Objects

if do pairwise test for every objects, we get O(n^2) run time

broad phase, n-body processing

narrow phase, pair processing

The best possible visuals games must run at 60 fps

depending on the type of game, collision detection may account for, 10 to 30% of a frame,

which leaves you 2 to 5ms for collision detection

for an action packed game, you may only get 50 to 250micro-seconds available to handle the collision for each object

can it handle objects of largely varying sizes?

implicit grid using bit arrays

dense array (with double-linked lists)

sparse array (with double-linked lists)

**Objects are associated with a single cell**

If using axis-aligned bounding box, most neighboring cells have to be tested only if the current object actually overlaps into them.

**Particle system collision**

https://www.youtube.com/watch?v=a8uZUqIEsoI

**Additional Grid Considerations**

the implication is that grids are kept from frame to frame and are incrementally updated

**Hierarchical Girds**

objects are inserted into a single on just one of the levels. To minimize the number of neighboring cells that have to be tested for collisions, objects are inserted into the h-grid at the level where the cells are large enough to contain the bounding volume of the object.

This way objects are guaranteed to overlap at most four cells.

**QuadTree**

Collision detection of particle systems using quadtree

https://www.youtube.com/watch?v=fuexOsLOfl0

http://dylanscoderef.wordpress.com/algorithms-and-concepts/quadtrees/

**Octree**

pointer based and array based representation

the array representation is more suitable for static scenes and static octrees. The pointer-based representation is more useful for dynamic scenes

**k-d tree**

**Apparently the most versatile one is BSP trees**

**Resources**

**Papers about Collision Detection**

http://gamma.cs.unc.edu/research/collision/

**Different spatial partitioning solutions**

http://www.altdevblogaday.com/2011/02/21/spatial-partitioning-part-1-survey-of-spatial-partitioning-solutions/

Grid: 3d grids with 2d distribution

**Collision detection based on spatial partitioning**

http://cg.informatik.uni-freiburg.de/course\_notes/sim\_08\_sp.pdf