Heap Based Priority Queues

4-6-2011

Opening Discussion

- Minute essay comments
  - Complexity of kD-tree code.
  - How do I code so fast?
- Solutions to the interclass problem.
- Testing our kD-tree.

Heaps

- We previously implemented a priority queue using a sorted linked list. This data structure has the disadvantage that adding takes O(n) time. For data structures we try to avoid O(n) if at all possible for the common operations because it typically leads to $O(n^2)$ overall performance.
- There is another data structure we can use to implement a priority queue called a heap. Using a heap, none of the operations will be worse than O(log n).
- There are actually many types of heaps. We will be doing the simplest type, the binary heap.

Binary Heaps as Trees

- You can view a binary heap as a type of binary tree, though our final implementation will not actually use a binary tree.
- Binary heaps are complete trees that have heap ordering.
  - A complete tree is one that only starts filling the next row after the one above it is completely full. We will fill rows from left to right adding in new nodes until we have to move down to the next row.
  - Heap ordering is simply the property that each node has a higher priority than its children.

Adding

- When we add to a binary heap we simply place the new element at the next free spot on the tree. We then have to shift things to restore heap ordering. We refer to the new node as a bubble which moves up through the tree until it comes to rest.

Removing

- The only item we can remove from the tree is the root, which has the highest priority thanks to the heap ordering. To keep the heap complete we move the last element up to the root and let it sink down through the tree until it comes to rest.
- Both of these operations are always O(log n) because the heap is complete.

Heaps as Arrays

- The tree structure is nice for visualizing the heap, but it is extremely inefficient for implementation.
- If we order the nodes starting at 1 you will see nice mathematical relationships between parents and children. We use this numbering as an index into a array and store the heap in an array or an array based list.

Code

- Let's write a binary heap.

Minute Essay

- Why are the add and remove operations on a binary heap always O(log n) when a normal binary tree can't promise that?