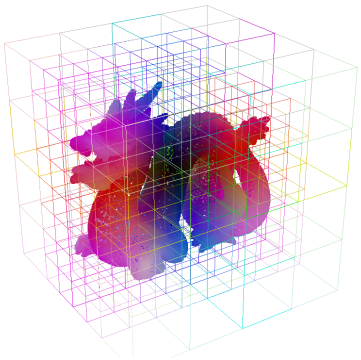
3D Geometry Processing
Exercise 2

Fall 2013

Hand in: 10.10.2013, 16:00



Hash Octree

This week you will complete the implementation of a hashtable based octree. This kind of octree implementation allows extremely fast and flexible navigation through its adjacency structure by mapping the adjacency operations to bitwise operations on the hash keys ((&), or (|), not ($\sim$), bitshifts (<<, >>), dilated addition/subtraction). We will use this octree implementation in the next week's exercise on surface reconstruction. The provided hashbased octree is already implemented to a large extent, but its core, the methods for Morton code manipulations and the methods to navigate through the hash octree, is missing.

1 Morton Code Manipulations

1. Download the basecode_exercise2.zip package provided on the Ilias exercise page, and copy the class `MortonCodes.java` and the class `Assignment2.java` into a new assignment2 package in your Eclipse Project.

2. The class *MortonCode* takes care of the Morton code manipulations. Implement the following methods as discussed in the exercise session:

   (a) `long parentCode(long hash)`: this method should return the Morton code logically associated to the parent of the cell.

(b) `long nbrCode(long hash, int l, int Obxyz)`: this method should use *dilated addition* to find a neighbor code on the level `l`. The parameter Obxyz encodes the difference vector, e.g. 0b101 would denote that the caller is seeking the neighbor which lies at the relative grid position +1x, +0y, +1z. The method should check if the dilated addition produces an overflow, and return -1 if it does.

(c) `long nbrCodeMinus(long hash, int l, int Obxyz)`: this method should use *dilated subtraction* to find a neighbor code on the level `l`. The parameter Obxyz encodes the difference vector, e.g. 0b101 would denote that the caller is seeking the neighbor which lies at the relative grid position -1x, -0y, -1z. The method should check if the dilated subtraction produces an underflow, and return -1 if it does.

(d) Implement the two methods `isCellOnLevelXGrid(long code, int l)` and `isVertexOnLevelXGrid(long code, int l , int depth)`. The first one should check if the (cell) Morton code is logically associated to a level l cell, and the second should test if the vertex code is associated to a vertex that occurs on the level l grid. The vertex code should be assumed to be padded with zeros to have the length $3depth + 1$.
(*Hint:* $\sim (-1L << k)$ produces a mask where the last k bits are 1...)

3. Thoroughly test your methods on example Morton codes. Some examples are given in the `Assignment2.java` class, but you should make up more codes in order to test z-underflow (asking for the negative z-neighbor when there is none), arbitrary overflow cases and everything else you can think of. **4** Points

2 Hashtree Navigation

1. Copy the remaining files from basecode_exercise2.zip into the corresponding packages in your Eclipse Project, familiarize yourself with the code. The hashtree is implemented in the classes `HashOctree.java`, `HashOctreeCell.java` and `HashOctreeVertex.java`. The unimplemented methods are all located in the `HashOctree.java` class.

2. Navigation to parent cells:

   (a) Implement the method `HashOctreeCell getParent(HashOctreeCell cell)`, which should return the parent cell of the argument.

   (b) Visualize the parent-child adjacencies similarly as in Figure 1 a).

   2 Points

3. Navigation from cells to neighbor cells:

   (a) Implement the methods `HashOctreeCell getNbr_c2c(HashOctreeCell c, int Obxyz)` and `HashOctreeCell getNbr_c2cMinus(HashOctreeCell c, int Obxyz)`. The methods should return the neighbor cell at the relative grid position denoted by Obxyz on the parameter's cell level c.lvl. If no such cell exisist it should return the closest existing lower lever cell that would contain the queried neighbor; such a cell is found by iterating through the parent codes of the neighbor code, and returning the first existing cell.

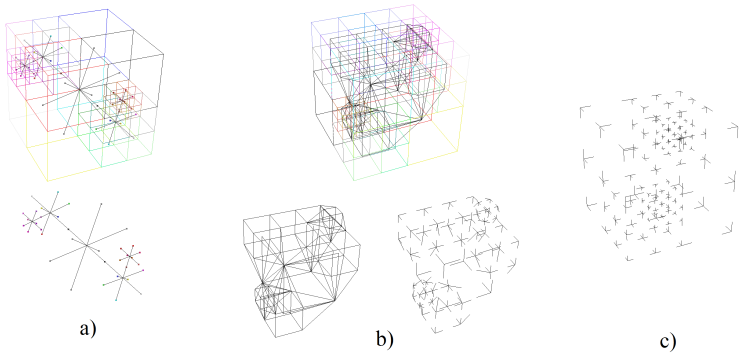


Figure 1: Visualization of the child-parent relation by connecting them(a), the cell to cell neighbors by drawing lines from the cell center pointing in the direction of the adjacent cells (b) and the vertex to vertex neighbors (c).

(b) Visualize the cell to cell adjacencies in order to check if they are correctly computed, e.g. as in Figure 1 b).

2 Points

4. Navigation from vertices to neighbor vertices:

   (a) Implement the methods `HashOctreeVertex getNbr_v2v(HashOctreeVertex v, int Obxyz)` and `HashOctreeVertex getNbr_v2vMinus(HashOctreeVertex v, int Obxyz)`. The methods should start seeking the neighbor vertex Obxyz starting on the highest grid level v occurs (v.maxLevel) and then iterate through the levels until either an existing vertex is found or the lowest grid on which v occurs is reached (v.minLevel). If no vertex is found, null should be returned.

   (b) Visualize the vertex to vertex adjacencies in order to check if they are correctly computed, e.g. as in Figure 1 b).

   2 Points

3 Hand-In

Don't forget to:

1. Commit your code via the Ilias exercise page before the deadline.
2. Prepare your code demonstration and reserve a time slot for your demo via the Google doc shared in the Forum.