Block Size	Blocks Allocated	Blocks Used		
		2^{10}	2^{15}	2^{20}
2	$0.55n$	59%	67%	70%
4	$1.3n$	90%	90%	88%
6	$1.55n$	90%	90%	87%
8	$1.3n$	78%	73%	75%
10	$1.8n$	30%	51%	63%

Fig. 5. The number of blocks of each size that are allocated for an n-vertex 3D mesh, and the percentage of blocks that were used for $n = 2^{10}$, 2^{15}, and 2^{20}.

list of the corresponding neighbors. The code for each neighbor v' is followed by a code for the number of nibbles in the encoding of the representative edge (v, v'), and a pointer to the first block containing the data for that edge. (The pointer is stored using the same hash trick as above to keep pointer sizes small.) Every representative edge has its own block allocated from the memory pool, with the capability to allocate additional blocks if needed.

When an edge is queried, our implementation loads only the list for one vertex and for the edge itself into the cache. It does not need to decompress the other edges adjoining that vertex.

Since the number of nibbles needed per representative edge is quite variable, our data structure allocates from pools of 2, 4, 6, 8, or 10-byte blocks to reduce wasted space. The number of blocks in each pool was determined experimentally and is shown in Figure 5. The data structure ensures that each pool always has at least 10% free space; if a block cannot be allocated from a given pool, the data structure looks for a larger one. The initial block for each vertex comes from a separate array containing blocks of size 7.

6.3. *Dynamic point generation*

To support dynamic point generation we use an expanded label space. If a total of n vertices are to be generated, we allow for $2n$ possible labels. Each label receives a one-byte hash pointer which, if the label is in use, points to the initial data block for the corresponding vertex. The initial vertices are spread evenly across the label space.

6.4. *Incremental Delaunay*

We implemented a Delaunay triangulation algorithm in two and three dimensions using our compressed data structure. We employ the well-known Bowyer-Watson kernel[21,22] to incrementally generate the mesh. During the course of the algorithm a Delaunay triangulation of the current pointset is maintained. An incremental step inserts a new vertex into the mesh by determining the elements that violate