Fast Compressed Segmentation Volumes for Scientific Visualization Supplementary

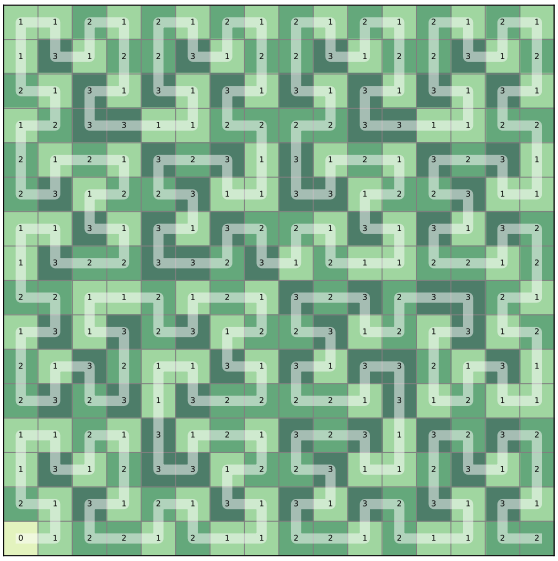

Fig. 1: The number of valid neighbor references per node within the same LOD when using a Hilbert curve in 2D. The average number on an infinitely sized brick is 2. For this 16^2 brick it is 1.875 because of border nodes.

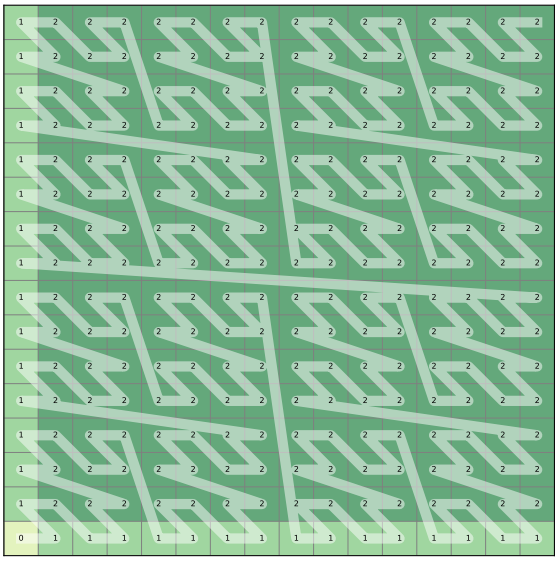

Fig. 2: The number of valid neighbor references within the same LOD when using a Morton curve in 2D. Again, the average number is 2 for an infinitely sized brick and 1.875 for this 16^2 brick, but the number of neighbors for each (non-boundary) position is constant. The 3D case is analog.

1 Comparison of Hilbert and Morton Z-curves

Our compression encodes multi-grid nodes in a distinct order along a space filling curve. At first, it may seem that the choice of the space filling curve may significantly influence compression ratios, e.g. by using a Hilbert curve with better space filling properties instead of a Morton Z-curve. In the following we will explain how this is not the case and why we use the Morton Z-curve instead of a Hilbert curve in our method. The white lines in Figure 1 and Figure 2 show a 2D example of each curve.

1.1 Influence on usable operations

The most frequent parent reuse operations R_p in our compression are not affected by the choice of curve as all previous levels-of-detail (LOD) are always decoded before the current LOD. The palette reuse operations P_0 and P_δ are affected by the curve but occur seldomly. The most interesting case that depends on the traversal order are the neighbor reuse operations $R_{\{x,y,z\}}$. Reusing labels from neighbors is only feasible if they have already been en/decoded. This is when the neighbor's index along the curve is lower than the current one. If this is not the case, we have to fall back to using the label of the neighbors parent which is suboptimal (see Sec. 3.2 in the paper). The number of valid, referencable neighbors on the same level along a Hilbert curve in 2D (the 3D case is analogous) is 1, 2, or 3 as shown in Figure 1. The expected number for a node is 2, except for the boundary where the expected value is $\frac{22}{15}$ as no references can be carried out over brick borders. For a Morton Z-curve, the average as well as the total number of referencable neighbors is exactly the same, but more evenly distributed as shown in Figure 2.

1.2 Influence on performance

Even though fast implementations for Hilbert curves exist, Morton indices can always be computed with fewer instructions. We heavily rely on fast indexing as indices have to be computed multiple times for each of the output elements. The regular structure of Z-curves allows for further optimizations: For decoding, we need to store temporary elements from coarser LODs (see Fig. 4 in the paper). With Morton Z-curves we can simply use larger, constant step sizes in the curve for the target LOD. With Hilbert curves such indexing is more costly. Deciding if a reference to a neighbor is possible within the current level is trivial with Morton indexing: all references to the left or downwards (in 2D, 3D analogous) are valid, while all references to the right or upwards have to refer to the neighbor's parent instead. Determining which of the possible two neighbors along an axis is referenced in the reuse operations (Fig. 3 in the paper) can be directly inferred from the Morton index. For Hilbert curves this again requires additional instructions.

1.3 Experimental results

In our experiments, we found that on some data sets the compression rate using a Hilbert curve is very slightly better (Cells, Cortex) while on other data sets, the Morton Z-curve yields better compression (Fiber). Table 1 lists compression rates and run times in comparison. Overall, the compression rates are almost identical. Compression run times, however, are twice or even up to three times as long when using a Hilbert curve instead of a Morton Z-curve and increase for larger brick sizes b. In summary, the different reasons listed in Sec. 1.2 lead to a significantly worse runtime performance of Hilbert curves compared to Morton Z-curves which does not justify the almost negligible compression rate improvements.

	Hilbert Curve				Morton Z-Curve	
	CR (%)		Time (s)		CR (%)	Time (s)
CELLS $b = 16$	3.557%	(-0.004%)	5.190s	(248%)	3.561%	2.090s
CELLS $b = 32$	3.002%	(-0.014%)	5.324s	(272%)	3.016%	1.957s
CELLS $b = 64$	2.789%	(-0.016%)	7.292s$^+$	(337%)	2.805%	2.161s$^+$
FIBER $b = 16$	1.737%	($\pm$0%)	7.645s	(197%)	1.737%	3.887s
FIBER $b = 32$	1.016%	(-0.001%)	8.024s	(214%)	1.017%	3.744s
FIBER $b = 64$	0.901%	(+0.002%)	8.360s$^+$	(260%)	0.899%	3.219s$^+$
CORTEX $b = 64$	1.443%	(-0.016%)	1101.954s$^+$	(317%)	1.459%	347.999s$^+$

Table 1: Compression rates (CR) and compression times in seconds for data sets with different brick sizes b. For Hilbert curves, a comparison with the Morton curve result is given in brackets for each value. Compression times using a Hilbert curve are two to three times as long as with Morton Z-curves while compression rates only differ at a late digit. For larger brick sizes we use 8 instead of 16 threads (marked by ()+ after the timing value).