EUROGRAPHICS 2020 / U. Assarsson and D. Panozzo
(Guest Editors)

Volume 39 (2020), Number 2

Modifying Compressed Voxel Representations - Supplementary

Supplementary Material

In this supplementary document we present auxiliary result for our work "Interactively Modifying Compressed Sparse Voxel Representations". The main paper focuses on the core method and the HashDAG data structure. It uses a select set of editing operations to evaluate the performance and behaviour of the data structure. Specifically, we have implemented several variants of the copy tool. We demonstrate some of these variants herein. *(Note: Names of figures and sections are prefixed with an 'S' in this supplementary document. References to sections and figures without the 'S'-prefix refer to material in the main paper.)*

Sa. Compressed-domain Copies

In our main evaluation (Section 4), we use the copy operation as a method for evaluating complex editing operations, that is, for evaluating insertion of complex arbitrary shapes. For this reason, we first decompress the source voxel volume and then use it as a template for the editing operation. We only measure the time of the insertion, and omit the decompression stage from our timings (see description in main paper).

For a dedicated copy-tool, it is more useful to perform copies without decompression. Such a tool would query the source DAG directly at each step in the depth-first traversal of the editing operation (see Section 3.1). Using the DAG structure as a source has the particular advantage that it encodes information about empty subtrees natively. Since we only copy non-empty voxels, we can use this information to avoid descending into nodes/subtrees that are empty in the source DAG. This significantly cuts down on traversal time when the source is very sparsely populated.

Figure S1 shows a comparison of different copy operation implementations. We always use six threads in this evaluation. The copy sequence is the same as the one in Figure 8 in the main paper. We include four variations of the copy operation:

Ref. The reference measurement is a repetition of the measurement from Figure 8, i.e, where the source volume is first decompressed. As in Figure 8, the source decompression time is not included.

Compr. (Full) Compressed copy with no empty-subtree skipping. In particular, this will check each leaf voxel of the source volume individually.

Compr. (NE) Compressed copy with empty-subtree skipping. We check for empty subtrees on all but the last two levels (the last two levels are encoded as a single 64-bit word).

Dec. (NE) Copy with initial decompression, but with empty-subtree checking for all levels except the last four.

Compressed copies add, on the one hand, more complex queries when checking the source volume. Each query needs to traverse the source DAG to determine if a certain node is occupied, resulting in multiple incoherent memory reads per query. Compressed copies without empty-space skipping are thus much more costly than our reference measurements. However, empty space skipping works very well for this test sequence.

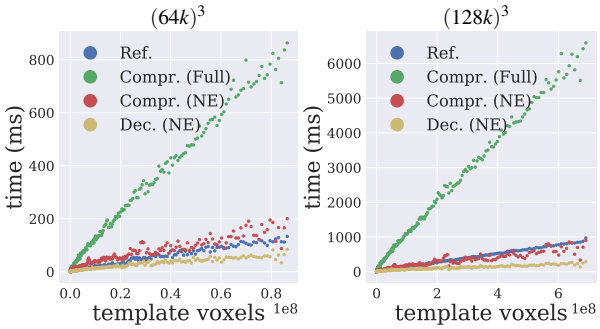

Figure S1: *Copy performance using six threads. We compare the results from the paper (*Ref.*, c.f. Figure 8 in the main paper), where the source volume was decompressed before the copy, with copy operations that source their data directly from the compressed DAG (*Compr. *variants), avoiding the decompression step. Here, early skipping of empty subtrees in the source volume improves performance significantly (*Compr. (NE)*). We include a measurement that naively combines decompression and the empty-subtree optimization (*Dec. (NE)*), indicating that further optimizations are possible.*

For a real-world application, we believe that combining the two methods might be viable, as indicated by the *Dec. (NE)* measurements. *Dec. (NE)* uses the DAG for empty-subtree checking at earlier levels, but then switches to a decompressed volume for the levels close to the leaves. Note that a real-world implementation would likely want to decompress the subvolumes close to the leaves individually, rather than decompressing the full volume upfront (which we do in this test for simplicity).

Sb. High-frequency geometry

The *San Miguel* scene contains models of two (biological) trees (see Figure 5, bottom row). The geometry of the trees shows a much higher frequency when compared to the geometry in the *Epic Citadel* scene.

In Figure S2, we display results from benchmarking copies of the tree geometry (targeting mainly their foliage). The overall behaviour is similar to that in Figure S1, but the performance varies much more. The reason is apparent from the number of voxels changed by individual editing operations. The voxel density in the trees is generally higher (up to 2×) than that of the *Epic Citadel* scene, and has a much higher variance.

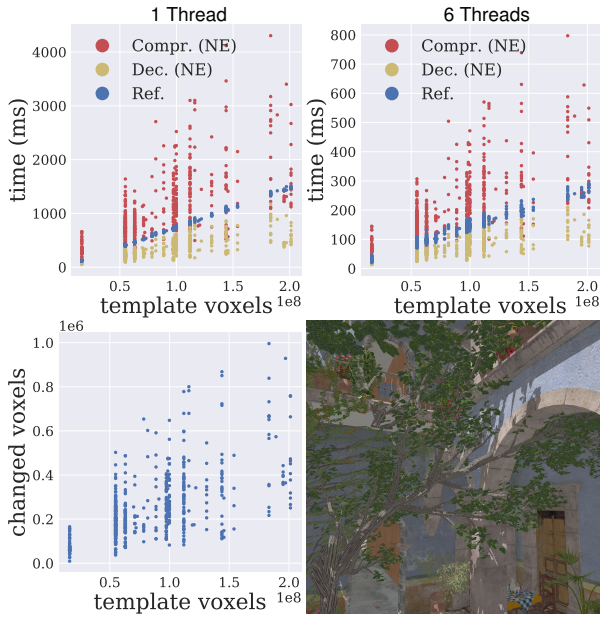

Figure S2: *Performance of copies in the* San Miguel *scene at* $(64k)^3$. *We copy parts of the tree (focusing on foliage specifically), with results of a (partial) copy shown in the bottom right screenshot. Compared to the scenario in Figure 8 and Figure S1, the geometry here is of much higher frequency and includes less empty space. We include measurements for the three copy implementations (see Figure S1; we omit the* Compr. (Full) *here), using a single thread and six threads, respectively. Similar to Figure 8, we show the total number of changed voxels in the plot to the bottom left. Note that the groupings along the x-axis correspond to edits with the same editing footprint, as selected by the user.*

Sc. Transforming Copies

The editing operation traverses the destination DAG as a full tree (i.e., including empty volumes) and determines at each step whether or not the volume represented by the corresponding node of the traversal is affected by the editing operation (see Section 3.1). This continues, for parts affected by the edit, down to the leaf level, where the method inserts/removes leaf voxels as indicated by the editing operation. A copy tool would simply store a translation between the destination and the source. Each time the copy tool queries whether or not a node is affected by it, it would take that node's volume, apply the translation, and query the source DAG with that volume.

We demonstrate a basic transforming copy by simply replacing the translation with an arbitrary transformation. Our implementation is kept intentionally simple, and is clearly suboptimal in many aspects. First, it foregoes the empty-subtree optimization that is possible with compressed copies (Section Sa). Second, we internally represent the size of the edit with a conservative axis-aligned box that encloses the transformed volume. Consequently, a large amount of compute power might be allocated to empty space. Third, the transformation is performed with a full matrix multiplication for each voxel instead of just transforming a basis. In this current implementation, the matrix multiplication leads to an approximately 10% overhead.

Figure S3: *(top/mid) Multiple copies of the statue (see Figure 1) with different scalings and rotations applied. (bottom left) The tents make it easy to see our "swirl", a position-dependent transform. The amount of rotation around the center of the tent depends on the height above ground (the middle tent shows a plain copy for reference). Such a transform would be non-trivial with a low-poly mesh, as it would likely require additional tessellation (however, the voxel version instead suffers from aliasing that is common when transforming discretely sampled data). (bottom right) Finally, we use the swirl to make the front right statue turn the upper body towards the viewer.*

Figure S3 shows a few select results of copies with transformations, including rotated and scaled copies. We also show a special "swirl", which uses a position-dependent transform.

Note that our basic implementation uses a simple non-interpolated sampling scheme (equivalent to nearest sampling of textures). This introduces some visible artefacts such as small holes/missing voxels, depending on the transformation. The errors are especially noticeable during minification, due to the thin voxel surfaces resulting from voxelizing a triangle mesh. Better sampling schemes would likely improve the results (e.g., one could use the hierarchical structure of the DAG in a manner similar to mipmapping). This is however out-of-scope for the current work.