Topology-Change-Aware Volumetric Fusion for Dynamic Scene Reconstruction
—Supplementary Material—

Chao Li and Xiaohu Guo

Department of Computer Science,
The University of Texas at Dallas
{Chao.Li2, xguo}@utdallas.edu

This supplementary file consists of:

- Details of optimization method
- Details of data structure design towards real-time performance

1 Details of Optimization Method

For the ease of discussion, we provide all notations of the math symbols of this paper in Table 1.

Alternating optimization: solve three groups of unknowns by fixing the other two groups and solve one group.

Step 1 [Fix $\{\boldsymbol{R}_i^\top\}$ and $\{l_{ij}\}$, solve $\{\boldsymbol{t_i}\}$] Set

$$\frac{\partial E_{total}(\boldsymbol{t_i})}{\partial \boldsymbol{t_i}} = \boldsymbol{0} \tag{1}$$

Solve $A^\top W A\boldsymbol{x} = -A^\top W\boldsymbol{b}$ with Preconditioned Conjugate Gradient (PCG), where $\boldsymbol{x}$ is the stacked vector of all $\boldsymbol{t_i}$ and W is the diagonal matrix of term weights.

$$A = \begin{pmatrix} & & \cdots & & \\ \cdots & \alpha_i(\boldsymbol{R}^\top \boldsymbol{n}_y)^\top & \cdots & \alpha_j(\boldsymbol{R}^\top \boldsymbol{n}_y)^\top & \cdots \\ & & \cdots & & \\ \cdots & \alpha_i \boldsymbol{R} & \cdots & \alpha_j \boldsymbol{R} & \cdots \\ & & \cdots & & \\ \cdots & l_{ij} I & \cdots & -l_{ij} I & \cdots \\ & & \cdots & & \end{pmatrix} \tag{2}$$

$$b = \begin{pmatrix} \vdots \\ \boldsymbol{n}_y^\top (\boldsymbol{T}(\boldsymbol{x}) - \boldsymbol{y}) \\ \vdots \\ \boldsymbol{T}(\boldsymbol{f}) - \boldsymbol{y} \\ \vdots \\ l_{ij}(\boldsymbol{R}_i - I)(\boldsymbol{g}_i - \boldsymbol{g}_j) \\ \vdots \end{pmatrix} \tag{3}$$

Table 1. Definitions of the math symbols used in this paper.

$\mathcal{C}_n$:color image of the n^{th} frame	$\mathcal{V}_n$:TSDF grid updated from the n^{th} frame
$\mathcal{D}_n$:depth image of the n^{th} frame	$\mathcal{G}_n$:EDG grid updated from the n^{th} frame
e_{ij}:edge of EDG connecting the i^{th} and j^{th} nodes	l_{ij}:line process parameter between the i^{th} and j^{th} nodes
$\{c^{\mathcal{V}}\}$:the cells of TSDF volume	$\{c^{\mathcal{G}}\}$:the cells of EDG
$\{g^{\mathcal{V}}\}$:the grid-points (voxels) of TSDF volume	$\{g^{\mathcal{G}}\}$:the nodes of EDG
$\{\mathbf{R}, \mathbf{t}\}$:global rotation and translation from the canonical to current frame	
$\{\mathbf{R}_i, \mathbf{t}_i\}$:local rotation and displacement of the i^{th} node from the canonical to current frame	
$\overline{\mathcal{M}_n}$:surface mesh defined in the canonical space and reconstructed from the images of first n frames	
$\mathcal{M}_n$:surface mesh of $\overline{\mathcal{M}_n}$ being warped to the space of the n^{th} frame	

Step 2 [Fix $\{\boldsymbol{t_i}\}$ and $\{l_{ij}\}$, solve $\{\boldsymbol{R}_i^\top\}$] For each $\boldsymbol{R}_i^\top$, it is a least square rigid estimation, which has closed form solution. Therefore, all $\{\boldsymbol{R}_i^\top\}$ could be solved in parallel.

First, compute the cross-covariance matrix A for all $\boldsymbol{g}_i$ corresponding terms:

$$A = XLY^\top \tag{4}$$

$$X = \begin{pmatrix} \cdots \\ \boldsymbol{g}_i - \boldsymbol{g}_j \\ \cdots \end{pmatrix} \tag{5}$$

$$L = \begin{pmatrix} \ddots & & \\ & l_{ij} & \\ & & \ddots \end{pmatrix} \tag{6}$$

$$Y = \begin{pmatrix} \cdots \\ [\boldsymbol{g}_i + \boldsymbol{t}_i - (\boldsymbol{g}_j + \boldsymbol{t}_j)]^\top \\ \cdots \end{pmatrix} \tag{7}$$

Secondly, by solving the Singular Value Decomposition (SVD) of matrix A, the optimal value of $\varDelta R_i^*$ is:

$$\varDelta R_i^* = V \begin{pmatrix} 1 & & \\ & 1 & \\ & & det(VU^\top) \end{pmatrix} U^\top, \tag{8}$$

where

$$A = U \varSigma V^\top, \tag{9}$$

Step 3 [Fix $\{\boldsymbol{R}_i^\top\}$ and $\{\boldsymbol{t_i}\}$, solve $\{l_{ij}\}$] By setting

$$\frac{\partial E_{reg}(l_{ij})}{\partial l_{ij}} = \mathbf{0} \tag{10}$$

, we have

$$l_{ij} = \left(\frac{\mu}{\mu + \|\boldsymbol{R}_i(\boldsymbol{g}_i - \boldsymbol{g}_j) - [\boldsymbol{g}_i + \boldsymbol{t}_i - (\boldsymbol{g}_j + \boldsymbol{t}_j)]\|^2}\right)^2 \tag{11}$$

.

Initialization: $\boldsymbol{R}_i^\top \leftarrow \boldsymbol{I}, \boldsymbol{t_i} \leftarrow \boldsymbol{t_i'}$(optimal $\boldsymbol{t_i}$ solved from previous frame), $l_{ij} \leftarrow 1.0$

2 Details of Data Structure Design Towards Real-Time Performance

There are several requirements to meet when re-designing the data structure of our topological-change-aware fusion framework towards real-time performance.

1. Efficient cell duplicate and merge operation.
2. Fast EDG/volume cell to node/voxel mapping and reverse mapping when duplicate cell exists.

For the second requirement, in details, fast vertex/voxel to EDG cell mapping and EDG cell to node mapping is required to compute the deformation of each vertex/voxel by trilinear interpolation based on the estimated deformation field. Fast volume cell to voxel mapping is required to do marching cubes to extract surface mesh.

2.1 Embedded Deformation Graph (EDG)

To meet all these requirement, for EDG, we add a node bucket as an intermediate level shown in Figure 1. This node bucket has a fixed size which is the max number of node copies we allow in our system. All EDG node buckets are stored in a flat vector. Given a 1D index i of EDG node, if the pointer to a node bucket in this indexed entry is null, it means this node is inactive. If the pointer is not null, it means at least one copy of this node is active. The index of a node copy could be computed as $8 * i + offset$. The following is the c++ code for our new data structure:

Listing 1.1. C++ code using listings

```
Struct Node {
    //local translation
    Vector3f translate;
    //local rotation
    Matrix3f rotate;
    Node* neighbors;
    //index1d: 1D index of the node;
    //bucket_offset: offset of in NodeBucket.
    Int2 index {index1d, bucket_offset}
    //offsets of 8 nodes sharing the same
    //cell with this node as the
    //left-front-bottom one
    array<int,8> cell_offsets;
    //Real or virtual node
    bool realOrVirtual;
    bool activeOrNot;
    //sequential id in the graph
    //only used for duplicate and merge
    int parentID;
};
Struct NodeBucket {
    Node* nodeCopies[8];
};
vector<NodeBucket *> DeformGraph;
```

In this way, each cell just needs to maintain the left-front-bottom node, by visiting "cell_offsets" and mathematically computing the "index1d" of all 8 nodes based on regular grid, we could get the mapping from the cell to all its contained nodes. The combination of "index1d" and "cell_offset" indicates the location of a node in the data structure.

After initialization, when there is no duplicate cell, each NodeBucket only contains one node copy when this node is active.

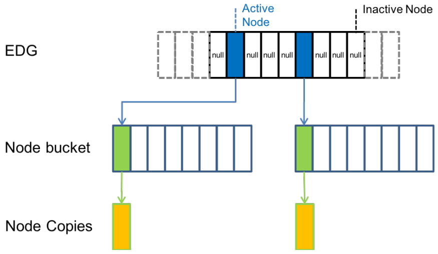

Fig. 1. Illustration of re-designed EDG data structure for topological changes.

Figure 2 and Figure 3 shows the steps to duplicate and merge EDG cells. Several strategies are used to improve the performance. First, we only consider cells containing cutting edges, which is a small portion of the entire set of active EDG nodes. In this step, new vector of NodeBucket will be created which only contains nodes from cutting cells. Secondly, in the cell duplicate step, we create node copies according to the number of connected components in each cutting node cell in EDG. Shown in Figure 3 "Cell Duplicate", the light blue node is duplicate into 4 nodes: one real node (Orange) and one virtual node (Purple) from the top cell; one real node (Green) and one virtual node (Red) from the bottom cell. Their parentIDs will be recorded, which are the offsets of the nodes that they inherit from. In the case shown in Figure 3, because there is already one node copy existing in the original EDG NodeBucket vector, the offset of new node copies starting from 1. (Orange) node and (purple) node are all real nodes and inherit from node 0, so their parentID is 0. (Green) node and (red) node are all virtual nodes and inherit from node 0, but they will not be merged to the parent node 0, so their parentID is 3, which is the offset of the (Green) node. Thirdly, in the cell merging step, we could just use UnionFind to merge all node copies of each NodeBucket individually based on their parentIDs (shown in Figure 3 "Cell Merging").

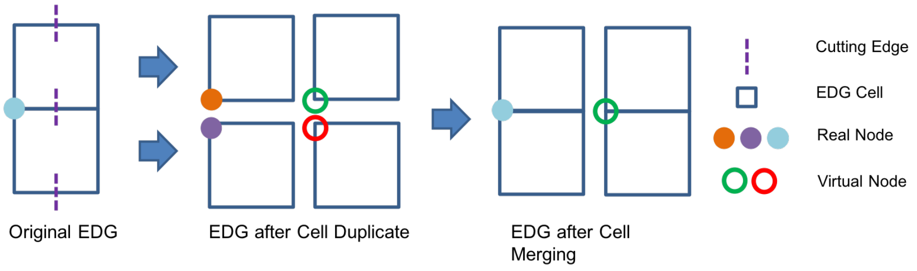

Fig. 2. Steps to duplicate EDG cells and merge them.

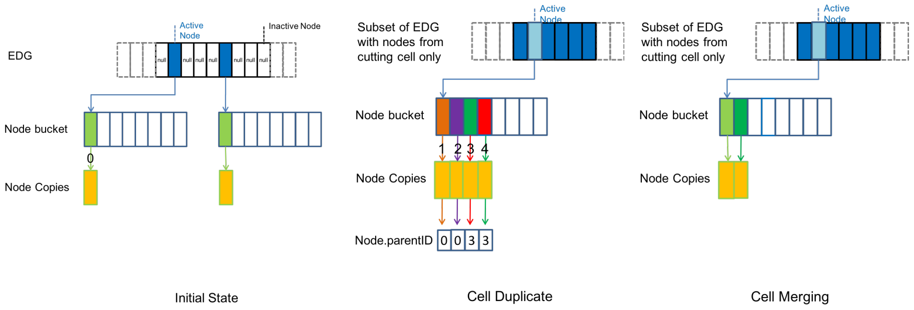

Fig. 3. Illustration of steps to duplicate EDG cells and merge them by our re-designed data structure.

2.2 TSDF Volume

We use a similar way to represent our new TSDF volume data structure. The following is the c++ code for our new data structure:

Listing 1.2. C++ code using listings

```
Struct Voxel {
    float depth;
    float weight;
    Vector3i RGB; //if needed
    Vector3f warped_pos;
    Int4 index {voxel_index1d,voxel_bucket_offset,
    node_index1d, node_bucket_offset};
    array<int,8> voxel_offsets;
    bool realOrVirtual;
    //sequential id in the graph
    //only used for duplicate and merge
    int parentID;
};
Struct VoxelBucket {
    Voxel* voxelCopies[8];
};
```

```
vector<VoxelBucket *> TSDFVolume;
```

When we doing cell duplicate and merging, the belonged EDG cell of each voxel could be recorded. When we doing marching cubes based mesh extraction, fast vertex/voxel to EDG cell mapping could be passed from voxel to vertex by recording the id of the left-front-bottom node in belonged EDG cell in "Voxel.index.node_index1d" and "Voxel.index.node_bucket_offset". Fast volume cell to voxel mapping is maintained in as similar way as the EDG cell to node mapping by using the property "Voxel.voxel_offsets".