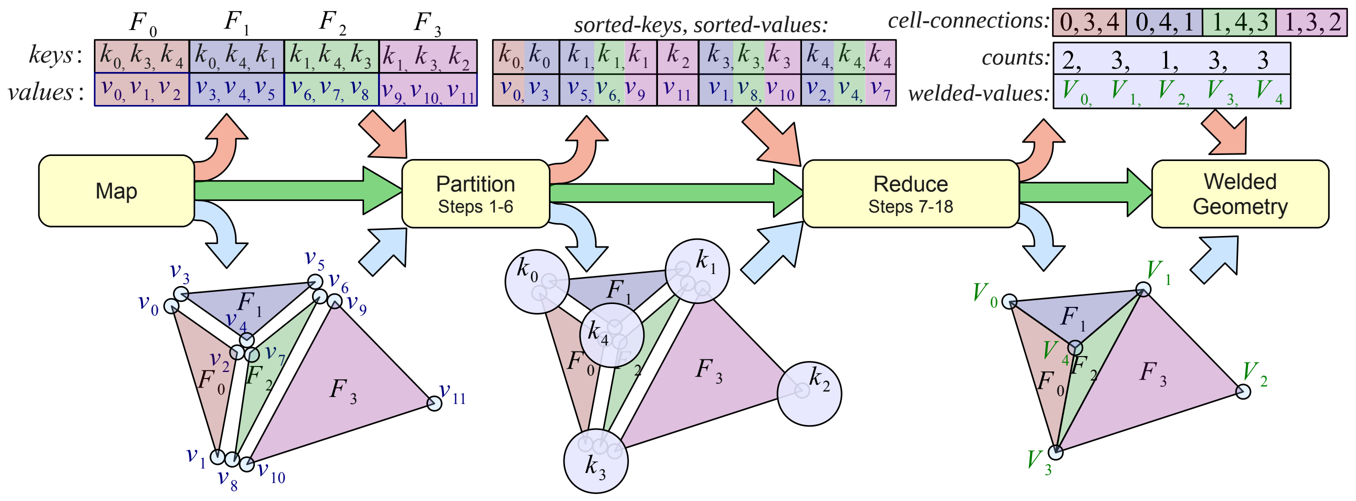

F_0
F_1
F_2
F_3
keys: k_0, k_3, k_4 | k_0, k_4, k_1 | k_1, k_4, k_3 | k_1, k_3, k_2
values: v_0, v_1, v_2 | v_3, v_4, v_5 | v_6, v_7, v_8 | v_9, v_{10}, v_{11}
sorted-keys, sorted-values:
k_0, k_0 | k_1, k_1, k_1 | k_2 | k_3, k_3, k_3 | k_4, k_4, k_4
v_0, v_3 | v_5, v_6, v_9 | v_{11} | v_1, v_8, v_{10} | v_2, v_4, v_7
cell-connections: 0,3,4 | 0,4,1 | 1,4,3 | 1,3,2
counts: 2, 3, 1, 3, 3
welded-values: V_0, V_1, V_2, V_3, V_4
Map
Partition
Steps 1-6
Reduce
Steps 7-18
Welded Geometry

Figure 1: *An example of using* KEY-WELD *with input mesh topology to find output vertex connections. After a map operation defines some cells at left, a partition phase groups the keys and a reduce phase combines vertices and establishes a connections array to the new indices.*

1 General Merging Algorithm

Figure 1 provides a simple example of applying our technique using input mesh topology, described in Section 4.1 of the paper, to find connections among generated vertices. First, a mapping operation generates the connections for a set of cells. Vertices are duplicated to allow independent operation on multiple threads. For the purposes of this paper we assume the map operation is a previously known algorithm such as Marching Cubes with the trivial extension that cell connection lists contain pairs of input index (key) and output vertex (value) rather than just the output vertices.

Next, key-value pairs are partitioned by sorting the pairs based on keys. From the sorted list of keys we can efficiently extract a list of unique keys, which serves to identify the connected vertices to be created. This list of sorted unique keys can also used to look up where each original unsorted key resides in the final list of merged vertices, which is how we generate the *cell-connections* array defining the cell topology.

The final step is to merge values with identical keys in the reduction phase. This reduction operation provides an opportunity to combine neighborhood information such as averaging normals across surface polygons. To facilitate averaging we also generate a *counts* array marking the number of cells incident to each vertex. Although this array is not specifically necessary to describe the final topology, it can be leveraged to find vertex incidence lists using the VERTEX-INCIDENCE-LIST method described in Algorithm 2.

```
KEY-WELD($keys, values, compare, merge, transform$)
$keys$: An array of keys uniquely identifying elements.
$values$: An array of mergeable elements for each key.
$compare$: A key comparator $compare(k_1, k_2) \to$ BOOL.
$merge$: A merging operator $merge(m_1, m_2) \to m_3$.
$transform$: A transformation $transform(m_1, size) \to out$.
       ▷ Sort $values$ using $keys$ to make groups.
 1     $(sorted\text{-}keys, sorted\text{-}values) \leftarrow$
           KEY-SORT($keys, values, compare$)
       ▷ Determine the new indices for each element.
 2     $unique\text{-}keys \leftarrow$ UNIQUE($sorted\text{-}keys$)
 3     $cell\text{-}connections \leftarrow$
           VECTORIZED-FIND($unique\text{-}keys, keys$)
       ▷ Get a reverse map from output to sorted arrays.
 4     $reverse\text{-}map \leftarrow$
           VECTORIZED-FIND($sorted\text{-}keys, unique\text{-}keys$)
       ▷ Get the size of each group (one per unique key).
 5     $weld\text{-}array\text{-}size \leftarrow$ LENGTH($reverse\text{-}map$)
 6     $counts \leftarrow \{\}$
 7     for $i \leftarrow 0$ to $weld\text{-}array\text{-}size - 2$
           do in parallel
 8             $counts[i] \leftarrow reverse\text{-}map[i+1]$
                           $- reverse\text{-}map[i]$
 9     $counts[weld\text{-}array\text{-}size - 1] \leftarrow$
           LENGTH($sorted\text{-}keys$)
               $- reverse\text{-}map[weld\text{-}array\text{-}size - 1]$
       ▷ Merge each group into a single element.
10     $welded\text{-}values \leftarrow \{\}$
11     for $weld\text{-}index \leftarrow 0$ to $weld\text{-}array\text{-}size - 1$
           do in parallel
12             $sort\text{-}index\text{-}start \leftarrow reverse\text{-}map[weld\text{-}index]$
13             $count \leftarrow counts[weld\text{-}index]$
14             $v \leftarrow sorted\text{-}values[sort\text{-}index\text{-}start]$
15             for $group\text{-}index \leftarrow 1$ to $count - 1$
                   do
16                     $sort\text{-}index \leftarrow sort\text{-}index\text{-}start$
                                   $+ group\text{-}index$
17                     $v \leftarrow merge(sorted\text{-}values[sort\text{-}index])$
18             $welded\text{-}values[weld\text{-}index] \leftarrow$
                       $transform(v, count)$
19     return ($welded\text{-}values, cell\text{-}connections, counts$)
```

Alg. 1: The KEY-WELD *procedure welds vertices and allows multiple local samples of vertex attributes to be merged to form a better sampling without non-local operation during geometry generation.*

```
VERTEX-INCIDENCE-LISTS(cell-ids,
                       cell-connections,
                       counts)
cell-ids: An array of cell identifiers for each cell vertex.
    Usually sequential repeated indices $\{0,0,0,1,1,1,\ldots\}$
cell-connections: Result from KEY-WELD.
counts: Result from KEY-WELD, cell counts per vertex.

    ▷ Use a key-sort with cell connections as the keys to
    ▷ group cell identifiers by vertices.
1   $(-, links) \leftarrow$ KEY-SORT(cell-connections, cell-ids)
    ▷ The size of each incidence list is stored in count.
2   links-counts $\leftarrow$ counts
    ▷ Use an exclusive scan to find the offset to the
    ▷ incidence list for each vertex.
3   links-offsets $\leftarrow$ EXCLUSIVE-SCAN(links-counts)
4   return (links, links-counts, links-offsets)
```

Alg. 2: VERTEX-INCIDENCE-LISTS *takes the output of the* KEY-WELD *algorithm and quickly generates an incidence list, represented as an array of* cell-ids *and an array of pointers into* cell-ids *called* links-offsets *that represents the start of each cell's incidence list. Adjacency lists across edges may be produced by removing incidences that do not occur exactly twice in a cell, and adjacency lists across faces may be generated similarly.*

```
VERTEX-WELD(vertices)
vertices: Array of vertex data (i.e. coordinates).
    ▷ Sort vertices to make identical vertices adjacent.
1   sorted-vertices $\leftarrow$ LEXICOGRAPHIC-SORT(vertices)
    ▷ Copy first element of each group of duplicates.
2   welded-vertices $\leftarrow$ UNIQUE(sorted-vertices)
    ▷ For each item in vertices find the corresponding
    ▷ index in welded-vertices.
3   cell-connections $\leftarrow$
        VECTORIZED-FIND(welded-vertices, vertices)
4   return (welded-vertices, cell-connections)
```

Alg. 3: VERTEX-WELD, *as demonstrated by Bell, takes geometric soup as input and produces a welded topology.*

1 Nonmanifold Surfaces

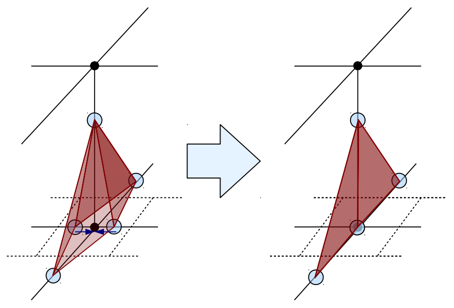

Figure 1: *An example of mesh coarsening causing a non-manifold, overlapping surface.*

A problem with the coarsening technique is that in some cases, such as the case shown in Figure 1, non-manifold surface elements may be generated by collapsing vertices from different parts of the surface. In this example, two vertices from different faces are merged and a 3D volume is collapsed into a single plane. If such surface elements are undesirable, they may be eliminated by calculating the maximum edge curvature of each triangle, and removing all triangles containing an edge curvature of 180°.