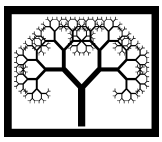
Paper 40



Proceedings of the Fourth International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, P. Iványi and B.H.V. Topping (Editors), Civil-Comp Press, Stirlingshire, Scotland.

A Surface Reconstruction Algorithm for Real-Time Swept Volume Computation

A. Leutgeb, M.F. Hava and B.M. Gruber
Industrial Software Applications
RISC Software GmbH
Hagenberg, Austria

Abstract

During the simulation of subtractive manufacturing processes the real-time generation of exact swept volumes is essential. While low quality approximations have been available for some time, the just-in-time reconstruction of high quality swept volumes has not been possible until recently due to the high computational costs of such an approach. A point cloud based approach is presented in this paper, which has no limitations regarding complexities of the tool geometry and the tool trajectory. The main contribution of the paper is a parallel implementation of the ball pivoting algorithm (BPA) for surface reconstruction. For a given point cloud with uniform point density the algorithm constructs a water-tight surface. Because of its high triangle throughput and good scalability, it is a well-fitting method for high quality swept volume computation in real-time.

Keywords: ball pivoting, surface reconstruction, point cloud, parallel, swept volumes, real-time, regular grid.

1 Introduction

For the simulation of subtractive manufacturing processes the computation of swept volumes, generated by cutter movements along a specified trajectory (tool path), is essential. During the simulation swept volumes are removed from the geometric model of the raw stock, resulting in the final machined surfaces or work piece models. The swept volume is often defined by the analytical or discrete description of its surface, the swept envelope. An overview of existing swept volume computation methods is given in [1]. Methods based on analytical approaches suffer from certain limita-

tions like the reconstruction of sharp edges/points on the surface, different algorithms for specific cutter types, and/or the impossibility to obtain the swept volume as a whole closed form model. These potential limitations for general applicability motivate approximative approaches. Our method is based on surface reconstruction of point clouds (see [2]).

Thereby the point cloud is constructed in the following way. The triangle polyhedral boundary representation (B-rep) of the cutter geometry is an approximation of the real tool geometry with a specified model accuracy defined by the maximum edge length of a triangle. We declare a point as the combination of a vertex and an associated oriented normal vector. Each point of the cutter geometry mesh is repeatedly sampled along the trajectory (based on individual transformations described in [3]) according to a specified sampling density equal to the model accuracy. Points contributing to the sweep envelope (for envelope theory and tangent condition property see [4]) are added to the point cloud.

Our main contribution is a parallel variant of the region growing based Ball Pivoting Algorithm (BPA) proposed by F. Bernardini et al. in [5], appropriate for efficient surface reconstruction of such point clouds. For the purpose of parallelization the problem of surface reconstruction is divided into several independent sub-problems by dividing the point cloud into individual *work slices* along a spatial dimension. The work slices are separated by *stitching slices*. At first the unconnected surfaces in the work slices are constructed in parallel. Then stitching of pairs of adjacent surfaces starts in parallel using the points inside the stitching area resulting in new surfaces. This parallel stitching process is repeated until the final surface is created. The implemented algorithm provides good strong scalability. The high triangle throughput of the surface reconstruction makes it a good candidate for real-time swept volume computation. Our approach outperforms the parallel octree based BPA implementation of Julie Digne (see [6]) regarding the absolute number of triangle throughput and scalability, as we see later on in the result section.

2 Related work

Surface reconstruction algorithms are widely used in reverse engineering and industry design. Thereby the surface is reconstructed from scanned 3D point cloud data. In our case point clouds are generated algorithmically, so that the following conditions apply:

- The point density is uniform (no under sampling).
- Each point has an associated oriented normal vector.
- The point cloud contains neither noise nor outlier points.

Points are added to the point cloud based on envelope theory and tangent condition property (see [4]). Nevertheless it may contain points not contributing to the sweep envelope, because of discretized tool geometries and trajectories. Furthermore arbitrary

trajectories lead to self intersecting swept volumes (see [2]). Therefore any surface reconstruction algorithm must satisfy the following constraints:

Constraint 1: Points inside self intersecting regions must be ignored.

Constraint 2: Points not contributing to the sweep envelope because of discretization, must be ignored.

In this section we give a brief overview of several surface reconstruction methods divided into the three categories: implicit, Voronoi/Delauney based, and region growing based.

Implicit surface reconstruction methods use implicit mathematical functions to interpolate the points of the point cloud. The resulting mesh is the triangulated surface of these functions. These methods have in common, that they produce a water-tight surface, even in the case of sparse and noised data. The generated surfaces may not pass through all points of the point cloud resulting in the loss of details. Because implicit surface reconstruction methods do not conform to our surface reconstruction constraints and have high computational costs (see [7]), they are not appropriate for real-time swept volume computation.

Voronoi/Delauney based surface reconstruction methods use the structures of the Delauney triangulation/Voronoi diagram of the points to drive the surface reconstruction process. The computation of these structures has a run time complexity of $O(n^2)$, where n is the number of points of the point cloud. One prominent representative is Tight Cocone (see [8]), which comes with the guarantee to construct a water-tight surface provided that under sampling is local and noise is within a certain tolerance. Because Voronoi/Delauney based methods have high computational costs, they are not appropriate for real-time swept volume computation.

The basic principle of the region growing based ball pivoting algorithm (BPA) proposed by F. Bernardini et al. in [5] is as following. For a point cloud with a specified uniform point density, a ball with a certain radius β is placed on the point cloud touching three outside points, which form a triangle (called *seed triangle*). Then the ball is pivoted around each edge of the triangle building new triangles with the pivoting edge and the new touched points. This pivoting procedure is repeated for all new triangles until no more pivoting edges are left and we finally obtain a closed water tight surface. Our implementation stops processing after one surface was reconstructed (not considering all points of the cloud). Because we have a uniform point density the ball cannot pass through the point cloud during pivoting and therefore conforms to the surface reconstruction constraints. The BPA has a runtime complexity of $O(n)$, where n is the number of points, which makes it a perfect candidate for real-time swept volume computation.

Besides the BPA there are other region growing based algorithms. The work of Xiaokun Li et al. [9] proposes a priority driven approach, where growing in directions of flat areas is preferred. The selection of candidate points for the creation of new triangles is based on a cost function considering the new triangle's inner angles, the

surface continuity between the originating triangle and the new one, and the distance of the candidate point to the front edge. This cost function guarantees a good quality of the generated triangles and robustness against noise. The processing speed of the priority driven approach is slightly higher than that of the BPA.

Another promising region growing based approach regarding processing speed was introduced by L. Di Angelo et al. [10]. This method is based on the Gabriel 2 - Simplex (G2S) criterion, which requires a triangle's smallest circumscribing ball to be empty. If during the growing the end points of the front edge (i.e. pivoting edge) and any candidate point conform to the G2S criterion, a new triangle is generated. This work was further improved by L. Di Angelo et al. [11] concerning the robustness of the approach in the presence of locally non-flat and not sufficiently sampled point clouds. Thereby a priority driven region growing and post-processing for erasing non-manifold vertices were introduced. All of the mentioned region growing surface reconstruction methods, except the BPA, presume surface smoothness and therefore do not conform to the surface reconstruction constraints.

The work of Julie Digne [6] provides another parallel implementation of the BPA. The problem of surface reconstruction is divided into independent problems in the following way. The points of the point cloud are stored inside an Octree space partitioning structure, where for a given ball radius β the smallest cells of the octree have size $\delta > 2\beta$. These smallest octree cells represent the work unit per thread. For all cells one level above the smallest cells, the child cells are put into individual sets identified by the child index. So we have eight sets of smallest cells, where the cells of each set are not adjacent and can be processed in parallel without any data synchronization.

The major difference to our approach is the granularity of work units per thread and the the maximum number of work units. In our approach the maximum number of work units is equal to the number of operating system threads, whereas in the work of Julie Digne the maximum number of work units is a function of the spatial extent of the point cloud and the smallest cell size δ, which is in any case significantly higher. For each work unit a seed triangle must be identified, which is more time consuming than expanding the surface. So the run-time of Julie Digne's algorithm is more dominated by finding the seed triangle, than in our approach. The result section compares the two implementations regarding the surface reconstruction speed (number of triangles per second) and the scalability (strong scaling).

3 Acceleration structure

In order to guarantee a fast search for candidate points and their validation when pivoting an edge, the use of an acceleration structure is required. Our implementation is based on an axis-aligned regular grid (uniform space decomposition), consisting of $X \times Y \times Z$ cells. In addition to its fundamental purpose of speeding up the pivoting itself, the grid is also crucial for our parallelization approach, which will be detailed in the next section.

For the construction of an axis-aligned regular grid the axis-aligned bounding box (AABB) of the point cloud we want to reconstruct is computed (see algorithm 1). To minimize the number of cells needed during candidate lookup, our cells are sized corresponding to the radius of the ball. From the AABB and the preferred cell size we can derive the number of cells along each axis. All points are then mapped into them based on their coordinates.

Algorithm 1 constructGrid(P, M), Construction of an Axis-Aligned Regular Grid From a Point Cloud.

Input: A point cloud P; a guaranteed maximum point distance M
Output: An axis-aligned regular grid G containing all points of P

1: $G \leftarrow newGrid()$
2: $G.radius \leftarrow M$
3: $G.cells \leftarrow newCells\left(\left\lceil \frac{maxDim(P)-minDim(P)}{M} \right\rceil\right)$
4: **for all** $p \in P$ **do**
5: $\quad c \leftarrow cellFromPoint(p)$
6: $\quad$ Add p to c
7: **end for**
8: **return** G

4 Slicing and stitching

Parallel surface reconstruction based on this data structure can be implemented in a divide and conquer fashion by slicing the grid along its longest dimension. Each of the resulting work slices is used as input for an independent BPA invocation, therefore allowing us to run multiple surface reconstructions on the same point cloud in parallel.

Whilst this approach is theoretically sound, it does not prevent race conditions on points belonging to triangles which span multiple work slices. To avoid any kind of synchronization during the surface reconstruction, we introduce the concept of stitching.

Between the individual work slices we define stitching slices, where no surface reconstruction may occur. All points lying inside a stitching slice get marked to prevent pivoting. In order to prohibit all situations where a candidate point is accessed from two adjacent work slices, a stitching slice must be at least as big as the diameter of the ball. A parallelized variant of the BPA is executed on the distinct work slices.

4.1 Seeding

Before the actual triangulation of a work slice can begin, we need to determine a seed triangle per work slice. For this we introduce algorithm 2 that finds a seed triangle in the cells of the work slice. The algorithm uses a brute-force approach to find a combination of three compatible points, with a maximum distance of the radius of the

ball. Points are compatible when their vertex normals point into the same half space. The three points form a valid seed triangle when a sphere, which has the radius of the ball and is located in the circumsphere center of the triangle, contains no other points. Additionally it is enforced that none of the points of the seed triangle may be marked for stitching.

Algorithm 2 findSeedTriangleInRange(G, R), Finding a Seed Triangle inside a Work Slice.

Input: The axis-aligned regular grid G to search in; the range R of valid cells to use for search
Output: A seed triangle

for all $i \in R$ **do**
  for all $p \in G.cell_i$ **do**
    if p is not marked for stitching **then**
      $n \leftarrow reachableNeighbours(p, G, G.radius \times 2)$
      $n' \leftarrow sortByAscendingDistance(p, n)$
      if $n'.count \geq 2$ **then**
        for all $(q, s) \in n$ **do**
          if q or s is marked for stitching **then**
            restart with next p
          else
            if (p, q, s) are compatible **and** (p, q, s) has empty ball **then**
              return (p, q, s)
            end if
          end if
        end for
      end if
    end if
  end for
end for
return no seed triangle can be constructed in R

If for whatever reason no seed triangle can be constructed in a work slice, our algorithm does not actually abort the surface reconstruction, but simply defers the triangulation for that particular work slice to the stitching phase. This fallback based approach allows us to reconstruct the surface, albeit with decreased performance, if at least one seed triangle for the whole point cloud has been found.

4.2 Triangulating

After a seed triangle has been found, we start the triangulation for a work slice (see algorithm 3), with the three edges of the seed triangle as initial input, pushed on the stack E. The triangulation phase ends when the stack of remaining edges is empty. Before an edge is processed, it's validated that:

1. The edge is not classified as inner (already connected to two triangles) or boundary (connected to one triangle, with no further available candidates).

2. None of the edge's points is marked as belonging to a stitching slice.

Criterion 1 is necessary as the classification of an edge on the stack can be modified transparently by a preceding iteration. See [5] and [6] for extensive information on edge/point classifications. If criterion 2 does not hold, the edge is stored for later use during the stitching phase.

Algorithm 3 expandTriangulation(G, M, E), Expanding the Triangulation in a Work Slice.

Input: The axis-aligned regular grid G to search in; the mesh M to add triangles to; a stack of available edges E to pivot

Output: A modified M; a stack B of edges that are currently not pivot-able

```
1:  B ← newStack()
2:  while E is not empty do
3:    e ← E.pop()
4:    if e.kind = FRONT then
5:      if e is marked for stitching then
6:        Add e to B
7:      else
8:        c ← findCandidate(G, e)
9:        if c does not exist then
10:         e.kind ← BOUNDARY
11:       else
12:         t ← newTriangle(e, c)
13:         Add t to M
14:         e.kind ← INNER
15:         e.triangle_2 ← t
16:         completeTriangleEdges(E, e, c, t)
17:       end if
18:     end if
19:   end if
20: end while
21: return (M, B)
```

If both criteria are fullfilled (line 8), an attempt is made to determine the next candidate point (see algorithm 4). The candidate will always be in the close neighbourhood of the pivoted edge. A valid candidate has to fulfill several requirements:

1. It may not be an inner point (i.e. a point whose adjacent edges are marked as inner).
2. The point and the edge must be compatible, with compatible meaning that the surface normal of the candidate triangle must point into the same half-space as all three vertex normals of the candidate triangle.
3. The ball may not contain any additional points.

Out of all candidates fulfilling these requirements, the one with the smallest rotation angle from old to new position is selected and used to expand the triangulation.

Algorithm 4 findCandidate(G, E), Finding a New Candidate Point For an Edge.

Input: The axis-aligned regular grid G to search in; the edge E to find a new candidate for
Output: A new candidate point C

$c \leftarrow calculateCircumSphereCenter(E.triangle_1, G.radius)$
$a \leftarrow c - E.midpoint$
$n \leftarrow reachableNeighbours(E, |a| + G.radius)$
$A \leftarrow 2\pi$
$C \leftarrow UNDEFINED$
for all $p \in n$ **do**
  if $p.kind \neq INNER$ **and** p is compatible with E **then**
    $c' \leftarrow calculateCircumSphereCenter(newTriangle(E, p), G.radius)$
    if c' exists **then**
      $b \leftarrow c' - E.midpoint$
      $\alpha \leftarrow acos(\hat{a} \cdot \hat{b})$
      if $\alpha < A$ **and** (E, p) has empty ball **then**
        $C \leftarrow p$
        $A \leftarrow \alpha$
      end if
    end if
  end if
end for
return C

If no candidate could be determined, the pivoted edge is marked as boundary. Otherwise the edge is marked as inner and a new triangle is created, exported and connected to the pivoted edge. Before continuing with the next edge from the stack, the two other edges of the newly created triangle must be re-classified (see algorithm 5).

Algorithm 5 completeTriangleEdges(S, E, P, T), Handle All Edges of The New Triangle.

Input: A stack of Edges S to store new edges in; the pivoted edge E; the selected point P for the new triangle T
Output: A modified S

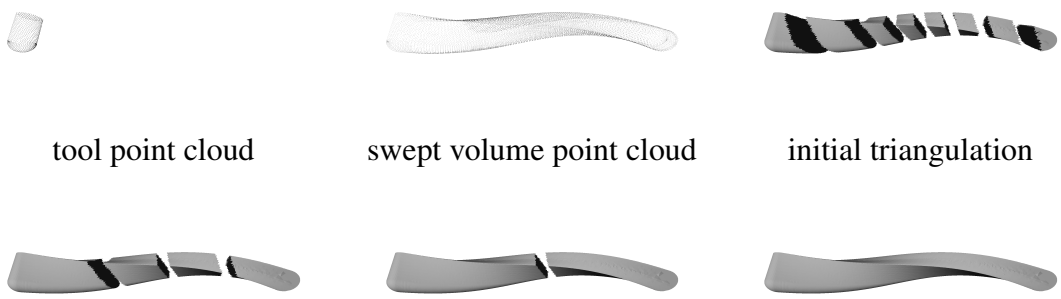
for all $p \in E$ **do**
  $x \leftarrow findExistingEdgeBetween(p, P)$
  if x exists **then**
    $x.triangle_2 \leftarrow T$
    $x.kind \leftarrow INNER$
  else
    $x' \leftarrow newEdge(p, c)$
    $x'.triangle_1 \leftarrow T$
    $x'.kind \leftarrow FRONT$
    $S.push(x')$
  end if
end for

The two edges to consider are those between the edge's source/target and the selected candidate point. For both of these we need to determine if they are already registered - meaning they are currently classified as front edges (connected to one triangle and not pivoted yet) - or if we have to register new edges. Preexisting edges get assigned their second triangle and are marked as inner edges, preventing further pivotation if they are later popped from the stack. At this point non-existing edges have to be created. These new edges get their first triangle assigned, are marked as front edges, and are pushed onto the stack to be used in a subsequent iteration.

4.3 Stitching

When the initial surface reconstruction for all work slices is completed, the parallel stitching phase begins. Distinct pairs of adjacent work slices are stitched via their shared stitching slice, by utilizing the algorithms 3, 4 and 5 again. Before the pivoting starts, all points inside the shared stitching slice are unmarked, which in turn removes the flag from a part of the previously stored edges and therefore allows the algorithm to pivot them this time. As the same algorithm as in the initial triangulation is used, the algorithm will end once all input edges have been pivoted or deferred to the next stitching phase.

The two stitched work slices are merged into one combined work slice that encompasses the area of both initial work slices and their shared stitching slice. The stitching and merging of slices is repeated with decreasing amounts of parallelization, until only one comprehensive work slice remains - and therefore all stitching slices and stitching edges have been removed (see figure 1 for the phases of the algorithm).



tool point cloud | swept volume point cloud | initial triangulation

first stitching | second stitching | final stitching

Figure 1: Visualization of slicing and stitching phases.

5 Optimal parallelization

Our concept for parallelization allows for some flexibility, as whilst the size of stitching slices is fixed, the size for work slices can be set arbitrarily, though making the work slices too big will limit the amount of parallelism (and speedup) we can achieve. Based on the longest dimension of our grid N, the size of a single stitching slice S and the minimal size of a work slice n, we can determine the number of work slices W with equation 1, which additionally ensures that the amount of work slices never exceeds the amount of available hardware threads T.

$$W = min\left(T, \frac{N+S}{S+n}\right) \tag{1}$$

Maximizing the amount of work slices based on this equation is essential to the performance of the initial triangulation phase and the subsequent stitching phases.

6 Results

For benchmarking we use high quality point clouds of several popular scenes (see figure 2), using different radii. For better reproducibility of our results, all scenes were normalized into a unit bounding box with length 1 and centered at the origin. Therefore all given radii are not only absolute lengths, but can also be seen as ratios relative to the bounding box size.

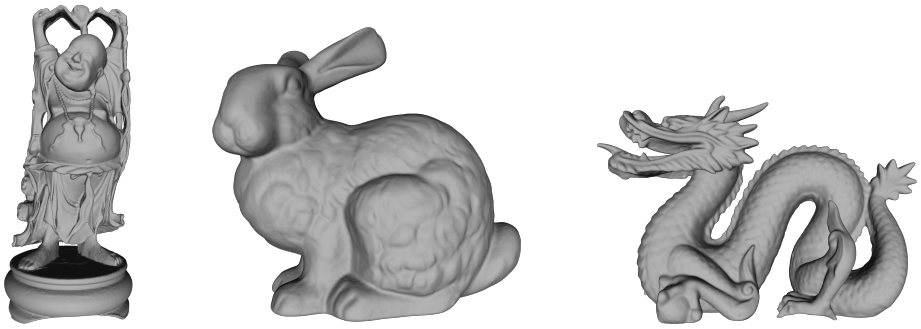


Figure 2: Popular benchmark scenes used for performance tests.

All our tests were done using a system with a single Intel Core i7-3770 processor (base clock frequency: 3.4GHz, max turbo frequency: 3.9GHz, number of cores: 4). To compare our approach (Slice BPA) with a preexisting parallel octree based BPA implementation (Octree BPA), we used the freely available source code of [6]. The exact same input and hardware were used when generating the comparison data. All results of our benchmarks are presented in table 1.

scene			Slice BPA			Octree BPA		
points	radius	T	time[s]	kΔ/s	speedup	time[s]	kΔ/s	speedup
Bunny 508,508	0.006	1	10.63	95.4		40.71	25.0	
		2	6.57	154.6	1.6	23.82	42.7	1.7
		4	4.06	250.2	2.6	15.66	64.9	2.6
	0.004	1	8.01	126.8		18.34	55.4	
		2	5.23	194.1	1.5	12.80	79.4	1.4
		4	3.52	288.3	2.3	8.61	118.1	2.1
Dragon 486,388	0.006	1	11.78	77.9		134.33	6.8	
		2	7.25	126.4	1.6	108.07	8.5	1.2
		4	5.72	160.4	2.1	93.89	9.8	1.4
	0.004	1	7.71	122.6		33.03	28.6	
		2	4.85	194.9	1.6	23.35	40.5	1.4
		4	4.29	220.4	1.8	20.06	47.2	1.6
Buddha 589,451	0.006	1	20.17	48.9		386.10	2.6	
		2	12.55	78.6	1.6	260.88	3.8	1.5
		4	7.35	134.2	2.7	210.53	4.7	1.8
	0.004	1	12.09	90.2		75.49	14.5	
		2	7.62	143.1	1.6	63.37	17.2	1.2
		4	4.62	236.1	2.6	53.15	20.6	1.4

Table 1: Runtime results and comparison.

Our results include the cumulative time for building the acceleration structure and reconstructing the surface, the kilo-triangles emitted per second and the speedup when

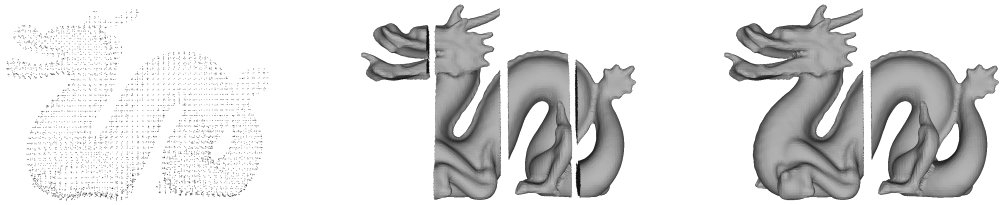
increasing the thread count. Note that for the comparison algorithm, we ensured that the same code path is used for both single-threaded and multi-threaded execution (by always specifying the '-p' option), as otherwise the scalability of the algorithm can't be measured reliably.

The first observation is that our algorithm fares substantially better in terms of overall runtime when compared to the competing algorithm and allows us to emit significantly more triangles per second. Another observation is that our algorithm provides good scalability. If we look at the speedup values, we will notice a scene-dependent variation. This is a known characteristic of our current approach, that will be explained in the next section.

7 Conclusion

Our approach of parallel surface reconstruction via the BPA works during the initial phase embarrassingly parallel on the different work slices, and afterwards needs a number of stitching phases which is proportional to $\log W$. It provides an overall good scalability (strong scaling) regarding the speed up with a different number of threads and is appropriate for real-time swept volume computation.

One observation is the variation of the speedup depending on the different test scenes. This is due to the shape of the point cloud inside a work slice, which may consist of several unconnected sub point clouds. In each work slice the BPA based region growing process is only executed once. So other unconnected sub point clouds are not considered for surface reconstruction and must be reconstructed in the later on stitching phase as illustrated in figure 3.



dragon point cloud initial triangulation first stitching

Figure 3: Illustration of reconstruction process with two unconnected sub point clouds in first work slice.

During the initial triangulation of the first work slice only the head of the dragon was reconstructed, but not the body. The missing body part is later on reconstructed in the first stitching phase. This increases the run-time of stitching and results in a reduction of the speed up. In order to overcome this problem, we will improve the processing of the work slices by reconstructing the surfaces for all unconnected sub point clouds.

Another planned improvement is to combine the several parallel stitching phases into one parallel stitching phase. Given n work slices the stitching is done by $n-1$ threads in parallel. Each stitching thread expands the regions by using the front edge collections of his adjacent work slices. Because we have a spatial separation inside the work slice front edge collections between border edges of the left and right stitching slice, during processing no data synchronization is necessary. So the run-time for surface reconstruction is dominated by the initial triangulation and we should achieve near linear speed up with a different number of threads.

Acknowledgements

This research was funded within the scope of the EU-program "Regionale Wettbewerbsfähigkeit OÖ 2007-2013 (Regio 13)" by the European Regional Development Fund and the state of Upper Austria.

References

[1] K. Abdel-Malek, J. Yang, D. Blackmore, K. Joy, "Swept Volumes: Fundation, Perspectives, and Applications.", *International Journal of Shape Modeling*, 12 (1): 87–127, 2006.

[2] X. Gong, "Triangle-mesh based cutter-workpiece engagement extraction for general milling processes", Master's thesis, University of British Columbia, 2013.

[3] S. Abrams, P.K. Allen, "Computing swept volumes", *The Journal of Visualization and Computer Animation*, 11(2): 69–82, 1999.

[4] W. Wang, K. Wang, "Geometric modeling for swept volume of moving solids", *Computer Graphics and Applications*, 6(12): 8–17, 1986.

[5] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, "The ball-pivoting algorithm for surface reconstruction", *IEEE Transactions on Visualization and Computer Graphics*, 5(4): 349–359, 1999.

[6] J. Digne, "An Analysis and Implementation of a Parallel Ball Pivoting Algorithm", *Image Processing On Line*, pages 149–168, 2014.

[7] N. Rathore, R. Gupta, "Review on Surface Reconstruction Algorithms", *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(3): 288–291, 2014.

[8] T.K. Dey, S. Goswami, "Tight Cocone: A Water-tight Surface Reconstructor", in *SM '03 Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 127–134. ACM, 2003.

[9] X. Li, C.Y. Han, W.G. Wee, "On surface reconstruction: A priority driven approach", *Computer-Aided Design*, 41(9): 626–640, 2009.

[10] L.D. Angelo, P.D. Stefano, L. Giaccari, "A new mesh-growing algorithm for fast surface reconstruction", *Computer-Aided Design*, 43(6): 639–650, 2011.

[11] L.D. Angelo, L. Giaccari, "A fast algorithm for manifold reconstruction of surfaces", in *Proceedings of the international conference on innovative methods in product design*. Libreria Internazionale Cortina Padova, 2011.