EUROGRAPHICS 2016 / J. Jorge and M. Lin
(Guest Editors)

Volume 35 (2016), Number 2

Supplemental Material - Adapting FCNs to a Prescribed Scale

Anne Gehre Isaak Lim Leif Kobbelt

Visual Computing Institute, RWTH Aachen University

Abstract

In the supplemental material we add timings and values of the optimization with Gurobi [GO15], for each of the presented models. Also, we show further comparisons to curvature filtering. Furthermore, we give detailed insights on the implementation of the feature curve network abstraction by providing pseudo-code for the entire procedure.

1. Timings and Gurobi Optimization Analysis

Table 1 gives exact details on the iterations of the FCN computations including timings, variables, constraints and energy values per iteration. The number of iterations required ranges from 1-6.

2. Results

In Figure 1 we show further comparisons to curvature thresholding and filtering. In case of the triangle meshes we threshold absolute maximal curvature values. Also, for the candle we use curvature thresholding as described in [YBS05]. Especially, for the Skyscraper we can observe that either all features are preserved or removed since they align along surface elements with a dihedral angle of 90 degrees. Hence, all curvature values have about the same magnitude. For the Candel model the flame is preserved until all other features are suppressed, because it has very high curvature values. With the method described in [YBS05], this is not the case since they incorporate the segment length into their threshold. Nevertheless, we can still not control the feature density. E.g. by increasing the threshold so that the small scale details are removed, all other features with values below this threshold also disappear (e.g. top of the candel). With our subsampling method, all features that can be represented in the given resolution (i.e. target edge length) are preserved. E.g. the flame is suppressed, while larger features (e.g. top of the candle) are preserved.

For the filtering and thresholding of curvatures for the quadmeshes we used [BZK09] with a filter-kernel radius of $r_{\min}/2$. In the top rows of Figure 1 our method is depicted. Below we apply curvature thresholding with a threshold, where all important features are included. In the resulting quad meshes we can observe that this can lead to over-constrained parametrizations (e.g. the elephants tail degenerates). Also, the ears of the Elephant and the eyes of the Camel are regions with high feature density, which can lead to bad element quality if the respective feature directions do not align well (as can be observed in the respective models). Then if we further increase the threshold to avoid this effect, all other features with lower curvature (e.g. on the body of the Elephant/Camel) are suppressed as well, leading to bad alignment of the elements. In contrast our method avoids regions with high feature density, i.e. all less significant features that are closer than the minimum scale are suppressed by stronger features. At the same time weaker feature curves that are not in conflict with any closer feature are preserved (as the curves along the body of the Elephant/Camel).

3. Pseudo Code

In the following we give the pseudocode for the entire method. Parts which were discussed in more detail in the paper (e.g. computation of weights) are given only as an overview here.

Four Step Abstraction Loop The procedure COMPUTEFCN includes the four-step loop. The sets C_e and C_v contain the edge and vertex conflicts as pairs of edges/vertices.

procedure COMPUTEFCN(FCN = $(V, V^*, E, A), r_{\min}, r_{\max}$)
  COMPUTESURFACEPROPERTIES(FCN)
  do
    RESAMPLEFCN(FCN, $r_{\min}$, $r_{\max}$)
    $C_e, C_v \leftarrow$ COMPUTECONFLICTS(FCN)
    SINGLEEDGEWEIGHTS(FCN)

      ▷ includes optimization, edge removal, and collapse
    RESOLVECONFLICTS(FCN, C_e, C_v)
  while $|C_v| \neq 0$ or $|C_e| \neq 0$
end procedure

Weights The procedure COMPUTESURFACEPROPERTIES precomputes properties of the surface as curvature values. The function SINGLEEDGEWEIGHTS sets the property *weight* of each edge $e \in E$. Exact weighting factors are described in the paper.

procedure COMPUTESURFACEPROPERTIES(FCN = $(V, V^*, E, A), \mathcal{M}$)
  COMPUTECURVELENGTHS(FCN)
  COMPUTELOOPS(FCN)

Mesh	$r_{\min}$	Time/Iteration in s	Variables/Iteration	Constraints/Iteration	Energy/Iteration$(/10^4)$
Moai	0.2	0.078/0.005/0.002/0.002/0.002	6201/4852/4547/4451/4430	6107/3439/2916/2812/2783	15.84/15.18/14.34/14.33/14.33
	0.4	0.098/0.002/0.001/0.001/0.001/0.001	3806/1746/1440/1385/1368/1354	7006/1488/864/815/809/794	8.783/7.975/5.806/4.823/4.742/4.257
	0.5	0.361/0.001/0.0005/0.0005	3423/1101/894/882	7790/950/516/492	9.469/5.697/4.981/4.982
Octaflower	0.06	0.0023/0.00038	900/772	785/568	11.97/11.97
	0.08	0.0024/0.0002	710/572	646/408	9.144/9.144
	0.12	0.0029/0.0002	429/282	447/184	5.589/5.574
	0.2	0.0023/0.0001	224/104	267/48	3.1549/3.011
Candel	0.02	0.004/0.001	4025/3936	3218/3072	25.17/25.15
	0.04	0.026/0.0006	2037/1358	2194/1028	8.296/8.277
	0.09	0.151/0.0007	955/391	1725/311	2.49/2.485
Trumpet	0.06	0.018/0.0007	2431/1992	2395/1416	6.344/6.324
	0.09	2.15/0.0008	2112/1056	3192/696	3.788/3.019
	0.2	5.71/0.0001	1104/196	6576/116	0.7566/0.5375
Fandisc	0.026	0.028/0.002/0.001	4312/3948/3941	3687/2935/2920	18.92/18.71/18.71
	0.045	0.018/0.001/0.0007	2364/1901/1886	2309/1298/1284	11.36/10.8/10.8
	0.13	0.057/0.001	874/365	1742/287	2.82/2.02
Skyscraper	0.009	4.21/0.06/0.03/0.02	112802/66614/64246/64231	151032/53837/47926/47900	208.18/206.82/206.67/206.67
	0.022	49.6/0.02/0.006	46892/12536/11053	155979/11383/7688	33.56/32.67/32.5/
	0.04	59.87/0.01/0.001/0.001	27638/4448/3049/3038	160062/5894/1883/1865	10.51/9.365/9.308/9.308
Isidore Horse	0.02	0.32/0.008/0.003	10458/7634/7413	12690/7211/6800	54.9/54.52/543.8
	0.04	1.13/0.004	5750/2789	9766/2653	22.84/21.81
	0.062	0.8/0.004/0.0006/0.0005	4294/1283/1029/1016	9754/1385/833/814	5.229/4.087/4.083/4.083
Iphigenie	0.0135	72.09/0.1/0.006	33050/12915/10980	70947/13883/10045	22.21/20.59/20.03
	0.02	67.56/0.02/0.002	20084/5547/4023/	66908/5733/2562	9.101/7.929/7.555
Camel	0.02	0.077/0.008/0.007/0.007	10901/6784/6423/6369	10901/6784/6423/6369	12.77/12.44/12.35/12.33
	0.03	0.06/0.003/0.002/0.002	6388/4045/3883/3843	8828/3598/3265/3205	6.445/6.182/6.151/6.144
	0.07	0.03/0.0006	3704/737	10811/521	1.066/0.9093
Chinese Lion	0.02	0.31/0.02/0.005/0.003/0.003/0.003	25610/8702/6812/6511/6479/6469	46315/7478/4143/3733/3681/3665	6.346/6.205/6.153/6.148/6.148
	0.03	0.42/0.007/0.001/0.001	23149/4191/2872/2822	61620/4051/1535/1466	3.031/2.546/2.501/2.499
	0.04	0.66/0.004/0.001/0.0009	21797/2474/1663/1601	77365/2665/868/755	1.544/1.278/1.266/1.266
Elephant	0.02	0.05/0.003/0.002	5825/4623/4607	6524/4286/4260	7.886/7.908/7.907
	0.04	0.05/0.0007	2302/1356	3157/855	3.827/3.68
Rockerarm	0.02	1.46/0.007/0.003/0.002	11286/6408/6243/6232	18431/6757/6336/6320	14.74/13.70/13.58/13.58
	0.04	5.62/0.03/0.003/0.001	7436/2492/2252/2232	16440/2740/2175/2138	5.070/4.563/4.521/4.515
	0.06	8.46/0.006/0.0009/0.0008	6366/1416/1154/1143	18222/1783/1100/1084	2.555/1.944/1.884/1.884

Table 1: *Measurements of the optimization for the depicted examples. Computations were made on an Intel Core i7-4770 CPU.*

    COMPUTEINTEGRALCURVATURE(FCN,$\mathscr{M}$)
    COMPUTESYMMETRICARCS(FCN, $\mathscr{M}$)
end procedure

procedure SINGLEEDGEWEIGHTS(FCN = (V, V^*, E, A))
    e.weigth $\leftarrow$ I(e) · L(e) · Loop(e) · Sym(e)
end procedure

Arc Resampling RESAMPLEFCN describes the resampling process of the feature arcs. The samples are taken from the original curve segments, to which the current approximated arc refers.

procedure RESAMPLEFCN(FCN = $(V, V^*, E, A), r_{\min}, r_{\max}$)
    for $a \in A$ **do**
        define set of samples $S \leftarrow s_1, \ldots, s_n$
        Graph g ▷ build graph
        for $i \leftarrow 1, \ldots, n$ **do**
            for $j \leftarrow i, \ldots, n$ **do**
                if dist$(s_i, s_j) \in [r_{\min}, r_{\max}]$ **then**
                    $e \leftarrow g$.addEdge(s_i, s_j)
                    e.weight $\leftarrow$ integralEuclidianDist(e, a)
                end if
            end for
        end for
        ▷ get geometrically closest arc abstraction
        Path $p \leftarrow$shortestPath(s_1, s_n, g)
        if no path exists **then**
            ▷ return edge as an intermediate solution
            return $\{s_1, s_n\}$
        else
            return p
        end if
    end for
end procedure

Conflict Detection COMPUTECONFLICTS generates the edge and vertex conflict sets as discussed in the paper. Edge conflicts are computed by first checking whether the edges potentially conflict, and secondly if a valid triangle configuration exists.

procedure COMPUTECONFLICTS(FCN = $(V, V^*, E, A), r_{\min}$, $r_{\max}$)
    $C_e \leftarrow \emptyset$
    $C_v \leftarrow \emptyset$
    for $(e_0, e_1) \in E \times E$ **do**
        if $e_0 \neq e_1$ & dist$(e_0, e_1) < r_{\min}$ **then**
            if ! CHECKTRIANGLECONFIGURATIONS(e_0, e_1) **then**
                $C_e \leftarrow C_e \cup (e_0, e_1)$
            end if
        end if
    end for
    for $(v_0, v_1) \in V^* \times V^*$ **do**
        if dist$(v_0, v_1) < r_{\min}$ and $v_0 \neq v_1$ **then**
            $C_v \leftarrow C_v \cup (v_0, v_1)$
        end if
    end for

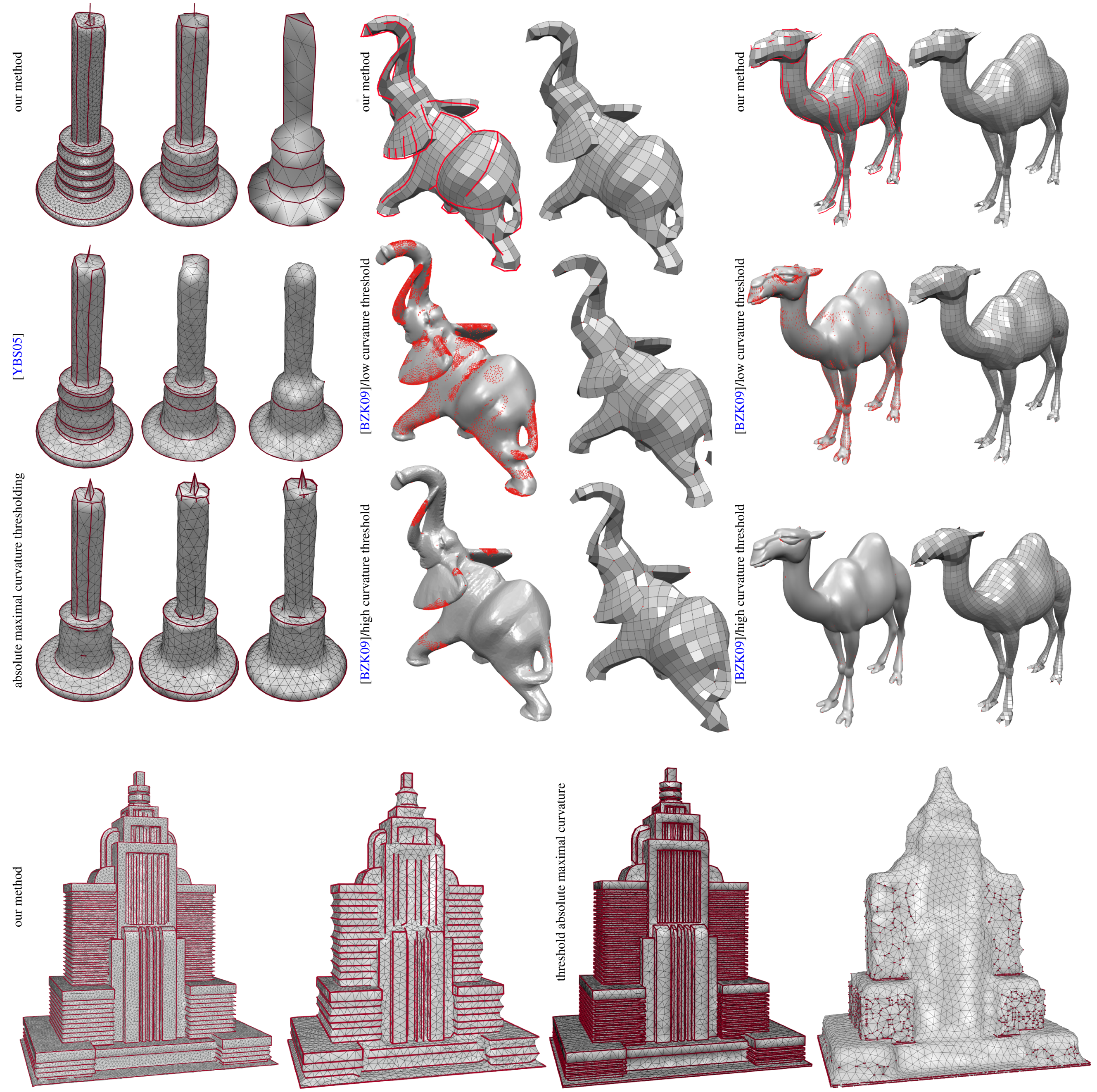

Figure 1: *Comparisons of our method to curvature thresholding/filtering methods. In case of the triangle meshes we threshold absolute curvature values. Also, for the candle we use curvature thresholding as described in [YBS05]. For the filtering and thresholding of curvatures for the quadmeshes we used [BZK09] with a filter-kernel radius of* $r_{\min}/2$. *Note that in all cases if we increase curvature thresholds such that all small-scale details are removed, also less prominent features are removed, which are important to convey the shape or to guarantee good element alignment.*

    return C_e, C_v
end procedure

CHECKTRIANGLECONFIGURATIONS tests possible adjacent and non-adjacent triangle configurations as discussed in the paper.

procedure CHECKTRIANGLECONFIGURATIONS(Edge e_0, Edge e_1)
    if ((adjacent(e_0, e_1) *or* e_0, e_1 connected by short edge) and

$\angle(e_0,e_1) \in [\alpha_{\min},\alpha_{\max}] \cup [2\alpha_{\min},2\alpha_{\max}] \cup [3\alpha_{\min},3\alpha_{\max}])$ **then**
  return true
 else if $||e_0|| \in [r_{\min}, r_{\max}]$ and $||e_1|| \in [r_{\min}, r_{\max}]$ **then**
  return true $\iff$ a non-adjacent configuration (cf. paper) applies
 else
  return false
 end if
end procedure

Resolve Conflicts: Optimization, Edge Removal, and Short Edge Collapse The procedure RESOLVECONFLICTS sets up the optimization model by translating discussed conflicts into constraints, as discribed in the paper. The function addVariable$(0,1)$ indicates that we add binary variables to the optimization model, the function addConstraint(c) adds the constraint c to the model. Then it maximizes the objective function, and deletes the edges that are set to 0 in the optimization, by either collapsing (only for short edges) or removing them completely. The function call optValue(b) returns whether the binary optimization variable b was set to 0 (remove) or 1 (preserve).

procedure RESOLVECONFLICTS(FCN $= (V, V^*, E, A), r_{\min}, C_e, C_v$)
 OptimizationModel m
 ObjectiveFunction $o \leftarrow 0.0$
 ▷ Variables that are set during optimization
 ▷ binary variables for edges
 for $i := 1,..,|E|$ **do**
  $b_i \leftarrow m$.addVariable$(0,1)$
 end for
 ▷ binary pseudo-variables for edges
 for $i = 1,..,|E|$ **do**
  $p_i \leftarrow m$.addVariable$(0,1)$
 end for
 ▷ binary variables for vertices
 for $i = 1,..,|V^*|$ **do**
  $c_i \leftarrow m$.addVariable$(0,1)$
 end for
 ▷ binary pseudo-variables for edge-pairs

 for $i = 0,..,|E|$ **do**
  for $j = 0,..,|E|$ **do**
   $a_{ij} \leftarrow m$.addVariable$(0,1)$
  end for
 end for
 ▷ set the objective function

 for $i = 0,..,|E|$ **do** ▷ single edge weights
  $o \leftarrow o + b_i \cdot e_i$.weight
 end for
 for $(e_i,e_j) \in E \times E$ **do** ▷ smoothness
  if (adjacent (e_i,e_j)) **then**
   $o \leftarrow o + \lambda_0 a_{ij} \cdot a_s(e_i,e_j)$
  end if
 end for
 for $(e_i,e_j) \in E \times E$ **do** ▷ orthogonality
  if dist$(e_i,e_j) < R$ **then**
   $o \leftarrow o + \lambda_1 a_{ij} \cdot a_{op}(e_i,e_j)$
  end if
 end for
 ▷ set constraints

 for $(e_i,e_j) \in C_e$ **do** ▷ edge conflicts
  m.addConstraint$(b_i + b_j \leq 1)$
 end for
 $V_{conflict} \leftarrow \emptyset$ ▷ vertex-conflicts
 for $(v_i,v_j) \in C_v$ **do**
  m.addConstraint$(c_i + c_j \leq 1)$
  $V_{conflict} \leftarrow V_{conflict} \cup v_i$
  $V_{conflict} \leftarrow V_{conflict} \cup v_j$
 end for
 ▷ Constraints to downgrade one of the conflicting vertices to regular vertices
 for $v_i \in V_{conflict}$ **do**
  if valence$(v_i) \geq 2$ **then**
   m.addConstraint($\sum_{e_j \in \text{one-ring}(v_i)} p_j \leq 2$)
  else
   m.addConstraint($\sum_{e_j \in \text{one-ring}(v_i)} p_j \leq 0$)
   for $e_j \in$ one-ring(v_i) **do**
    m.addConstraint$(p_j - b_j \leq c_i)$
    m.addConstraint$(b_j - p_j \leq c_i)$
   end for
  end if
 end for
 ▷ supress isolated short edges
 for $e_s = (v_i,v_j) \in E$ with $||e_s|| < r_{\min}$ **do**
  $C \leftarrow \sum_{e_k \in N(v_i) \setminus e_s} b_k + \sum_{e_k \in N(v_j) \setminus e_s} b_k \geq b_{e_s}$
  m.addConstraint(C)
 end for
 ▷ avoid generating small gaps in feature lines
 for $e_s = (v_i,v_j)$ with $||e_s|| < r_{\min}$ **do**
  if $(e_s,e_c) \in C_e$ or $(e_c,e_s) \in C_e$ **then**
   for $e_i (\neq e_s) \in N(v_i)$ **do**
    for $e_j (\neq e_s) \in N(v_j)$ **do**
     m.addConstraint$(b_c + b_i + b_j \leq 2)$
    end for
   end for
  end if
 end for
 m.maximize(o)
 ▷ remove all short edges that were set to 0 during optimization by collapsing them
 for $i = 1,..,|E|$ **do**
  if m.optValue$(e_i = (v_0,v_1)) == 0$ **then**
   if $||e_i|| < r_{\min}$ **then**
    collapse(v_0,v_1)
   end if
  end if
 end for
 ▷ check if the conflicts of each edge e_i are removed due to collapses and remove e_i otherwise
 for $i = 1,\ldots,|E|$ **do**
  if m.optValue$(e_i) == 0$ **then**
   for $(e_i,e) \in C_e$ **do**

```
                if !CHECKTRIANGLECONFIGURATIONS(e_i, e) then
                    FCN.deleteEdge(e_i)
                    break
                end if
            end for
        end if
    end for
end procedure
```

References

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. In *ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 77:1–77:10. doi:10.1145/1576246.1531383. 1, 3

[GO15] GUROBI OPTIMIZATION I.: Gurobi optimizer reference manual, 2015. URL: http://www.gurobi.com. 1

[YBS05] YOSHIZAWA S., BELYAEV A., SEIDEL H.-P.: Fast and robust detection of crest lines on meshes. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling* (New York, NY, USA, 2005), SPM '05, ACM, pp. 227–232. doi:10.1145/1060244.1060270. 1, 3