Supplemental Material - Adapting FCNs to a Prescribed Scale

Anne Gehre  Isak 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 candle). 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 $r_{\text{min}}/2$. 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.

1: procedure computeFCN($\text{FCN} = (V, V^*, E, A), r_{\text{min}}, r_{\text{max}}$)
2: computeSurfaceProperties($\text{FCN}$)
3: do
4: resampleFCN($\text{FCN}, r_{\text{min}}, r_{\text{max}}$)
5: $C_e, C_v \leftarrow$ computeConflicts($\text{FCN}$)
6: singleEdgeWeights($\text{FCN}$)
7: $C_e, C_v$
8: $\triangleright$ includes optimization, edge removal, and collapse
9: resolveConflicts($\text{FCN}, C_e, C_v$)
10: while $|C_v| \neq 0$ or $|C_e| \neq 0$
11: 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.

1: procedure computeSurfaceProperties($\text{FCN} = (V, V^*, E, A), \mathcal{\emptyset}$)
2: computeCurveLengths($\text{FCN}$)
3: computeLoops($\text{FCN}$)
Table 1: Measurements of the optimization for the depicted examples. Computations were made on an Intel Core i7-4770 CPU.

A. Gehre, I. Lim, L. Kobbelt / Supplemental Material - Adapting FCNs to a Prescribed Scale

Conflict Detection. \textsc{computeContacts} 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.

4: \textsc{computeIntegralCurvature}(FCN, \mathcal{M})
5: \textsc{computeSymmetricArccs}(FCN, \mathcal{M})
6: \text{end procedure}
1: \text{procedure singleEdgeWeights}(FCN = (V, V^*, E, A))
2: \text{e.weight} \leftarrow \text{(I(e) \cdot L(e) \cdot Loop(e) \cdot Sym(e))}
3: \text{end procedure}

Arc Resampling. \textsc{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.

1: \text{procedure resampleFCN}(FCN = (V, V^*, E, A), r_{min}, r_{max})
2: \text{for } a \in A \text{ do}
3: \text{define set of samples } S \leftarrow s_1, \ldots, s_n
4: \text{Graph } g \text{ \triangleright \ build graph}
5: \text{for } i = 1, \ldots, n \text{ do}
6: \text{for } j = i, \ldots, n \text{ do}
7: \text{if dist}(s_i, s_j) \in [r_{min}, r_{max}] \text{ then}
8: \text{e} \leftarrow g.addEdge(s_i, s_j)
9: \text{e.weight} \leftarrow \text{integralEuclidianDist}(e, a)
10: \text{end if}
11: \text{end for}
12: \text{end for}
13: \text{Path } p \leftarrow \text{shortestPath}(s_1, s_n, g)
14: \text{if no path exists then}
15: \text{return } \{s_1, s_n\}
16: \text{end if}
17: \text{else}
18: \text{return } p
19: \text{end if}
20: \text{end for}
21: \text{end procedure}
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.

16: return $C_e, C_v$
17: end procedure

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

1: procedure CHECKTRIANGLECONFIGURATIONS(Edge $e_0$, Edge $e_1$)
2: if ((adjacent($e_0, e_1$) or $e_0, e_1$ connected by short edge) and

© 2016 The Author(s)
Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.
\( \langle e_0, e_1 \rangle \in [\alpha_{\text{min}}, \alpha_{\text{max}}] \cup [2\alpha_{\text{min}}, 2\alpha_{\text{max}}] \cup [3\alpha_{\text{min}}, 3\alpha_{\text{max}}] \)

\[
\text{returns whether the binary optimization variable } C_e \text{ indicates that we add binary variables to the optimization model, which allows the objective function to be computed.}
\]

```python
for j = 0, ..., |E| do
    a_{ij} \leftarrow m.addVariable(0, 1)
end for
```

Resolve Conflicts: Optimization, Edge Removal, and Short Edge Collapse

The procedure RESOLVECONFLICTS sets up the optimization model by translating discussed conflicts into constraints, as described in the paper. The function addVariable(0,1) indicates that we add binary variables to the optimization model, which maximizes the objective function, and deletes the edges that are set to 0 in the optimization, either by collapsing (for short edges) or removing them completely. The function optValue returns whether the binary optimization variable \( b \) was set to 0 (remove) or 1 (preserve).

```python
for i := 1, ..., |E| do
    \( b_i \leftarrow m.addVariable(0, 1) \)
end for
```

Once the conflicts of each edge have been detected, the objective function is set to maximize the weight of the edge, which is done by setting the objective function to the weight of the edge if the conflicts are resolved.

```python
\text{ObjectiveFunction } a \leftarrow 0
```

```python
for i := 1, ..., |E| do
    \( p_i \leftarrow m.addVariable(0, 1) \)
end for
```

```python
for i := 1, ..., |V^*| do
    \( c_i \leftarrow m.addVariable(0, 1) \)
end for
```

```python
for i := 0, ..., |E| do
    for j = 0, ..., |E| do
        a_{ij} \leftarrow m.addVariable(0, 1)
    end for
end for
```

```python
a \leftarrow a + b_i \cdot e_i \cdot \text{weight}
```

```python
\text{smoothness } \quad a \leftarrow a + \lambda_3 a_{ij} \cdot a_{ij}
```

```python
\text{orthogonality } \quad \text{if dist}(e_i, e_j) < R \text{ then }
```

```python
\text{orthogonality } \quad \text{if dist}(e_i, e_j) < R \text{ then }
```

```python
\text{set the objective function }
```

```python
\text{set constraints }
```

```python
\text{set vertex-conflicts }
```

```python
\text{Constraints to downgrade one of the conflicting vertices to regular vertices }
```

```python
\text{suppress isolated short edges }
```

```python
\text{avoid generating small gaps in feature lines }
```

```python
\text{remove all short edges that were set to 0 during optimization by collapsing them }
```

```python
\text{check if the conflicts of each edge } e_i \text{ are removed due to collapses and remove } e_i \text{ otherwise }
```

```python
(\text{adjacent (}e_i, e_j)) \text{ then }
```

```python
if \text{dist}(e_i, e_j) < R \text{ then }
```

```python
\text{if dist}(e_i, e_j) < R \text{ then }
```

```python
\text{if dist}(e_i, e_j) < R \text{ then }
```

```python
\text{if dist}(e_i, e_j) < R \text{ then }
```

© 2016 The Author(s)
if \textit{CHECK_TRIANGLE_CONFIGURATIONS}(e_i, e) then
\begin{itemize}
  \item FCN.deleteEdge(e_i)
  \item break
\end{itemize}
end if
end for
end if
end for
end procedure

References

