Supplemental Material

The supplemental material is organized as follows:

In Section A we give pseudocode of the algorithms described in Section 3, and show additional plots B.

A. Algorithms

```plaintext
Algorithm 1: Pseudocode for initialization of the disocclusion map for disocclusions caused by motion.
in : dim // image dimensions
in : dMap // first layer depth map
in : vMap // first layer velocity map
out: dstMap // disocclusion map
1 // Initialize the disocclusion map
2 foreach (x,y) ∈ {0,...,dim.x} × {0,...,dim.y} do
3   currentD ← dMap(x,y)
4   rightD ← dMap(x + 1, y)
5   topD ← dMap(x, y + 1)
6   vDx ← vDxMap(x,y)
7   vDy ← vDyMap(x,y)
8 // Compute disocclusion extents at vertical edges
9   disocclusionX(l.r.t.b) ← (0,0,0,0)
10 if vDx.x > 0 then
11   if currentD > rightD then
12     disocclusionX.l ← vDx.x
13     if vDx.y > 0 then
14       disocclusionX.t ← vDx.y
15       else
16       disocclusionX.b ← −vDx.y
17     else
18       disocclusionX.l ← −vDx.x
19     if vDx.y < 0 then
20       disocclusionX.b ← −vDx.y
21     else
22       disocclusionX.b ← vDx.y
23   end
24 end
25 // Compute disocclusion extents at horizontal edges
26 disocclusionY(l.r.t.b) ← (0,0,0,0)
27 if vDy.Y > 0 then
28   if currentD > topD then
29     disocclusionY.l ← vDy.Y
30     if vDy.X > 0 then
31       disocclusionY.r ← vDy.X
32       else
33       disocclusionY.r ← −vDy.X
34     else
35       disocclusionY.b ← −vDy.Y
36     if vDy.X < 0 then
37       disocclusionY.r ← −vDy.X
38     else
39       disocclusionY.r ← vDy.X
40   end
41 end
42
Algorithm 2: Pseudocode for spreading of disocclusions caused by motion. Input is a disocclusion map which is initialized with Algorithm 1.
in : dim // image dimensions
in : dstMap // disocclusion map
in : vMap // first layer velocity map
out: dstMap // disocclusion map
1 // Spread disocclusions
2 numLevels ← |log₂(max(dim.x,dim.y))|
3 level ← 1
4 m ← 1
5 srcMap ← dstMap
6 dstMap ← empty
7 while level ≤ numLevels do
8   // Relevant extents and offsets for horizontal and vertical neighbors
9   hvNeighbors ← {(r,(−m,0)),(l,(m,0)),(r,(0,−m)),(b,(0,m))}
10  // Relevant extents and offsets for diagonal neighbors
11  diagNeighbors ← {(rt, (−m,−m)), (lt, (m,−m)), (rb, (−m,m)), (lb, (m,m))}
12 foreach p = (x,y) ∈ {0,...,dim.x} × {0,...,dim.y} do
13   spread ← srcMap(p)
14   // Spread disocclusion from horizontal and vertical neighbors
15   foreach (e,o) ∈ hvNeighbors do
16     spread.e ← max(spread.e, srcMap(p + o).e − m)
17   end
18   // Spread disocclusion from diagonal neighbors
19   foreach (e,o) ∈ diagNeighbors do
20     spread.e ← max(spread.e, srcMap(p + o).e − (m,m))
21   end
22   dstMap(p) ← spread
23 end
24 level ← level + 1
25 m ← m - 2
26 swap(srcMap,dstMap)
27 end
28 end
29 dstMap ← srcMap
```

© 2016 The Author(s)

Computer Graphics Forum © 2016 The Eurographics Association and John Wiley & Sons Ltd.
B. Additional graphs and figures

Figure 15: Comparison of using a two depth layer (a) without minimum z-separation, two depth layers with correct z-separation (b) and infinite depth (c) against Blender reference (d) for multiple disocclusions. While there are still some artifact remaining when using the second layer, they are hardly noticeable during animation.

Figure 16: The t-fragments for $t = 0.25$ (see Figure 6 in the paper) visualized according to their world-space distance from the viewing ray over time.

Figure 17: Comparison of our approach (a) against Blender reference (b) for very large motion vectors.

Figure 18: Number of opaque and transparent fragments generated using the disocclusion map for early fragment culling.