# Supplemental material

Here we take a closer look of how to derive the different antiderivatives, needed by the ATM algorithm, using the symbolic solver of Mathematica (App. A) and show some example antiderivatives. In App. B we illustrate how some of the ATM algorithm details are implemented in code.

## APPENDIX A
## SYMBOLIC DERIVATION OF ANTIDERIVATIVES

The following Mathematica pseudo code illustrates the derivation of the functions, the horizontal-multi-cell antiderivative per-cell functions $\widetilde{H}^{ij}_{t_n,p_n}$ (in code **Hm<ij>**), and the per-cell difference functions $D^{ij}_{t_n,p_n}$ (in code **D<ij>**) as introduced in Sec. IV-A, that are used to evaluate $H_{t_n,p_n}$ in the ATM code shown in App. B.

```
(* We have vertical multi-cell antiderivative Vij plot: Fig. 11b. *)
V00[k_,l_] =  1/2 (k + 1) (l + 1)^2
V10[k_,l_] = -1/2 (k - 1) (l + 1)^2
V01[k_,l_] = -1/2 (k + 1) (l^2 - 2l - 1)
V11[k_,l_] =  1/2 (k - 1) (l^2 - 2l - 1)

g[k_] = t k + p  (* line function of quad-edge: k->l *)

(* Obtain Per-cell slanted antiderivative (Hpij, in paper H~~^ij) *)
(* which starts at zero at each cell's west border, plot: Fig. 11c.
     *)
Hi00[k_,t_,p_] = Integrate[V00[k, g[k]], k]
Hi10[k_,t_,p_] = Integrate[V10[k, g[k]], k]
Hi01[k_,t_,p_] = Integrate[V01[k, g[k]], k]
Hi11[k_,t_,p_] = Integrate[V11[k, g[k]], k]
Hp00[k_,t_,p_] = Hi00[k,t,p] - Hi00[-1,t,p]
Hp10[k_,t_,p_] = Hi10[k,t,p] - Hi10[ 0,t,p]
Hp01[k_,t_,p_] = Hi01[k,t,p] - Hi01[-1,t,p]
Hp11[k_,t_,p_] = Hi11[k,t,p] - Hi11[ 0,t,p]
(* Next only needed for Dij: *)
Hp02[k_,t_,p_] = Hi02[k,t,p] - Hi02[-1,t,p]
Hp12[k_,t_,p_] = Hi12[k,t,p] - Hi12[ 0,t,p]

(* Obtain horizontal-Multi-cell antiderivative (Hm, in paper H~~^ij) *)
(* See plot: Fig. 11d. *)
Hm00[k_,t_,p_] = Hp00[k,t,p] + 0
Hm10[k_,t_,p_] = Hp10[k,t,p] + Hp00[0,t,p]
Hm01[k_,t_,p_] = Hp01[k,t,p] + 0
Hm11[k_,t_,p_] = Hp11[k,t,p] + Hp01[0,t,p]
Hm[k_,t_,p_] := Piecewise[{
  {Hm00[k,t,p], -1 <= k < 0 && -1 <= g[k] < 0},
  {Hm10[k,t,p],  0 <= k < 1 && -1 <= g[k] < 0},
  {Hm01[k,t,p], -1 <= k < 0 &&  0 <= g[k] < 1},
  {Hm11[k,t,p],  0 <= k < 1 &&  0 <= g[k] < 1}
  }]
Hm02[k_,t_,p_] = Hp02[k,t,p] + 0
Hm12[k_,t_,p_] = Hp12[k,t,p] + Hp02[0,t,p]

gInv[t_,p_,l_] := (l - p) / t  (* inverse line function: l->k *)

(* Obtain vertical Difference function Dij for hor. cell borders *)
D00[k_,t_,p_] = Hm00[gInv[t,p,-1], t, p] - 0  (* border l = -1 *)
D10[k_,t_,p_] = Hm10[gInv[t,p,-1], t, p] - 0
D01[k_,t_,p_] = Hm01[gInv[t,p,0], t, p] - Hm00[gInv[t,p,0], t, p]
D11[k_,t_,p_] = Hm11[gInv[t,p,0], t, p] - Hm10[gInv[t,p,0], t, p]
(* Contradictory to south of D*0 the north of D*2 is not zero: *)
D02[k_,t_,p_] = Hm02[gInv[t,p,1], t, p] - Hm01[gInv[t,p,1], t, p]
D12[k_,t_,p_] = Hm12[gInv[t,p,1], t, p] - Hm11[gInv[t,p,1], t, p]

(* Only accumulate the Dij-s to Hm when line (t,p) crosses cell-ij
   south border, and is above/below that border for pos/neg t
   (and k is within prefilter support). These area-bounded functions
   enable straightforward accumulation. *)
st = Sign[t]
D00area[k_,t_,p_] = Piecewise[{ { D00[k,t,p],
  0>(-p-1)st && 0<(-p-1+t)st  && (*crosses cell's border*)
  0>(-g[k]-1)st && k<1 } }] (* above border for t>0 *)
D10area[k_,t_,p_] = Piecewise[{ { D10[k,t,p],
  0<(-p-1)st                  &&  0>(-g[k]-1)st && k<1 } }]
D01area[k_,t_,p_] = Piecewise[{ { D01[k,t,p],
  0>(-p+0)st && 0<(-p+0+t)st  &&  0>(-g[k]+0) st && k<1 } }]
D11area[k_,t_,p_] = Piecewise[{ { D11[k,t,p],
  0<(-p+0)st                  &&  0>(-g[k]+0)st && k<1 } }]
D02area[k_,t_,p_] = Piecewise[{ { D02[k,t,p],
  0>(-p+1)st && 0<(-p+1+t)st  &&  0>(-g[k]+1) st && k<1 } }]
D12area[k_,t_,p_] = Piecewise[{ { D12[k,t,p],
  0<(-p+1)st                  &&  0>(-g[k]+1)st && k<1 } }]
Dsum[k_,t_,p_] = D00area[k,t,p] + D10area[k,t,p] + D01area[k,t,p]
  + D11area[k,t,p] + D02area[k,t,p] + D12area[k,t,p]

(* Obtain H for verification and to plot it, see Fig. 11e *)
H[k_,t_,p_] := Hm[k,t,p] - Dsum[k,t,p] st
```

Listing 1. Mathematica pseudo code; deriving **Hm<ij>** and **D<ij>** and showing how to use them to construct **H** for the example 2x2 (Tent) prefilter.

To illustrate the complexity, the derived cell functions **Hm<ij>** and **D<ij>** for the Tent prefilter are shown in Tab. II and Tab. III, respectively. Since we apply clamping of the quad-edge against the prefilter support (Sec. V-A), we only need to define cell functions in and on the borders of the support. For the difference functions this means 6 functions: 4 for the south borders of each cell and 4 for the north borders (with two north- and south-borders the same). Note, the two **Hm02** and **Hm12** cell functions above the support; these are only needed for deriving the prefilter north-border difference functions.

| i j | **Hm<ij>** |
|---|---|
| 00 | $\frac{1}{24}(k+1)^2\left(t^2\left(3k^2-2k+1\right)+4t(2k-1)(p+1)+6(p+1)^2\right)$ |
| 10 | $\frac{1}{24}\left(t^2\left(-3k^4+4k^3+1\right)-4t(k-1)^2(2k+1)(p+1)$ $-6\left(k^2-2k-1\right)(p+1)^2\right)$ |
| 01 | $-\frac{1}{24}(k+1)^2\left(t^2\left(3k^2-2k+1\right)+4t(2k-1)(p-1)$ $+6\left(p^2-2p-1\right)\right)$ |
| 11 | $\frac{1}{24}\left(t^2\left(3k^4-4k^3-1\right)+4t(k-1)^2(2k+1)(p-1)$ $+6\left(k^2-2k-1\right)\left(p^2-2p-1\right)\right)$ |
| | Next needed for deriving **D02** and **D12** |
| 02 | $\frac{1}{2}k^2+k+\frac{1}{2}$ |
| 12 | $-\frac{1}{2}k^2+k+\frac{1}{2}$ |

TABLE II
MULTI-CELL ANTIDERIVATIVES **HM<IJ>** DERIVED FOR THE TENT FILTER.

| i j | **D<ij>** |
|---|---|
| 00 | $\frac{1}{24t^2}(-t+p+1)^4$ |
| 10 | $\frac{1}{24t^2}t^4-4t^3(p+1)+6t^2(p+1)^2-4t(p+1)^3-(p+1)^4$ |
| 01 | $-\frac{1}{12t^2}(t-p)^4$ |
| 11 | $\frac{1}{12t^2}-t^4+4t^3p-6t^2p^2+4tp^3+p^4$ |
| 02 | $\frac{1}{24t^2}(t-p+1)^4$ |
| 12 | $\frac{1}{24t^2}t^4-4t^3(p-1)+6t^2(p-1)^2-4t(p-1)^3-(p-1)^4$ |

TABLE III
DIFFERENCE FUNCTIONS **D<IJ>** DERIVED FOR THE TENT FILTER.

To indicate the higher complexity with cubic filters, tables IV and V show some of the cell functions derived for a strong up-sharpening Cubic (Mitchell [34] with B=0, C=1) filter.

| i j | **Hm<ij>** |
|---|---|
| 00 | $(1/10080)(2+x)^3(3t^4(-32+48x-48x^2+40x^3-30x^4+105x^5)+$ $168t(-2+3x+12x^2)(1+y0)(2+y0)^2+70(2+3x)(2+y0)^3(2+$ $3y0)+16t^3(4-6x+6x^2-5x^3+30x^4)(5+3y0)+840t^2x^3(8+$ $10y0+3y0^2))$ |
| 10 | $(1/10080)(-70(-6-12k+8k^3+3k^4)(2+p)^3(2+3p)-168(5-$ $30k^2+30k^4+12k^5)(1+p)(2+p)^2t-168(2-10k^3+12k^5+$ $5k^6)(8+10p+3p^2)t^2-8(-77-105k^4+140k^6+60k^7)(5+3p)t^3-$ $9(90-56k^5+80k^7+35k^8)t^4)$ |
| ... | |

TABLE IV
ANTIDERIVITIVES **HM<IJ>** DERIVED FOR THE CUBICSTRONG FILTER.

| i j | **D<ij>** |
|---|---|
| 00 | $(1/10080t^4)((2+p-2t)^6(9p^2-12p(-1+t)-4(3+8t+3t^2)))$ |
| 10 | $(1/10080t^4)(-3(2+p)^7(-2+3p)+8(2+p)^6(-1+3p)t-672(1+p)(2+p)^2t^5+672(8+10p+3p^2)t^6-736(5+3p)t^7+864t^8+6(2+p-t)^6(-4+3p^2-8t-9t^2+p(4+6t)))$ |
| ... | |

TABLE V
DIFFERENCE FUNCTIONS **D<IJ>** DERIVED FOR CUBICSTRONG FILTER.

We provide a GPU-based ATM demonstration application for Windows which includes source code of the shaders with the derived polynomial functions for all implemented filters.

## APPENDIX B
## CODE DETAILS

In this section we show pseudo code to evaluate $H_{t,p}(k)$ of a quad-edge end point, as used in (8), using an example prefilter of 2x2 cells (e.g a Tent). We follow the variable naming introduced in Supplemental App. A. As described at the end of Sec. IV and shown in the bottom of Lis. 1, $H_{t,p}(k)$ can be evaluated using the horizontal multi-cell antiderivative **Hm<ij>** and the sum of a set of intersected cell borders' difference functions **D<ij>** that are intersected by the line of integration. Lis. 2 shows the code for evaluating **Hm**. The variables **k** and **gk** (following Eq. 6) represent the quad edge end-point after clamping (Sec. V-A) and after handling near-vertical quad edges (Sec. V-B).

```
Hm(k, t, p, gk) // for 2x2 cells prefilter
{
  if (k >= 0) // 0 <= k < 1: in cell10 or cell11
  {
      if (gk >=  0)   Hm = Hm11(k, t, p) else
    /*if (gk >= -1)*/ Hm = Hm10(k, t, p)
  }
  else // -1 <= k < 0: in cell00 or cell01
  {
      if (gk >=  0)   Hm = Hm01(k, t, p) else
    /*if (gk >= -1)*/ Hm = Hm00(k, t, p)
  }
  return Hm
}
```

Listing 2. Pseudo code for evaluating **Hm** of a 2x2-cell prefilter

Lis. 3 shows the code for evaluating the sum (**Dsum**) of the set of **D<ij>**-s that are on the integration line.

```
Dsum(t, p, gk) // for 2x2 cells prefilter
{
  if (t >= 0) st = 1 else st = -1
  Dsum = 0
  // top row D*2, only for negative tangent,
  if (t < 0 && gk <= 1) // below horizontal line: g(k) <= +1
  {
    if ( p <= 1  &&  p > 1 + t )    Dsum += D02(t, p)
    else if ( p > 1 )               Dsum += D12(t, p)
  }
  // middle row D*1, if g(k) is above 0-line when t>=0 or
  if ( (t>=0 && gk>=0) || (t<0 && gk<0)) // if below 0-line when t<0
  {
    // for pos. t: p >= 0 && p < t, for neg. t: p <= 0 && p > t
    // above conditions combine to:
    if ((0 >= (-p + 0)*st) && (0 < (-p + 0 + t)*st))
                                    Dsum += D01(t, p)
    // for pos. t: p < 0, for neg. t: p > 0
    else if (0 < (-p + 0)*st)       Dsum += D11(t, p)
  }
  // bottom row D*0, only for positive tangent,
  if ( t > 0 && gk >= -1) // above hor. line: g(k) >= -1
  {
    if ( p >= -1  &&  p < -1 + t )  Dsum += D00(t, p)
    else if ( p < -1 )              Dsum += D10(t, p)
  }
  return Dsum
}
```

Listing 3. Pseudo code for computing the sum of the set of **D<ij>** that are on the integration line.

Finally, the sought-after antiderivative $H_{t,p}(k)$ is evaluated with: **Hm**`(k,t,p,gk) -` **Dsum**`(t,p,gk) * sign(t).`