Algorithm 1 Creates $H \in \mathbb{R}^{(3m)\times(3n)}$

procedure HSTA_KERNEL
  $j :=$ blockIdx.x; $i :=$ blockIdx.y; $x :=$ threadIdx.x; $y :=$ threadIdx.y
  $l :=$ laneId; $w :=$ warpId; $numWarps = \lceil g^2/32 \rceil$
  Initialize *Hbuffer* in shared memory with 0
  Hlocal $\leftarrow$ GenerateMatrixHsta(i, j, x, y) ▷ Return 3×3 matrix
  `shfl_down`(*Hlocal*) ▷ Reduces matrices in the same warp
  if $l = 0$ **then**
    Hbuffer[warpId] = *Hlocal*
  if $x < 3$ and $y < 3$ **then**
    $Helem[x][y] \leftarrow$ `thrust::reduce`($Hbuffer[x][y][0], Hbuffer[x][y][numWarps]$)
    $H[i][j] \leftarrow Helem$ ▷ $H[i][j]$ is a 3×3 matrix
procedure RIGID_KERNEL($H, Hdiag$)
  $t :=$ blockDim.$x \times$ blockIdx.$x +$ threadIdx.x
  if $t < m$ **then**
    for $k := 1, n$ **do**
      $Hdiag[t] \leftarrow Hdiag[t] - H[t][k]$
procedure HSTA_ASSEMBLY
  Move data to GPU memory
  Allocate $H \in \mathbb{R}^{(3m)\times(3n)}$ in GPU memory
  Allocate $Hdiag \in \mathbb{R}^{m\times3\times3}$ in GPU memory and initialize with 0
  Allocate *Hbuffer* in GPU shared memory, buffer of matrices 3×3 of size $\lceil g^2/32 \rceil$
  Run `Hsta_kernel` with $m \times n$ blocks and $g \times g$ threads. Await for return
  Run `Rigid_kernel` with 128 threads and $\lceil m/128 \rceil$ blocks. Await for return
  Keep *Hdiag* in GPU memory and free H from GPU memory.