Appendix A: Appendix - Code Listings

```
float smoothPixel(Si, Sj, S, R, weights) {
    // compute the weight sum of pixels nearby
    // this code doesn't handle edge conditions
    // and assumes sum of weights[i,j] = 1.0
    float sum = 0.0;
    for (int j=0; j<R; j++)
        for (int i=0; i<R; i++)
           sum += weights[i,j]*S[Si+i,Sj+j]
    return sum; }
```

Listing 1: *Stencil computation in 2D: performs sum of product of nearby pixels with weights.*

```
// coarse-grained parallelism
#pragma omp parallel for
for (j=0;j<height;j++)
   for (i=0;i<width;i++)
       destImage[i,j] = smoothPixel(i, j, S, R,
    weights)

// fine-grained parallelism
#pragma omp parallel for
for (p=0;p<width*height;p++)
{
    int i, j;
    computeLocalIJ(p, width, height, &i, &j);
    destImage[i,j] = smoothPixel(i, j, S, R,
    weights)
}
```

Listing 2: *OpenMP parallel computation in 2D. In the coarse-grained approach, each thread gets a scanline to process. In the fine-grained approach, threads work on pixels, assigned as determined by OpenMP runtime distribution rules.*

```
template <typename InputArrayType, typename
    InputArrayType, typename OutputArrayType>
 VTKM_EXEC void operator()(const InputArrayType
    & inputImg, const InputArrayType & weights,
    OutputArrayType & outputImg,
 vtkm::Id indx) const
 {
// input: vtkm::Float64 inputImg, vtkm::Float64
    weights, vtkm::Id indx
// output: vtkm::Float64 outputImg
// assume private R: stencil size, iSize, jSize:
    size of 2D image
// compute (i,j) indices from 1D indx

int jVal = indx / iSize; // which row
int iVal = indx % iSize; // which column

float sum = 0.0;
for(int j=0; j<R; j++)
   for(int 0; i<R; i++)
      sum += weights[i,j]*inputImg[iVal+i,jVal+j
    ];
outputImg.Set(indx, sum);
```

Listing 3: *VTKm-FM algorithm in 2D: In VTK-m, execution environment iterates over the field and invokes the smoothPixel worklet in parallel.*

```
// input: BoundaryState &boundary, vtkm::Float64
    weights, InputFieldPortalType inputField
auto minIndices = boundary.MinNeighborIndices (
    this -> stencilRadius );
auto maxIndices = boundary.MaxNeighborIndices (
    this -> stencilRadius );

float sum=0.0;
for(vtkm::IdComponent j=minIndices [1]; j<=
    maxIndices[1]; ++j)
    for (vtkm::IdComponent i=minIndices[0]; i<=
    maxIndices[0]; ++i)
       sum += inputField.Get(i, j) * this->
    weights[i,j];
return static_cast <T> (sum);
```

Listing 4: *VTK-m-PN algorithm: similar to the VTK-m-FM algorithm, but without the global indexing computation as VTKm provides a view only to the local mesh/image neighborhood*