LA-UR-23-29392

Approved for public release; distribution is unlimited.

Title:	Data Remapping Between One-Dimensional Meshes
Author(s):	Ray, Navamita Mason, Carter Bayo Shevitz, Daniel Wolf
Intended for:	Documenting work done during internship.
Issued:	2023-08-15

Data Remapping Between One-Dimensional Meshes

Carter Mason, Daniel Shevitz, Navamita Ray

May-Aug 2023

Abstract

In this report, we describe two approaches to the problem of remapping data from a source mesh (on which data is available) onto a target mesh. We consider two separate methods to solve the problem: Point-wise and Conservative remap, and determine why one is more advantageous when considering different physics applications and elements. We utilize C++ functionalities to derive our findings along with the C++ "Chronos" library for timing measurements of our studies.

1 Introduction

In this report, we present two algorithms for performing mesh to mesh data remapping. The problem of mesh to mesh data remapping arises in many applications. In particular, in Arbitrary Lagrangian Eulerian (ALE) methods based hydrodynamic applications, a mesh moves along flow as a Lagrangian motion, till the mesh becomes skewed, at which point, a new mesh with better element shape is constructed and the data from the Lagrangian mesh is remapped to the new target mesh to continue with the simulation. Multi-physics applications also need similar data transfer where different physics interact with each other through shared domain boundaries.

The problem of remapping involves two meshes, where a discrete set of field values (data) is available on one of the mesh (source mesh). This discrete field is usually obtained from the numerical simulation and represents physical properties of the system of study such as density, temperature, velocity, etc. As a result, this discrete field is only defined at specific coordinates in the source mesh. Since the source and target mesh can be different, we need to find an approximation of the same on the target mesh using the data available on the source mesh.

In section 2, we describe the concepts of meshes and field data that are required by the remapping algorithms. Next, in section 3, we present a point-wise data remapping algorithm. We provide numerical results as well as profiling results for parts of the algorithm. In section 4, we present a conservative data remapping algorithm, and provide numerical results.

2 Meshes and Fields

2.1 Meshes

A mesh is a discretized approximation of a shape in any-dimension. For our purposes, we will work with one-dimensional domain, specifically, a bounded interval [0, 1]. This interval is partitioned into smaller intervals to obtain an one-dimensional mesh. Figure 1 shows the two main components that the mesh consists of :

- **Nodes**: Point coordinates that define the sub-intervals of the line
- **Cells**: An edge connecting two consecutive points, or a sub-interval of the line.

2.2 Remapping Data Between Two Meshes

As briefly described before, the problem of remapping involves transfer of data between two meshes. The mesh on which data is available is called a source mesh, and the mesh onto which the data has to be transferred is called the target mesh. There are two general locations where a physical field of interest to the simulation can be defined, at nodes or at cell-centers. In this report we consider two algorithms

- Point-wise remap: Node-to-node data transfer
- Conservative remap: Cell-center to cell-center data transfer while preserving area under the curve.

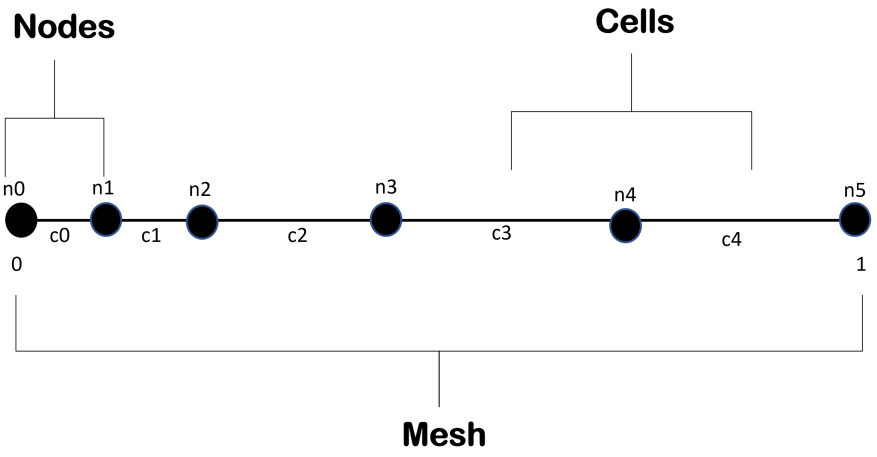


Figure 1: Mesh Illustration with labeled cells and nodes

3 Point-wise Remapping

We first consider the case, where the field is defined at the nodes of the source mesh, and we need to remap the field onto the nodes of the target mesh. In order to do this, we first obtain an approximation over the source mesh by using piece-wise linear interpolation, and then use this approximation to obtain field values at the target nodes.

3.1 Piece-wise Linear Approximation over Source Mesh

Given a source mesh with field values at its nodes, we can approximate the underlying function (from which the field values are obtained) by constructing a linear approximation over each source cell. Essentially, we create a series of linear segments connecting the nodes, giving us a piece-wise linear approximation of the field over the source mesh.

Figure 2 illustrates this process, where the discrete set of data on the source nodes is the input, and after interpolation, we obtain a piece-wise approximation to the field satisfying the condition that the approximation passes through each input data point.

The two pieces of information over each source cell that needs to be obtained are:

- Slope: The gradient of the line connecting two source nodes
- Y-intercept: The intercept of the line connecting two source nodes

We can use the node values to generate the slopes and intercepts of the linear approximation over each source cell. Given two points $\{x_1, y_1\}$ and $\{x_2, y_2\}$, we can find the slope of the line $y = mx + b$ joining them using the formula:

$$m = \frac{rise}{run} = \frac{y_2 - y_1}{x_2 - x_1} \tag{1}$$

We can then use the slope and one of the y values from the source points of the cell to find the y-intercept using:

$$b = y_1 - mx_1 = y_2 - mx_2 \tag{2}$$

To obtain an approximation at the target node, we now need to find which source cell the target node resides in, and once we find that source cell, we can use its corresponding linear approximation to obtain a field value at the target node. This is where search comes in.

3.2 Search

We need to obtain the source cell which the target node is in, so that we can use its gradient. We do this by checking which two source node values the target node value lies between. There are two methods we used for this: linear search and binary search.

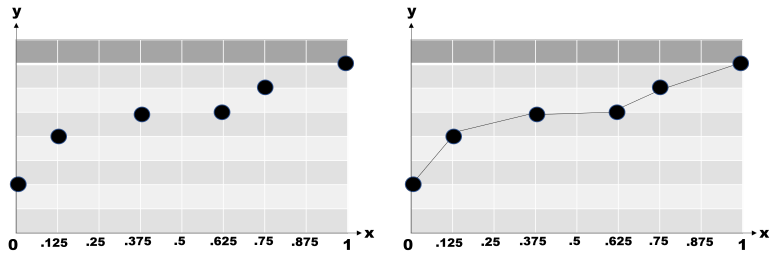


Figure 2: Piece-wise approximation before and after interpolation

3.2.1 Linear Search

Linear search is the process of sequentially checking a list of values for a specified point. We start at the first two nodes and check if the target value is on or between them. If it is, we have found our needed source cell, if it not, we check the second and the third node, and we repeat the process until we find our designated source node. If we do not find a desired cell after the last source node, then our target value is out of scope. Figure 3 illustrates the linear search process. Algorithm 1 provides the psuedo-code for the linear search algorithm.

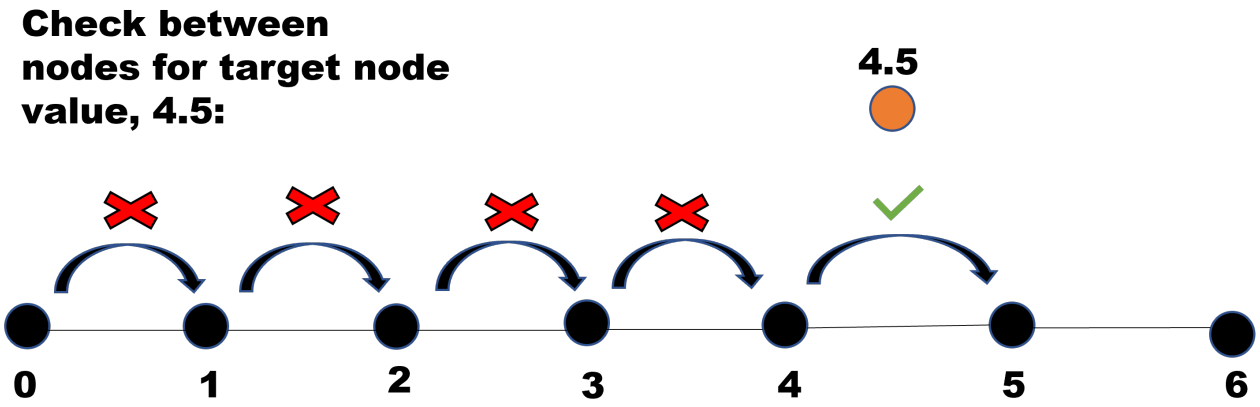


Figure 3: Linear Search

Algorithm 1 Linear Search Algorithm

```
Xt = x-target value
Xs = x-source value (held in an array)
n = number of source cells
initialize i to 0
for i to n do
    if (Xt ≥ Xs[i]) and (Xt ≤ Xs[i+1]) then
        return i;
    end if
end for
return -1;                                  ▷ target was not found in any source cell
```

3.2.2 Binary Search

Binary search is the "divide and conquer approach" where we go to the mid cell and check if our target node lies within. If it does, we have found our needed source cell, if our target node is lower than both, we disregard the upper half and repeat the process, and if our target node is higher, we disregard the lower half until and repeat. This is a much faster approach, especially when there is more data to be searched. For a search containing 6 nodes, binary search was able to find the target in only two searches (Figure 4), while linear search took 5 searches(Figure 3). Algorithm 2 provides the pseudo-code for the binary search algorithm.

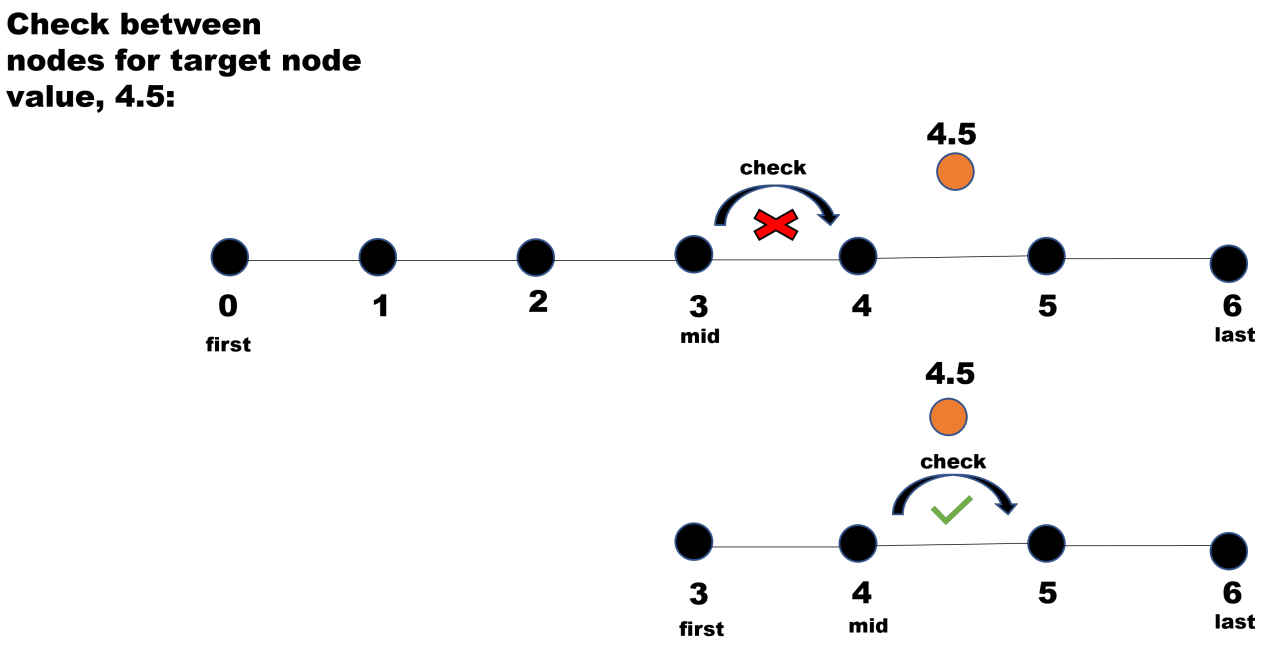


Figure 4: Binary Search ex.

3.3 Interpolation on Target Mesh

Once we have found the source cell that the target cell lies in, we know the slope and intercept of the linear approximation over the source cell to use for interpolation. Given a target point x_t, we can find an approximate y_t by using :

$$y_t = mx_t + b \tag{3}$$

where m and b are the source cell slope and intercept that the target node lies in. This gives us our interpolated field value at the given target node. Once all of the target node fields have been interpolated, this completes the process of remap.

3.4 Numerical Results

3.4.1 Verification Using Linear Function

For our problem of interpolating the unknown field values of a target mesh, we were given a source mesh with known field values and a target mesh with unknown field values. We first perform a verification test with an analytic function over the source mesh.

The meshes are generated by dividing the interval [0, 1] into equal length intervals. We choose a source mesh with 10 cells and a target mesh with 7 cells. Table 1 shows the nodes for the source and target meshes. We use the following linear function to obtain the discrete set of field values on the source mesh:

$$y = 3x + 2 \tag{4}$$

Figures 5a and 5b show the source field points before and after linear reconstruction. We now have the slopes and intercepts of each line approximation over cells of the source mesh. Using this function approximation, we can interpolate the field value at the target points. Figure 6 shows the target field data after the points are interpolated using the source mesh. We can see the the target points lie on the line that we chose over the source mesh. This completes our verification.

3.4.2 Interpolation with Quadratic Data

Now that we have verified that our code implementation works, we performed interpolation using a quadratic equation:

$$y = -4(x - 1/2)^2 + 1 \tag{5}$$

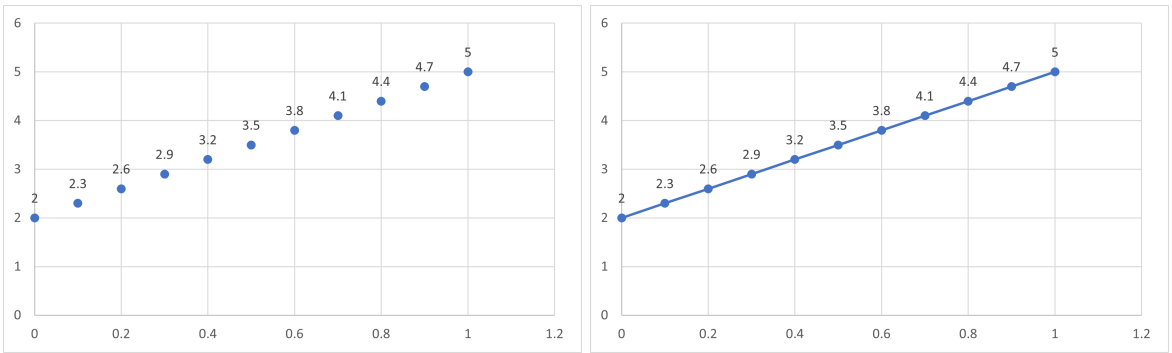
The source mesh has 10 cells and the target mesh has 9 cells. Table 2 displays the nodes for the target and source mesh. We can see how our interpolated target points (Figure 8) fall on the line approximations over the source cells (Figure 7).

Algorithm 2 Binary Search Algorithm

```
Xt = x-target value
Xs = x-source value (held in an array)
n = number of source cells
initialize i to 0
initialize first to 0
initialize last to n
for i to n do
    initialize mid to first + ((last − first)/2)
    if (Xt ≥ Xs[mid]) and (Xt ≤ Xs[mid+1]) then
        return mid;
    end if
    if (Xt > Xs[mid+1]) then
        first = mid+1
        mid = (last-first)/2
    end if
    if (Xt < Xs[mid]) then
        last = mid;
        mid = (last-first)/2;
    end if
end for
return -1;                                  ▷ target was not found in any source cell
```

Source Nodes	Target Nodes
0.00	0.00
0.10	0.14
0.20	0.29
0.30	0.43
0.40	0.57
0.50	0.71
0.60	0.86
0.70	1.0
0.80	
0.90	
1.00	

Table 1: Source and Target Mesh for Verification test



(a) Source Field

(b) Piece-wise Linear Interpolation

Figure 5: Input source field and the piece-wise linear approximation to the field over the source.

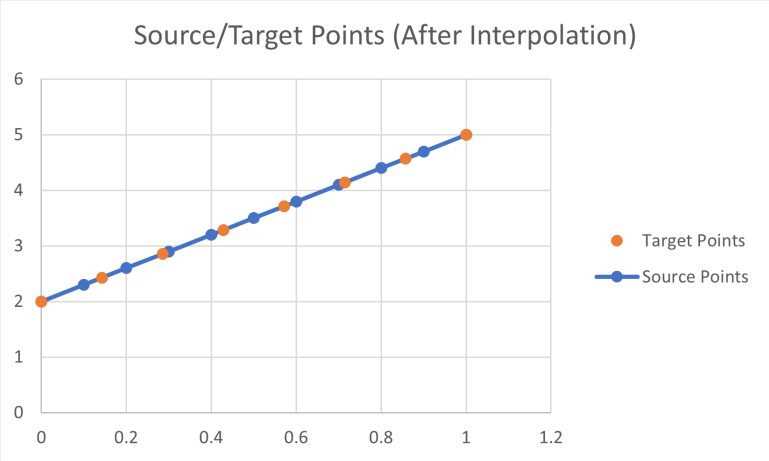


Figure 6: Target data after interpolation

Source Nodes	Target Nodes
0.00	0.05
0.10	0.15
0.20	0.25
0.30	0.35
0.40	0.45
0.50	0.55
0.60	0.65
0.70	0.75
0.80	0.85
0.90	0.95
1.00	

Table 2: Source and Target Mesh for Quadratic Test

3.4.3 Profiling Results

Profiling is the timing of different components of the program. We profiled both the linear and binary searches using the c++ "Chronos" library, to determine which search method yields faster results. We used the "chrono::high resolution clock::now()" function to register the start and end times for our searches. We used the function to put start and ending timing blocks around the loops for the searches, and then obtained the average times. We fixed the size of the target mesh (100) and vary the source mesh size from one cell to a million cell mesh.

$$t_{avg} = (t_{end} - t_{start})/n_t \tag{6}$$

This formula finds the average time by subtracting the start time from the end time and dividing that number by the number of target cells. Figure 10 shows the performance of the linear and binary search from one to one million source cells. Figure 9 shows a close up of the performance up-to 10,000 cells. The plots suggest that the binary search was exponentially faster over time with increasing number of cells and the time complexity for linear was O(n), while binary was O(log n).

4 Conservative Remapping

When it comes to the remap of a target mesh onto a source mesh, point-wise interpolation is a fine method to use. However, the problem is our mass, or area under the curve of our field data line, is not conserved with point-wise interpolation. Conservative remap uses the area under the curve of a source field (See Figure 11) to interpolate field data on-to a target mesh such that the total area is conserved after remap. The source field is defined at the source cell center instead of the node, and we can find the area under the line curve of these points. We approximate the area under the curve by using rectangles (integrals), giving a specific area to each individual source cell:

$$area = width * height \tag{7}$$

We find the field value of the nodes by taking the area of the cell and dividing it by the width of the cell, giving us the value at the midpoint of the cell. We will discuss the process of conservatively remapping a target

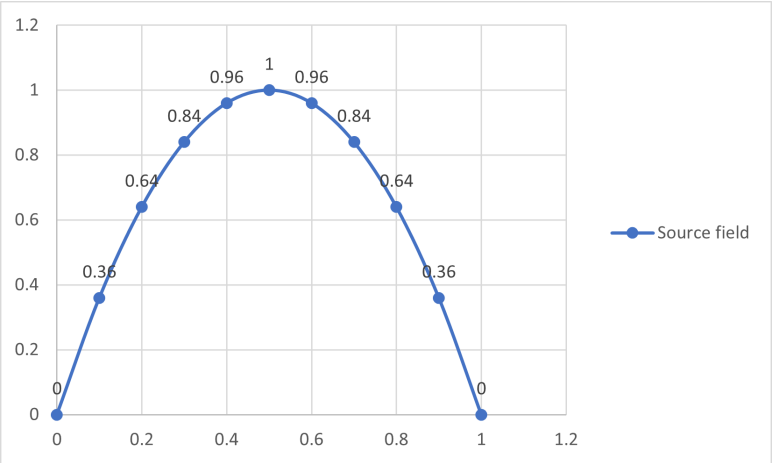


Figure 7: Source field data using a quadratic function

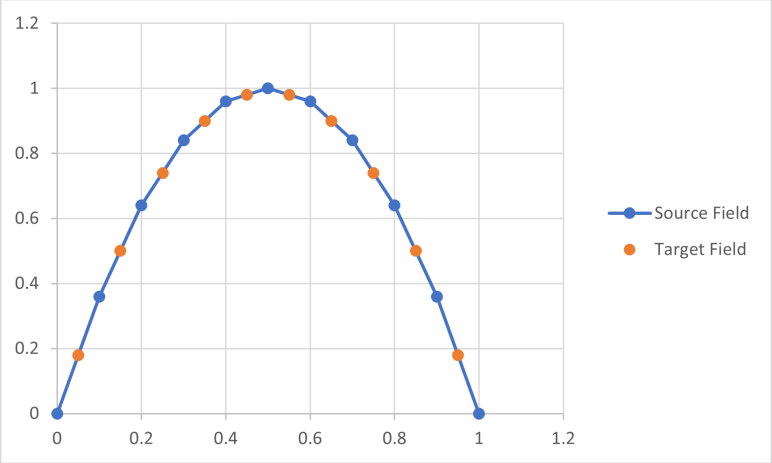


Figure 8: Target points after interpolation with source field

mesh onto a source mesh using a 3 step process: Search, Intersect, and Interpolate.

4.1 Search

The first step for conservative remap is to find the source cells that overlap with the target cells containing the points that we want to interpolate. We will do this using search. Our linear and binary searches will work similarly to the ones we used for linear interpolation, however we will need to find two points this time: the source cell that the target start node lies in and the source cell that the target end node lies in. Once we find which cells these points lie in, we can get the source cells in between as well.

Suppose we have a source and a target mesh and we want to find which source mesh cells the second target cell is in, as shown in Figure 12. We first would use the starting node of the target cell (.2) and find which source cell it is in with a search. We then would take the ending node of the target cell (.4) and do the same. As shown in Figure 13a, the target start node lies in source cell 1 and the target end node lies in source cell 3. Once we find the start and end source cells, we can find the cells in between (See Algorithms 3 and 4 for the complete linear and binary searches). We then return all of our source and cells found for use in our next section (Figure 13b).

4.2 Intersect

Once we find our overlapped source cells, we need to find how much of each source cell is intersecting with the target cells, so that we can properly approximate the area. Here is the process used for this (using our prior example):

- For the first found source cell, we take the source cell end node and subtract the target cell start node from it (Figure 14a).

Algorithm 3 Linear Search Function

Linear search function takes in a target start node (Ts), target end node (Te), an array of source nodes (Sn[n]), and the number of source nodes (n) and it returns a vector containing the source cells found.

Fc = First source cell
Lc = Last source cell
$vectorsize = 0$
n = number of source nodes
initialize i to 0
for i **to** n **do**
  if $(Ts \geq Sn$[i]) and $(Ts \leq Sn$[i+1]) **then**
    initialize Fc to i
  end if
  if $(Te \geq Sn$[i]) and $(Te \leq Sn$[i+1]) **then**
    initialize Lc to i
  end if
end for

Once the first and last cells are found, get the size of the vector to be returned. Use a for loop and initialize i to equal the first cell index, and continue the loop until i equals the last cell index.

initialize i to Fc
for i **to** Lc **do**
  $vectorsize$ ++
end for

Fill vector with the indexes of the source cells found and return.

Vector $SourceCellsFound$
initialize i to 0
for i **to** $vectorsize$ **do**
  $SourceCellsFound$[i] $= Fc + i$
end for

return $SourceCellsFound$; ▷ returns vector of source cells that overlap target nodes

Algorithm 4 Binary Search Function

Binary Search function takes in the same parameters as linear search. (Target start node (Ts), target end node (Te), an array of source nodes (Sn[n]), and the number of source nodes (n)

Fc = First source cell
Lc = Last source cell
$vectorsize = 0$
$first = 0$
$last$ = number of source cells

for i **to** n **do**
    $mid = first + ((last - first)/2)$ ▷ formula to get middle of array

    if $Ts \geq Sn$[mid]) and ($Ts \leq Sn$[mid+1]) **then**
        $Fc = mid$
        break
    end if

    if ($Ts > Sn$[mid+1]) **then**
        first = mid+1
        mid = (last-first)/2
    end if

    if ($Ts < Sn$[mid]) **then**
        last = mid;
        mid = (last-first)/2;
    end if

end for

Repeat the above algorithm to find the source cell for the target end node as well (replace "Ts" with "Te"). Once we find the first and last source cell needed, Repeat steps shown in Algorithm 3 to fill and return the vector.

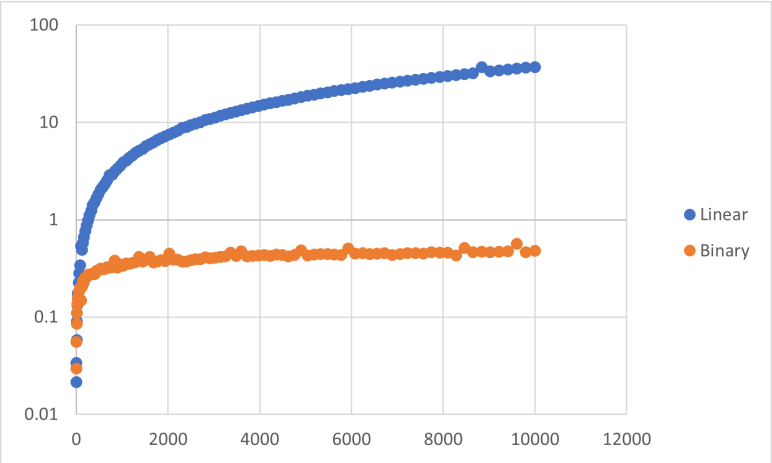


Figure 9: Performance comparison of linear and binary searches up-to 10,000 source cells

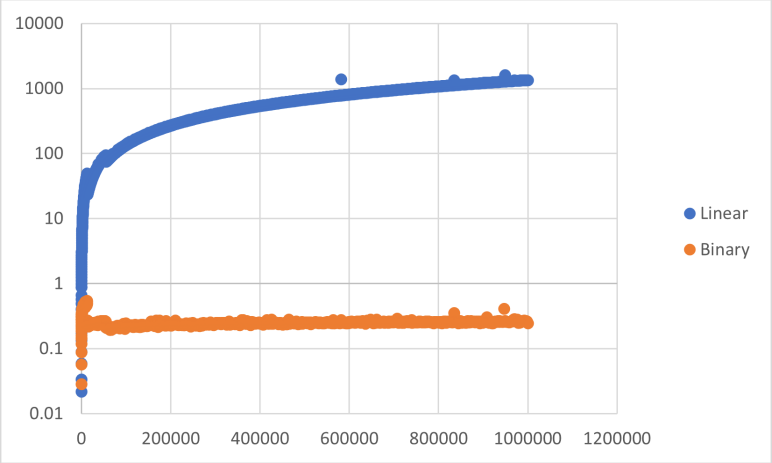


Figure 10: Performance comparison of linear and binary searches up-to one million source cells.

- For the middle source cells (if any), since they will use up all of the volume of the cells, we subtract the source cell start node from the source cell end node for each of those cells. This gives us the full width of each middle cell (Figure14b).
- Finally for the last found source cell, subtract the target cell end node from the source cell start node (Figure 14c).
- For the case where the two target points are within the same source cell, the amount intersected will be the target end node - the target start node. (Figure 14d).

We now have found our intersected values with each of the source cells and can move on to the final step: Interpolation.

4.3 Interpolate

We have now found our overlapped source cells with the target cells, and our intersected values of the overlapped source cells. For interpolation at the midpoint of a target cell, we can get the field value using target cell width and the area target cell area. We already know the target cell width using:

$$h_t = x_{start} - x_{end} \tag{8}$$

where x_{start} and x_{end} are the start and end nodes of the target cell. And we have the information needed to compute the area from search and intersect steps.

To find the area of the target cell, we use the intersected values of each source cell and multiply them by the corresponding heights (source field at source cell center) of each overlapping source cell. We take the total of all these values for the total area (See Figure 15). Once the target area is found, we interpolate the field value

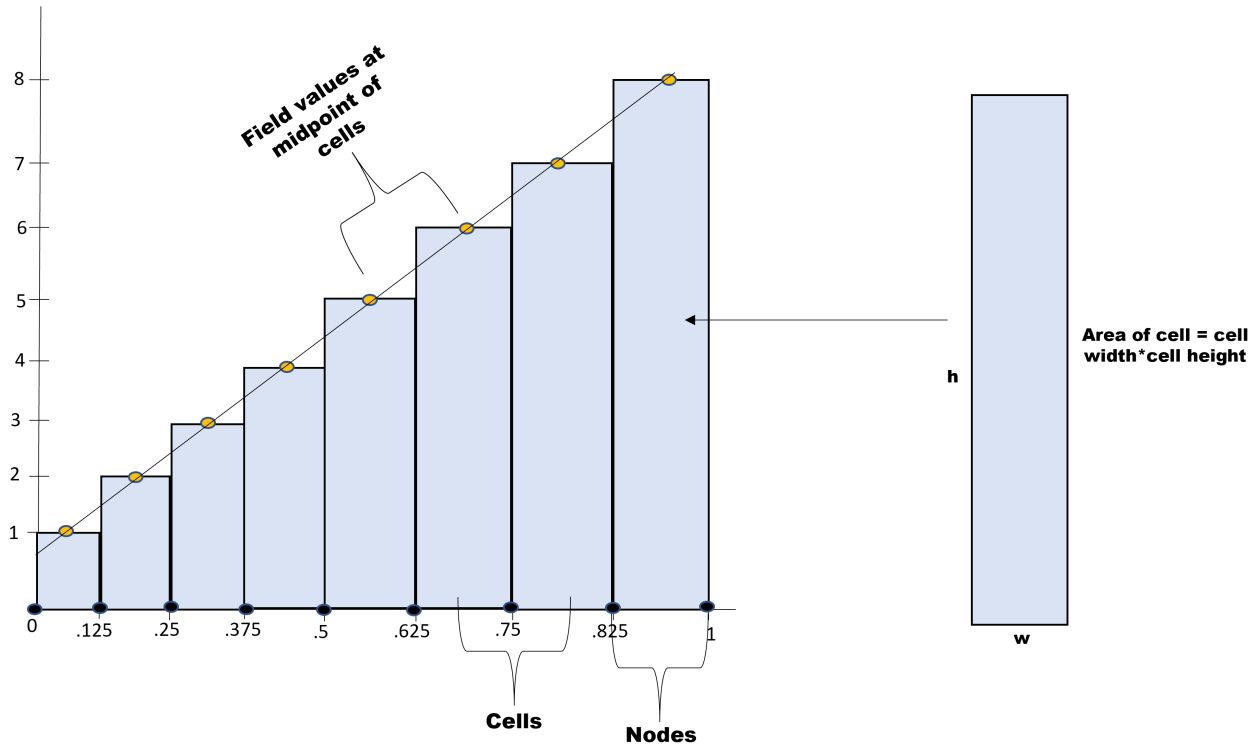


Figure 11: Mesh Illustration with labeled cells and field values

at the midpoint with this equation:

$$y_t = \frac{\sum_{n=1}^{N_s} I_n^{st} * y_n^s}{h_t} \tag{9}$$

where y_t is the approximated target field value, N_s is the number of source cells the target overlaps with, I^{st} is the intersection amount of the target cell with the overlapped source cells, y_n^s is their source field values, and h_t is the target cell width. Using this process, we have successfully interpolated the field value of a target mesh while conserving the area under the source mesh curve.

4.4 Numerical Results

4.4.1 Verification Using Linear Function

Because the goal of conservative remap is to conserve the area under the curve, we can verify this by remapping a target mesh of the same domain as a source mesh. Once the remap is done, the area under the curve of both the source and target mesh should be equal.

For this verification, we use a source mesh with with 8 cells and a target mesh with 5 cells (the cell centers are tabulated in Table 3. Our field values lie at the midpoints of these cells. We used the gradient:

$$y = 8x + 1/2 \tag{10}$$

This allows the y-values to increase by 1 each point. Figure 16a shows the source field data and Figure 16b shows the field data of the target mesh after it is interpolated. The area under the curve was preserved for both the target and source mesh 4, totaling out to 4.5 for each, making our verification successful.

4.4.2 Interpolation with Quadratic Equation

After verification that area is conserved under the curve, we interpolated a target mesh using a source mesh with a quadratic equation:

$$y = -4(x - 1/2)^2 + 1 \tag{11}$$

Our source mesh is equidistant between 0 and 1, and has 10 cells with field data on the midpoints of each cell. Our target midpoints are between each of the source midpoints. Figure 17a shows the source cells with the midpoints and field values and Figure 17b shows the interpolated target mesh using the source. We have now interpolated a target mesh more accurately, because the area under the curve of the line has been conserved. Table 5 displays the area under the source and target curve (note that our areas are different because the meshes are over slightly different domains). This method is much more efficient than piece-wise linear approximation.

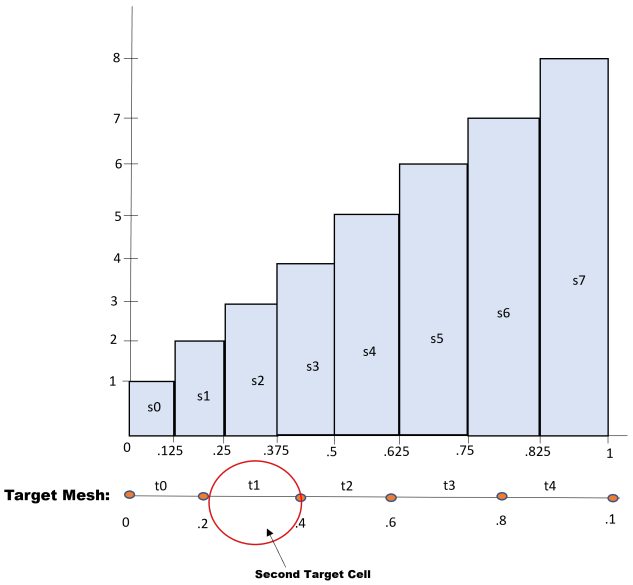
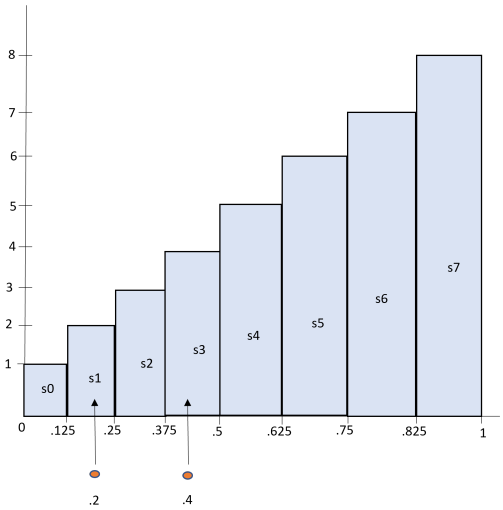
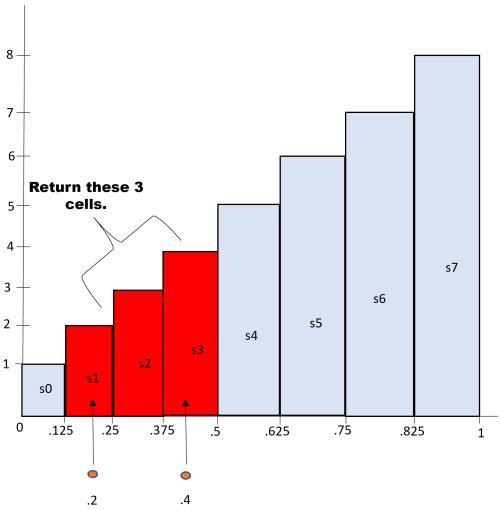


Figure 12: Source and target cells with second target cell circled



(a) Target Points found in Source cells 1 and 3



(b) Return the source cells found

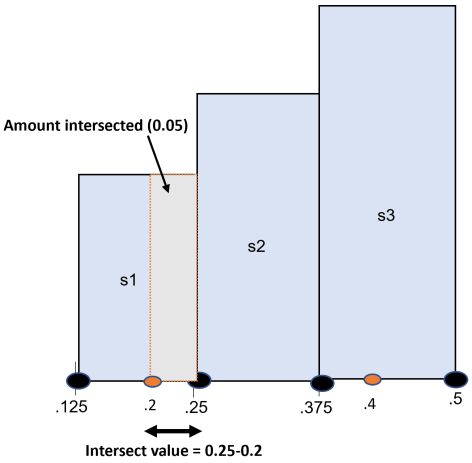
Figure 13: Search the overlapping source cells with the target cell.

4.4.3 Profiling Results

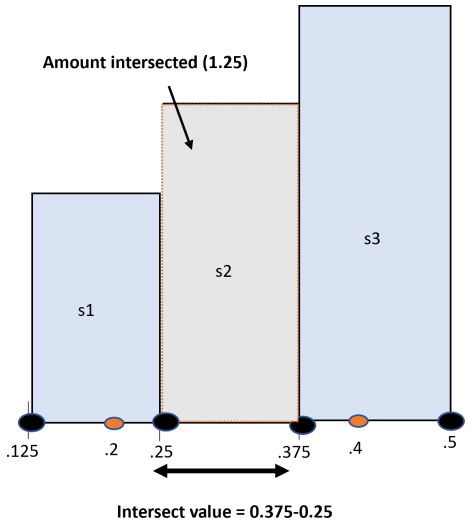
Again, we timed our linear and binary searches for to see which found our desired information more efficiently. Figure 18a displays searches going from 1 cell to 10,000 and Figure 18b goes from 1 to 1 million. In both cases, binary search yields faster results as expected.

5 Conclusion

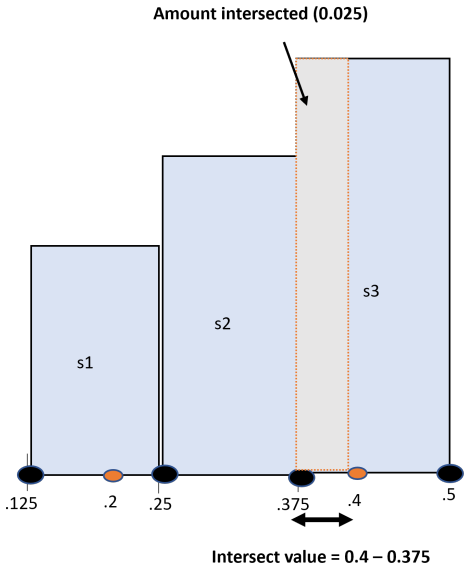
Overall, the completion of this study lead us to understand data remap of two meshes on a one-dimensional plane using both Point-wise and Conservative remap. We found that while point-wise does allow us to complete remap, it doesn't conserve mass, which is important when dealing with meshes containing specific properties that require the use and conservation of mass in order to be represented properly. The area of our source mesh and our remapped target mesh over the same domain were the same in both of our numerical tests, verifying that we didn't lose any data when going from the source to the target. We also were able to find differences in the performances of our searches we used when completing our code, and how binary search was faster in finding our nodes and cells than linear.



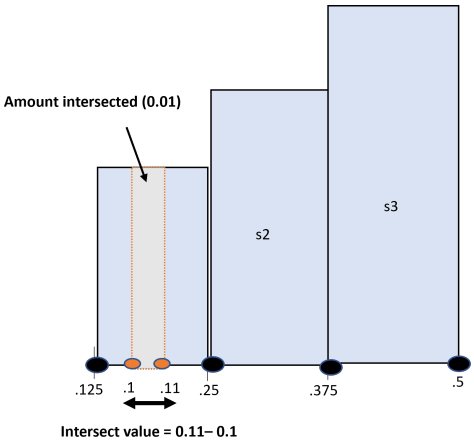
(a) First cell



(b) Middle cell(s)



(c) Last cell



(d) Target cell inside a source cell.

Figure 14: Finding overlap of the target cell with the source cells.

Source Cell Midpoints	Field Values
0.0625	0.1
0.1876	0.3
0.3125	0.5
0.4375	0.7
0.5625	0.9
0.6875	
0.8125	
0.9376	

Table 3: Source and Target Cell Centers.

Area under curve preserved
Source: 4.5000000000000000
Target: 4.5000000000000000

Table 4: Area under the curve for source and target for linear function.

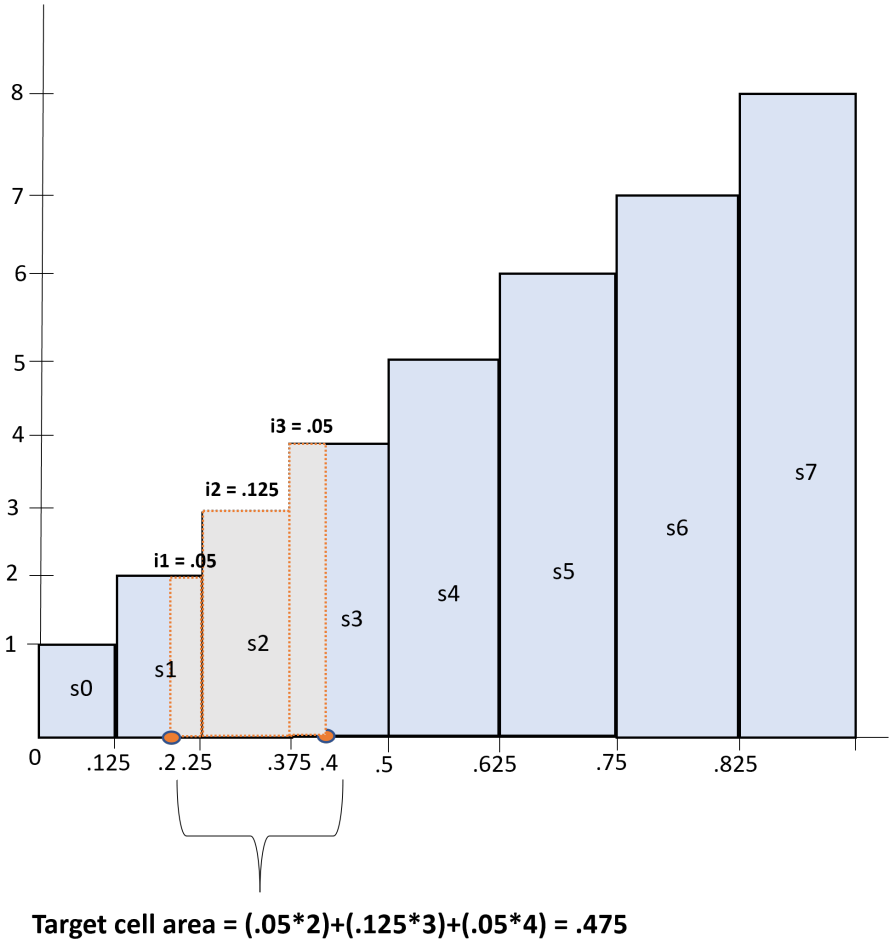
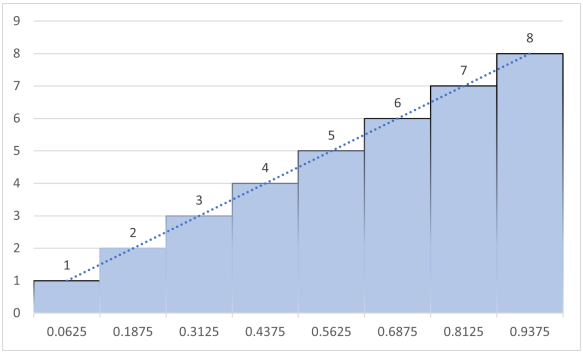
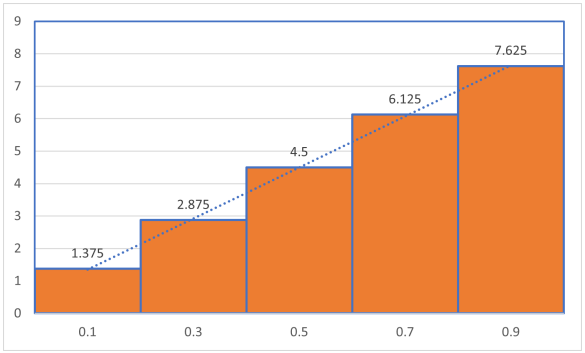


Figure 15: Summation of Source Cell intersect values x corresponding Source Cell Height



(a) Source Field Data at Cell Midpoints

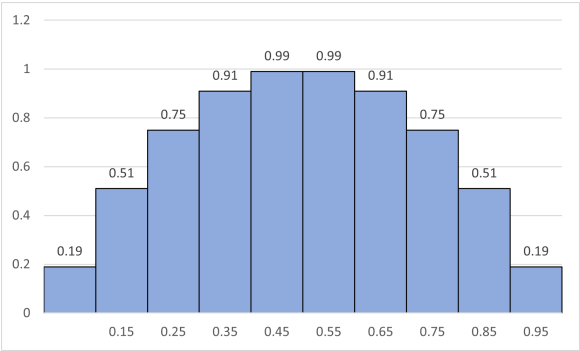


(b) Target points after interpolation

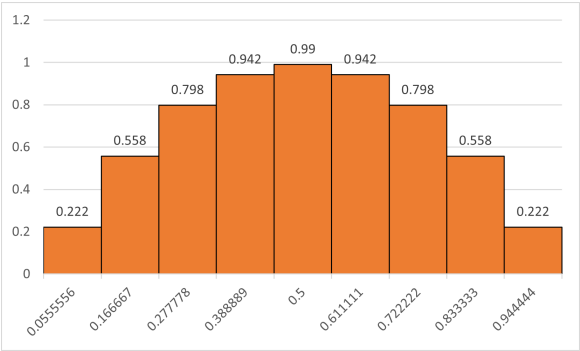
Figure 16: Source field and interpolated field on target mesh.

Area under curve preserved
Source: 0.6700000166893005 Target: 0.6700000166893005

Table 5: Area under the curve for source and target for quadratic function.

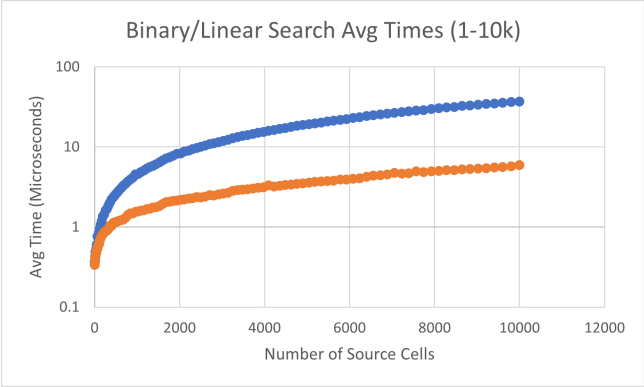


(a) Source cells with midpoints and field values

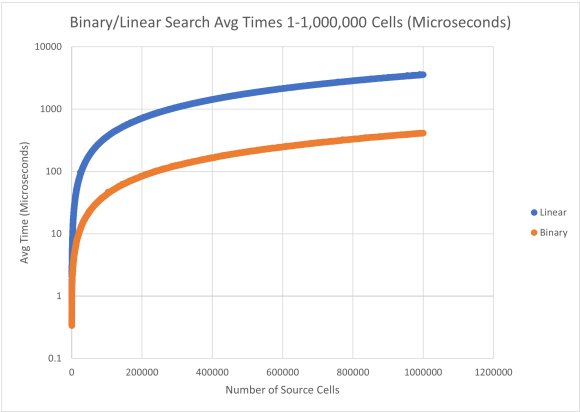


(b) Target points after interpolation

Figure 17: Source field using a quadratic function, and interpolated field on target.



(a) Up-to 10,000 cells.



(b) Up-to 1,000,000 cells.

Figure 18: Performance comparison of linear and binary search.