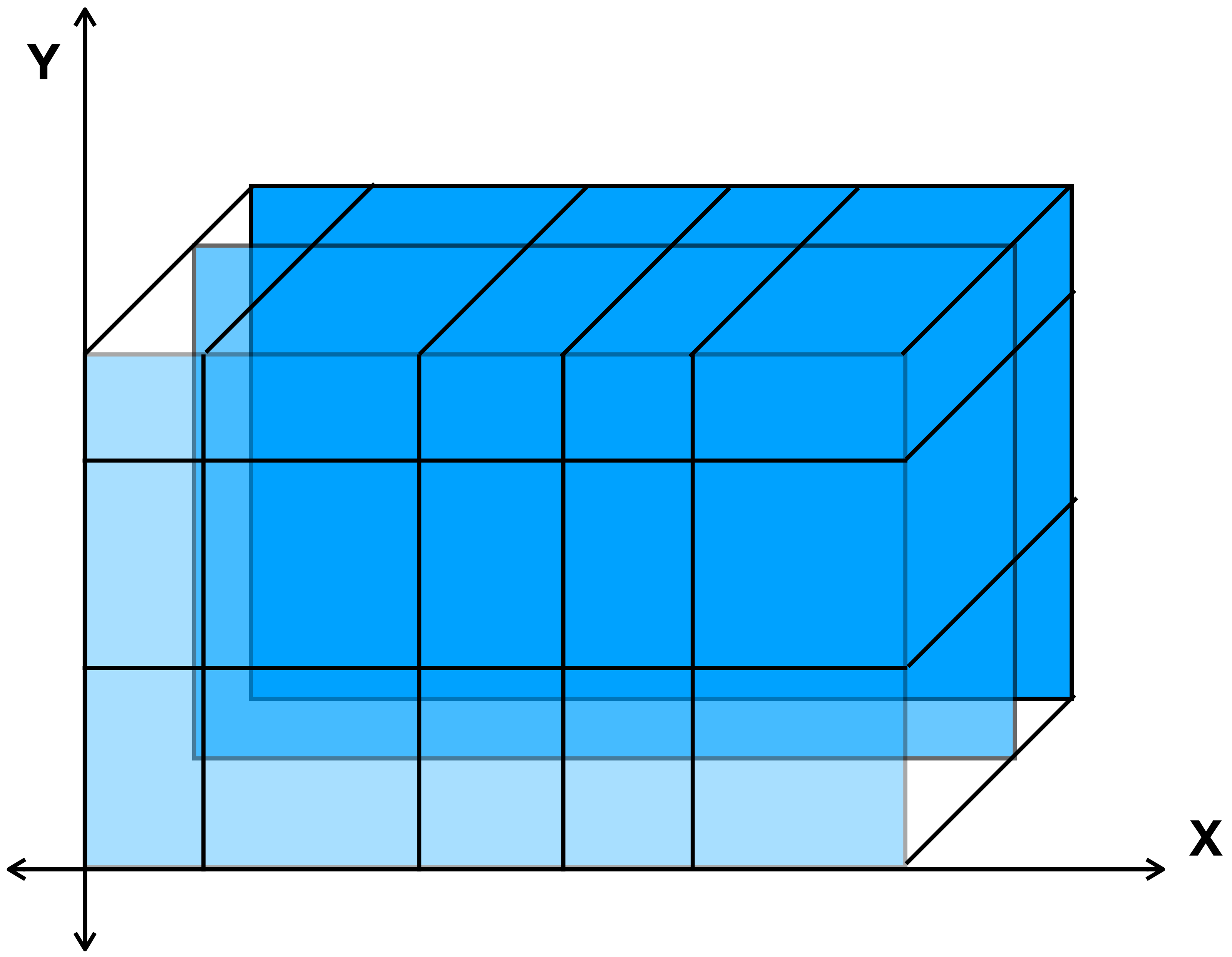
Cohesion

3-D Grid



Grid Data

dimX:

1.0	2.0	1.0	1.0	1.5
0	1	2	3	4

The width of each cell

minColDim: 1.0

The smallest width

distX:

0.5	2.0	3.5	4.5	5.75
0	1	2	3	4

The distance from the origin to the center of each cell

Fix this code

```
for (int i = 0; i < numCols; i++ ) {
    if (i == 0) {
        minColDim = dimX[i];
        distX[i] = dimX[i] / 2.0;
    } else {
        if (dimX[i] < minColDim) {
            minColWidth = dimX[i];
        }
        distX[i] = distX[i - 1] + dimX[i - 1] / 2.0 + dimX[i] / 2.0;
    }
}
```

What could be done to make the code easier to read and maintain?

Improvement 1

```
minColDim = dimX[0];
distX[0] = dimX[0] / 2.0;
for (int i = 1; i < numCols; i++ ) {
    if (dimX[i] < minColDim) {
        minColWidth = dimX[i];
    }

    distX[i] = distX[i - 1] + dimX[i - 1] / 2.0 + dimX[i] / 2.0;
}
```

Cohesion

- “The degree to which elements inside a module belong together”

  https://en.wikipedia.org/wiki/Cohesion_(computer_science)

- Highly cohesive code is good.
  - Less complex
  - More readable
  - Easier to maintain and test
- Your code may improve if you look at every block (not just a module or class).
- Cohesion is often associated with loose coupling among objects.

Improvement 2

```
minColDim = dimX[0];
for (int i = 1; i < numColumns; i++) {
    if (dimX[i] < minColDim) {
        minColDim = dimX[i];
    }
}

distX[0] = dimX[0] / 2.0;
for (int i = 1; i < numColumns; i++) {
    distX[i] = distX[i - 1] + dimX[i - 1] / 2.0 + dimX[i] / 2.0;
}
```

Improvement 3

```
minColDim = std::min_element(dimX, dimX + numCols);

distX[0] = dimX[0] / 2.0;
for (int i = 1; i < numColumns; i++) {
    distX[i] = distX[i - 1] + dimX[i - 1] / 2.0 + dimX[i] / 2.0;
}
```

For Java, see Collections.min(Collection)

For Python, see min(iterable)

Improvement 4?

```
minColDim = std::min_element(dimX, dimX + numCols);

float dimOver2 = new float[numCols];
for (int i = 0; i < numCols; i++) {
    dimOver2[i] = dimX[i] * 0.5;
}

distX[0] = dimOver2[0];
for (int i = 1; i < numColumns; i++) {
    distX[i] = distX[i - 1] + dimOver2[i - 1] + dimOver2[i];
}
```