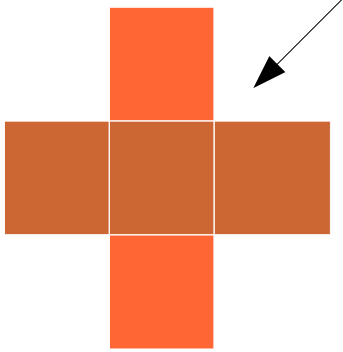
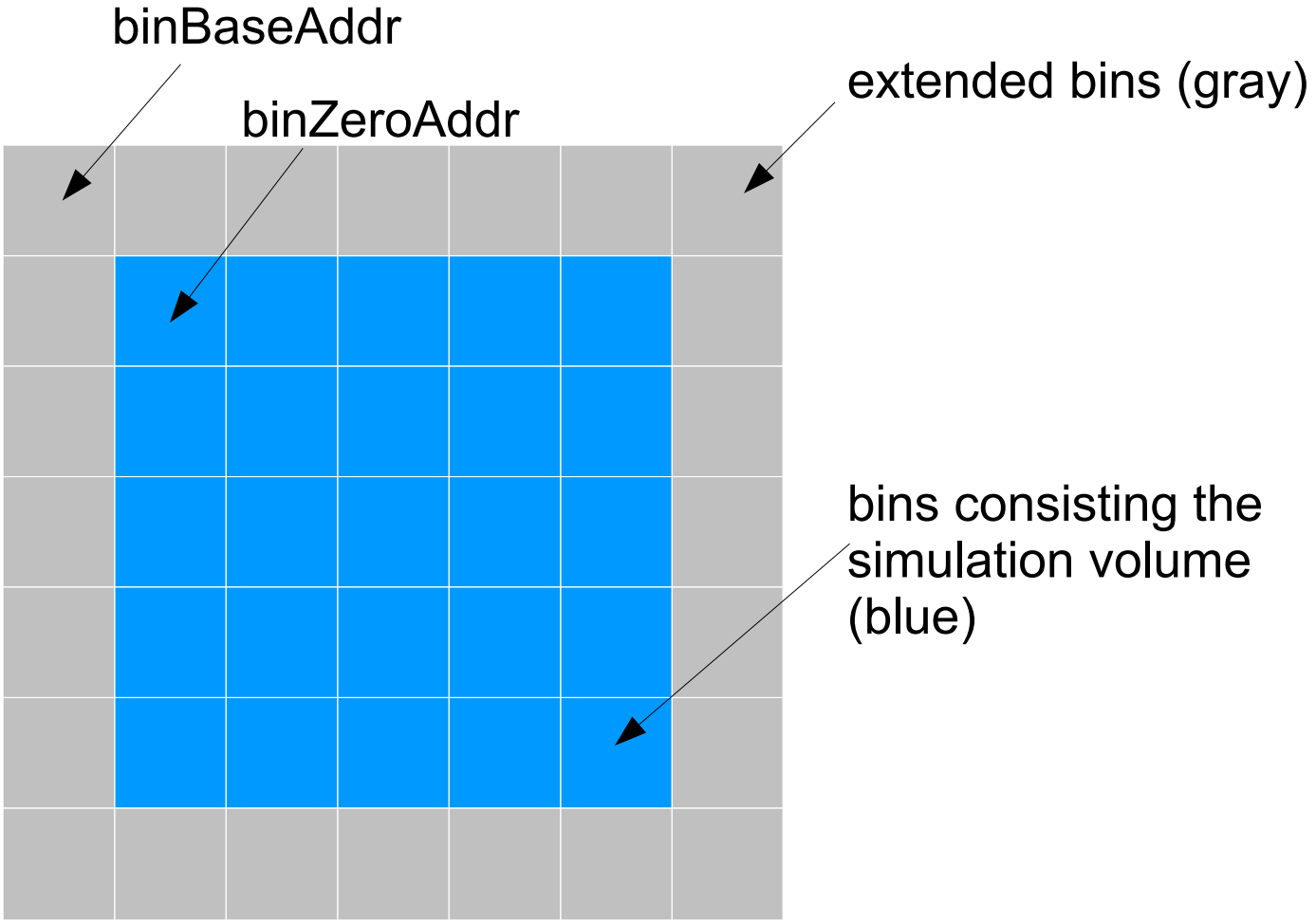
How to get surrounding bins

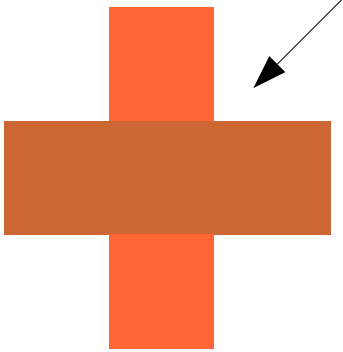
Neighborhood offset list =
{ (0,0), (-1, 0), (0, 1), (1, 0), (0, -1) }

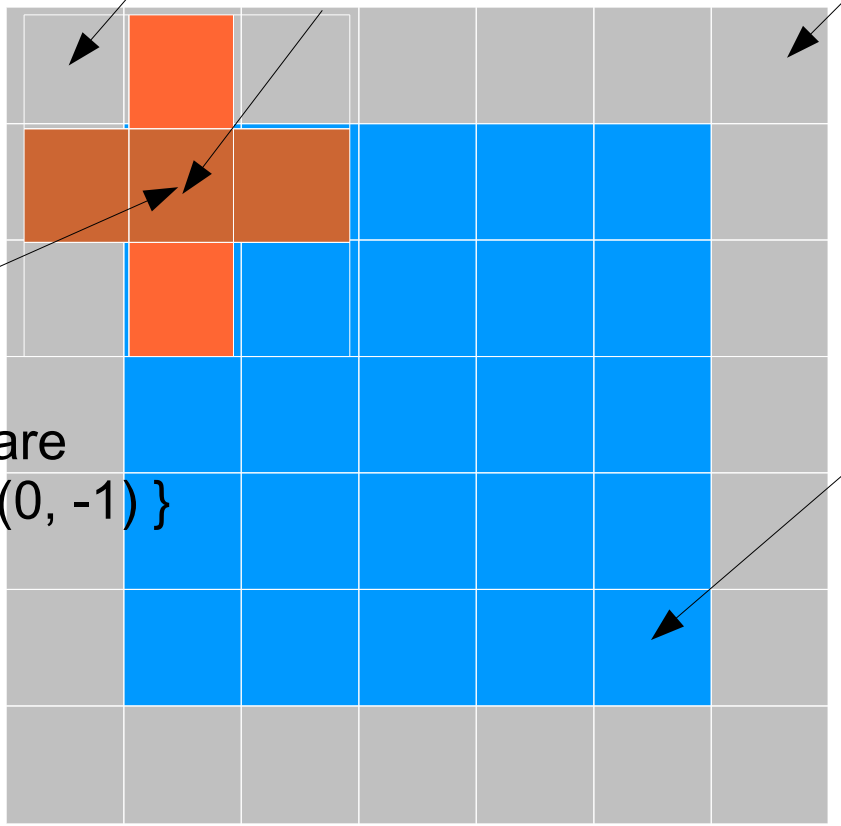




How to get surrounding bins

Neighborhood offset list =
{ (0,0), (-1, 0), (0, 1), (1, 0), (0, -1) }





binBaseAddr

binZeroAddr

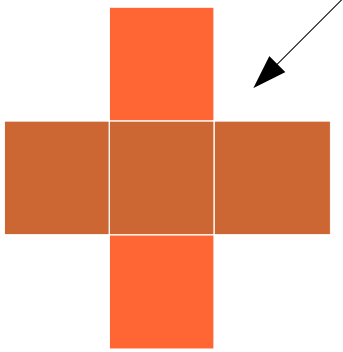
extended bins (gray)

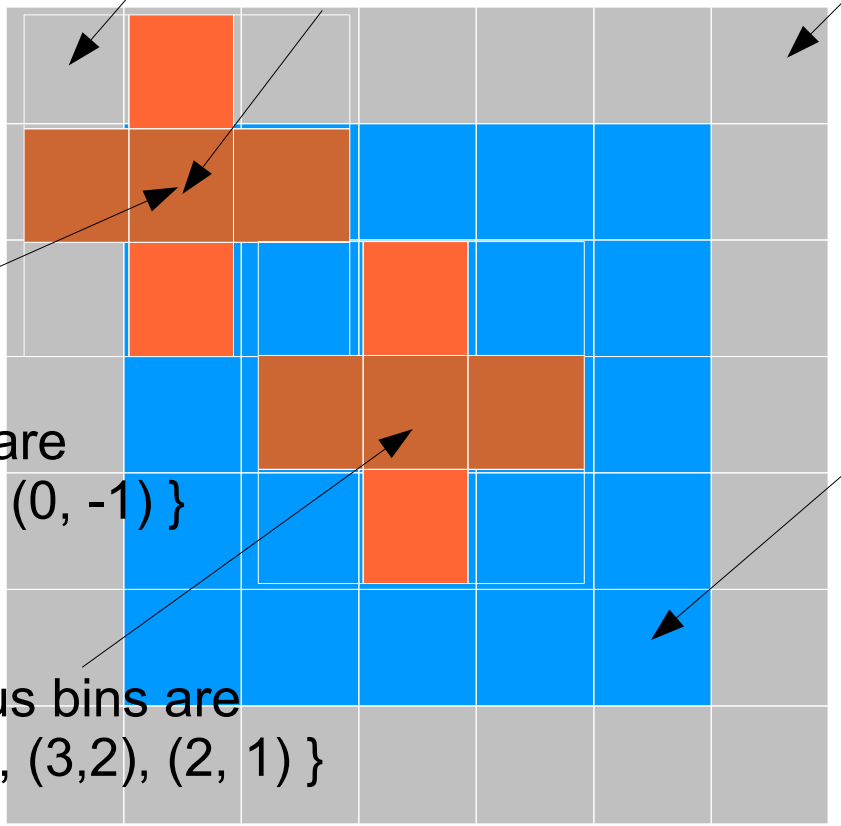
Position = (0, 0), thus bins are
{ (0,0), (-1, 0), (0, 1), (1,0), (0, -1) }

bins consisting the simulation volume (blue)

How to get surrounding bins

Neighborhood offset list =
{ (0,0), (-1, 0), (0, 1), (1, 0), (0, -1) }





binBaseAddr

binZeroAddr

extended bins (gray)

Position = (0, 0), thus bins are
{ (0,0), (-1, 0), (0, 1), (1, 0), (0, -1) }

bins consisting the simulation volume (blue)

Position = (2, 2), thus bins are
{ (2, 2), (1, 2), (2, 3), (3,2), (2, 1) }

```
// Begin of solution

for (j = 0; j < grid.y; j++) {
  float y = gridspacing * (float) j;
  long int voxaddr = grid.x*grid.y*k + grid.x*j;

  for (i = 0; i < grid.x; i++) {
    x = gridspacing * (float) i;

    int bx = (int) floorf(x * BIN_INVLEN);
    int by = (int) floorf(y * BIN_INVLEN);
    int bz = (int) floorf(z * BIN_INVLEN);

    float e = 0.0;
    for (n = 0; n < NbrListLen; n++) {
      int px = bx + NbrList[n].x;
      int py = by + NbrList[n].y;
      int pz = bz + NbrList[n].z;

      if ((0 <= px && px <= binDim.x) &&
          (0 <= py && py <= binDim.y) &&
          (0 <= pz && pz <= binDim.z)) {

        // iterate every atoms in the bin
        int index = (pz * binDim.y + py) * binDim.x + px;
        float4* bin = &binZeroAddr[index * BIN_DEPTH];
        int numAtomsInBin = bincntZeroAddr[index];

        for (m = 0; m < numAtomsInBin; m++) {
          float dx = bin[m].x - x;
          float dy = bin[m].y - y;
          float dz = bin[m].z - z;
          float q = bin[m].w;

          float dist = sqrtf(dx*dx + dy*dy + dz*dz);
          if (q == 0.0 || dist > cutoff) continue;

          float t = q / dist;
          float root_s = 1.0 - ((dist * dist) / (cutoff * cutoff));
          e += t * root_s * root_s;
        }
      }
    }
    energygrid[voxaddr + i] = e;
  }
}
```

```
for (j = 0; j < grid.y/TILE_SIZE; j++) {
  float y = gridspacing * (float) j * (float) TILE_SIZE;

  for (i = 0; i < grid.x/TILE_SIZE; i++) {
    x = gridspacing * (float) i * (float) TILE_SIZE;

    // #1. surrounding bins first
    int bx = (int) floorf(x * BIN_INVLEN);
    int by = (int) floorf(y * BIN_INVLEN);
    int bz = (int) floorf(z * BIN_INVLEN);

    memset(e, '\0', TILE_SIZE*TILE_SIZE*sizeof(float));

    for (n = 0; n < NbrListLen; n++) {
      int px = bx + NbrList[n].x;
      int py = by + NbrList[n].y;
      int pz = bz + NbrList[n].z;

      if ((0 <= px && px <= binDim.x) &&
          (0 <= py && py <= binDim.y) &&
          (0 <= pz && pz <= binDim.z)) {

        // iterate every atoms in the bin
        int index = (pz * binDim.y + py) * binDim.x + px;
        float4* bin = &binZeroAddr[index * BIN_DEPTH];
        int numAtomsInBin = bincntZeroAddr[index];

        for (w = 0; w < TILE_SIZE; w++) {
          for (v = 0; v < TILE_SIZE; v++) {
            for (m = 0; m < numAtomsInBin; m++) {
              float tx = v * gridspacing;
              float ty = w * gridspacing;

              float dx = bin[m].x - x - tx;
              float dy = bin[m].y - y - ty;
              float dz = bin[m].z - z;
              float q = bin[m].w;

              float dist = sqrtf(dx*dx + dy*dy + dz*dz);
              if (q == 0.0 || dist > cutoff) continue;

              float t = q / dist;
              float root_s = 1.0 - ((dist * dist) / (cutoff * cutoff));
              e[w*TILE_SIZE+v] += t * root_s * root_s;
            }
          }
        }
```