LA-SIGMA REU-SUBR

Optimizing Stereographic Visualization of Bonds in Atomistic Configurations Using Linked Lists

Jasmine Jones[1], Sanjay Kodiyalam[2], Amitava Jana[2]

[1]Department of Chemistry, Xavier University of Louisiana,

[2]Department of Mechanical Engineering, Southern University Baton Rouge[2]

7/26/2011

ABSTRACT

The code in a C/C++ Visual Studio project that is used to visualize bonds in atomistic configurations is further optimized via the use of linked lists. Benchmarking the original version of the code on a 2 x dual core, 3.22 GHz, Opteron when visualizing a Body Centered Cubic (BCC) lattice with all 8-bonded atoms showed that for having an interactive frame rate of 10 frames per second the upper limit on the number of atoms (= N) is ~33,600 when visualizing atoms alone and ~ 3,900 when visualizing bonds alone. The order N^2 dependence of the execution time to determine the bonded atoms and then display the bonds is changed to order N via the use of liked lists thereby raising the limit to ~15,400 atoms when visualizing bonds alone. The corresponding limit when visualizing both atoms and bonds is ~11,200 atoms. When applied to visualizing a configuration with ~47,000 atoms (4- and 6- bonded), from a molecular dynamics simulation of Alumina, the frame rate was 2.52 frames per second indicating the need for further optimization.

INTRODUCTION [1]

Visualizing atomistic configurations from simulations provides information complementary to numerical data and helps in identifying the mechanisms underlying the phenomena being studied [2]. Molecular dynamics simulations can have a large number of atoms: In many cases exceeding a million atoms. This makes it challenging for interactively visualizing the corresponding atomic configurations. For interactive visualization the frame rate must at least be 10 frames per second (fps) implying that the

time for displaying the configuration once must be ≤ 0.1 seconds. In this work an earlier version of a CAVE-library based C/C++ Visual Studio [1, 3] project in enhanced to increase the number of atoms that can be handled interactively.

The CAVE at Southern's College of Engineering (Fig. 1) is an 8 ft x 8 ft x 8 ft space with four displays – three on screen-walls and the fourth on the floor. Active stereographic viewing is enabled via the use of eye-ware synchronized to the rapidly alternating display of images corresponding to the left and right eyes. The user's viewpoint is detected via a sensor connected to the eye-ware and the corresponding perspective transformation for each of the displays is automatically carried out by the CAVE-Library. A second sensor is connected to a joystick and can be used for interacting with the user's visualization application. The CAVE is driven by a two node cluster with the Master node collecting and handling the sensors' information and the display node driving the projectors.

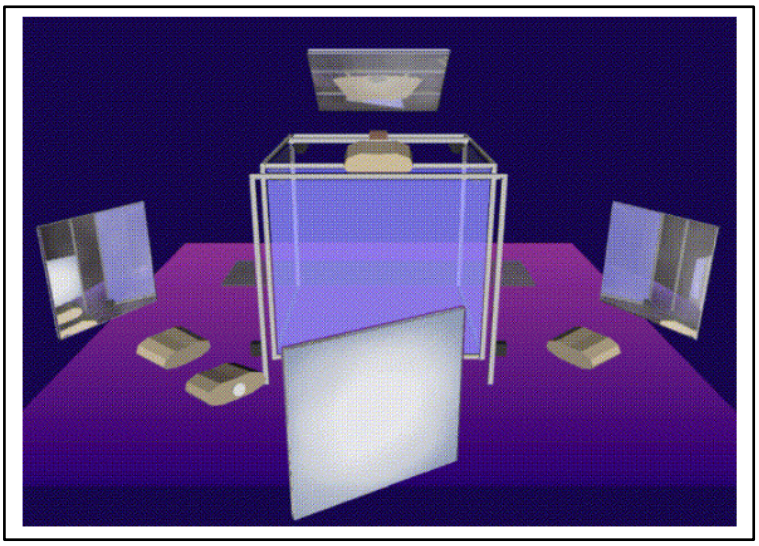

Figure 1. Schematic [4] of the CAVE at Southern's College of Engineering. Projection is via mirrors used to set the correct optical distance equal to the projector's throw.

LITERATURE REVIEW

Previous research [1, 3] developed the Visual Studio for visualizing atoms and bonds and tested the execution speed on the code on a 2.41 GHz, 2 x dual core Opteron machine when visualizing a BCC lattice with all 8-bonded atoms *i.e.* all nearest neighbor atoms are bonded. OpenGL display lists were used to render atoms of two types and bonds of one type. For having an interactive frame rate of 10 frames per second (fps) the upper bound on the number of atoms (= N) was ~30,000 when displaying atoms alone and ~3,000 when displaying bonds alone [1]. It was suggested that the bottleneck during the display of bonds may be overcome with the use of linked lists [1, 3].

METHODOLOGY

In order to measure improvements in execution speed on the current machine, the Master node in the CAVE (a 3.22 GHz, 2 x dual core Opteron), the execution of original code is benchmarked on this machine – in the non-stereo CAVE simulator mode as before [1, 3]. The entire configuration is visualized during this benchmarking as larger frame rates result when part of the configuration are beyond the user's view – see the "Application" section for an example.

When visualizing the BCC lattice, the execution time for displaying atoms alone scales linearly with N (Fig. 2a), as before [1], resulting in the upper bound of N = 33,600 for an interactive frame rate. When displaying bonds alone the execution time scales as N^2 (Fig. 2b), as before [1], with the corresponding limit of N = 3,900 for an interactive frame rate. This N^2 dependence of the execution time is due to the double loop over the atoms in the original code (Fig. 3) used to determine the bonded atoms.

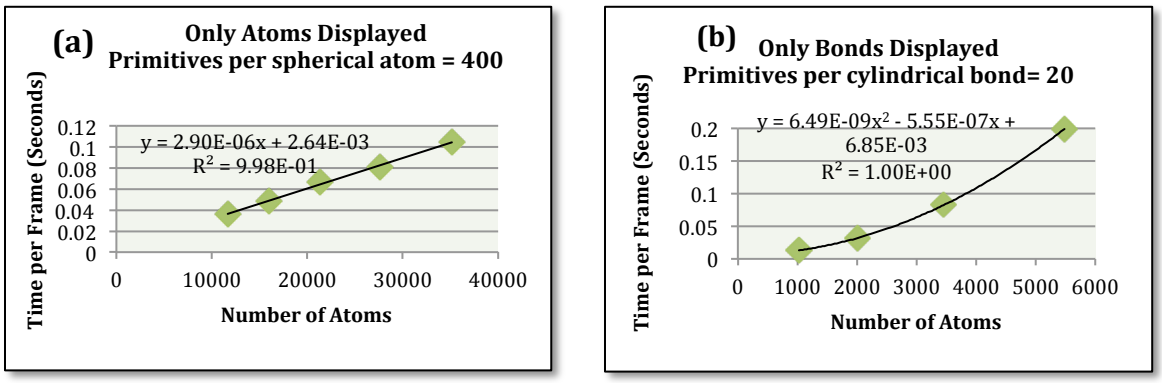

Figure 2. Variation in the execution time of the original code on the 2 × dual core, 3.22 GHz, Opteron when visualizing a BCC lattice with all 8-bonded atoms. (a) Only atoms displayed as spheres. (b) Only bonds displayed as cylinders.

```
for(int j=0;j<NAtoms;j++)
{
    glPushMatrix();

    x1= Position[j*3]; y1= Position[j*3+1]; z1= Position[j*3+2];
    glTranslated(x1,y1,z1);

    AtomType1= AtomType[j];
    for (int k=j+1; k<NAtoms;k++)
    {

        if (AtomType1==AtomType[k])continue;
        x2= Position[k*3]; y2=Position[k*3+1]; z2=Position[k*3+2];

        x=x2-x1; y=y2-y1; z=z2-z1; Distance2=(x*x)+(y*y)+(z*z);

        if (Distance2>=BondLength2)continue;

        glPushMatrix();

        [Open GL code for displaying bonds as cylinders]

        glPopMatrix();
    }
    glPopMatrix();
}
```

Figure 3. Part of the original code determining the bonded atoms and then displaying the bonds.

MODIFIED CODE

The 2nd (inner) loop over the atoms (Fig. 3) is avoided by "putting atoms into boxes" of size equal to the bond length (Fig. 4a) and searching only the boxes neighboring a particular box (Fig. 4b) to determine bonded atoms.

(a)

```
for (int i=0;i<NAtoms;i++)
{
    box_index_X=((Position[3*i]-PositionMin[0])/(box_size));
    box_index_Y=((Position[(3*i)+1]-PositionMin[1])/(box_size));
    box_index_Z=((Position[(3*i)+2]-PositionMin[2])/(box_size));
    LinkedList[i]=BoxHeader[box_index_X][box_index_Y][box_index_Z];
    BoxHeader[box_index_X][box_index_Y][box_index_Z]=i;
}
```

(b)

```
for (int bx=0;bx<nob[0];bx++)
{
    for (int by=0;by<nob[1];by++)
    {
        for (int bz=0;bz<nob[2];bz++)
        {
            totalna=0;

            for(int jj=0;jj<13;jj++) // loop over "half" of the 26 neighboring boxes
            {
                nbx=allnb[jj][0]; nby=allnb[jj][1]; nbz=allnb[jj][2];
                [Code for skipping nonexistant boxes near the boundaries]

                for (int anb=BoxHeader[bx+nbx][by+nby][bz+nbz];anb>=0;anb=LinkedList[anb])
                {
                    NeighborList[totalna]=anb;
                    totalna=totalna+1;
                }
            }
            for(int aob=BoxHeader[bx][by][bz];aob>=0;aob=LinkedList[aob])
            {
                glPushMatrix();
                index1=aob*3;
                x1= Position[index1]; y1= Position[index1+1]; z1= Position[index1+2];
                glTranslated(x1,y1,z1);
                for (int dv=0;dv<totalna;dv++) // loop over atoms in neighboring boxes
                {
                    anb=NeighborList[dv];

                    if (AtomType[aob]==AtomType[anb])continue;

                    index2=anb*3;
                    x2= Position[index2]; y2=Position[index2+1]; z2=Position[index2+2];

                    x=x2-x1; y=y2-y1; z=z2-z1; Distance2=(x*x)+(y*y)+(z*z);
                    if (Distance2>=BondLength2) continue;

                    glPushMatrix();
                    [Open GL code displaying bonds as cylinders]
                    glPopMatrix();
                }
                anb0=LinkedList[aob];

                for (int anb=anb0;anb>=0;anb=LinkedList[anb])
                {
                    [Same display logic as above loop]
                }

                glPopMatrix();
            }
        }
    }
}
```

Figure 4. Part of the modified code that replaces the original code (Fig. 3) determining the bonded atoms. (a) shows construction of the *BoxHeader* and *LinkedList* arrays. (b) shows use of the *BoxHeader* and *LinkedList* arrays to determine the bonded atoms. The OpenGL code for displaying bonds as cylinders is the same as in the original code (Fig. 3).

Atoms are "put into boxes" spanning the entire system based on their position. The size of the box is chosen to be equal to the bond length so that only neighboring boxes need to be "looked at" to identify the bonded atoms. To do so two arrays are defined: The *BoxHeader* array identifying an atom in the box and a *LinkedList* array that then identifies other atoms in the box. Construction of the two arrays proceeeds as outlined in Fig. 4a while sanning over all the atoms once. Whenever an atoms is found to be in a particular box the corresponding *BoxHeader* points to this atom via its index in the *Position* array while the *LinkedList* of the newly found atom's index points to the previous value of the *BoxHeader* i.e. the previously found atom. When completed, this construction results in the *BoxHeader*(s) pointing to the last atom found in the box(es) with the LinkedList identifying the other atoms in the boxes found earlier in the construction process. Both arrays are initialized to a negative number to indicate an empty box (as in *BoxHeader*) or scanning past all the atoms in the box (as in *LinkedList*).

Bonded atoms are identified by first scanning through all the atoms – as contained in the boxes (Fig. 4b). This step is similar to the first (outer) loop over atoms of the original code (Fig. 3) – the number of operations being proportional to N. The inner loop of the original code is replaced with the construction of a list of all the atoms in the boxes neighboring a particular box (the array *NeighborList* in Fig. 4b). It is this step that makes the search for bonds scale as O(N) rather than O(N^2) - as in this step, in a system of spatially uniform atom density, the number of operations is fixed number M unlike the search carried out by the inner loop in the originl code (Fig. 3) wherein the number of operations is again proportional to N. For every atom in the initial box selected, scanning

through the *NeighborList*, the bonded atoms are then determined as before [1, 3]: to be those that are spatially seperated by a distance less than the bondlength. Duplicate determination of bonded atoms is avoided by a careful choice of only 13 of the 26 boxes neighboring a particular box – as encoded in the *allnb* array (Fig. 4b).

RESULTS AND DISCUSSION

The use of linked lists make the execution time for the search for bonded atoms scale as N - as reflected in the timing for the display of bonds alone (Fig. 5a). The corresponding limit on the number of 8-bonded atoms of the BCC lattice, for an interactive frame rate of 10 fps, increases to 15,400. It must be noted that there is a decrease in execution time when duplicate determination of bonded atoms was avoided as described earlier via the choice of scanning over only 13 of the 26 boxes neighboring a particular box rather than via a comparison of the indices of the bonded atoms' in the *Position* array while scanning over all 26 boxes neighboring a particular box: The execution time for scanning over 13 of the neighboring boxes is .129s, and the execution time for scanning over all 26 neighboring boxes is .137s.

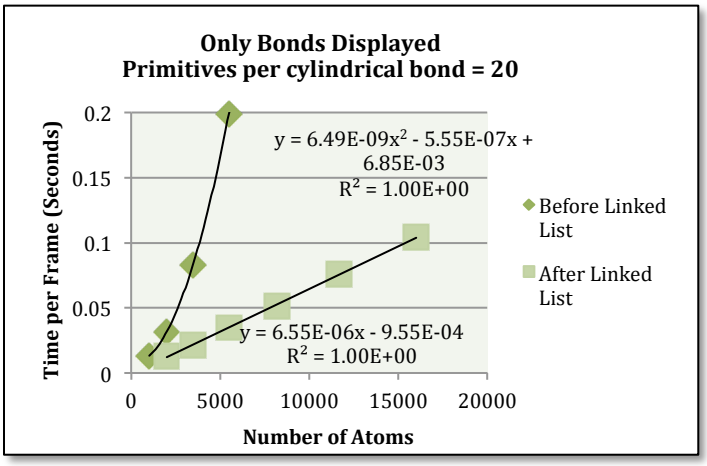

Figure 5. Variation in the execution time of the modified and original codes for the display of bonds alone. The quadratic dependence of the execution time of the original code on the number of atoms is changed to a linear one via the use of linked lists in the modified code.

As expected the execution time for the simultaneous display of both atoms and bonds now scales as order N – with upper limit of N = 11,200 for an interactive frame rate (Fig. 6). It is interesting to note that the slope of the execution time curves when displaying both atoms and bonds (Fig. 6) is less than the sum of the slopes of the curves when displaying atoms alone (Fig. 2a) and when displaying bonds alone. This observation suggests that there may be operations common to the display of both atoms and bonds – requiring further study for confirmation/identification of these operations.

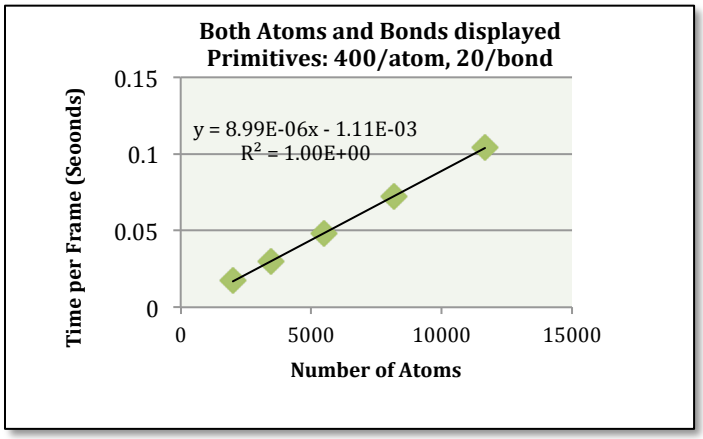

Figure 6. Variation in the execution time of the modified code to display both atoms and bonds. The linear dependence of this execution time on the number of atoms follows from the same dependence when displaying atoms alone (Fig. 2a) or bonds alone (Fig. 5).

APPLICATION

Applying the code to the entire ~47,000-atom configuration of Alumina (Fig. 7a) obtained from a molecular dynamics simulation, the execution time per frame is 0.397 s when displaying both atoms and bonds. As mentioned first under the Methodology section, this time is significantly smaller (0.285 s) when parts of this system were beyond user view. For the same system the executions time is 0.144 s when displaying atoms only, and 0.259 s when displaying bonds only. While the time for displaying atoms alone

is as expected by extrapolation of the curve in Fig. 2a, the time to display bonds alone is significantly smaller than expected by extrapolation of the linear curve in Fig. 5. This is due to the complex structure of the material (Fig. 7b) – with 4- and 6-bonded atoms.

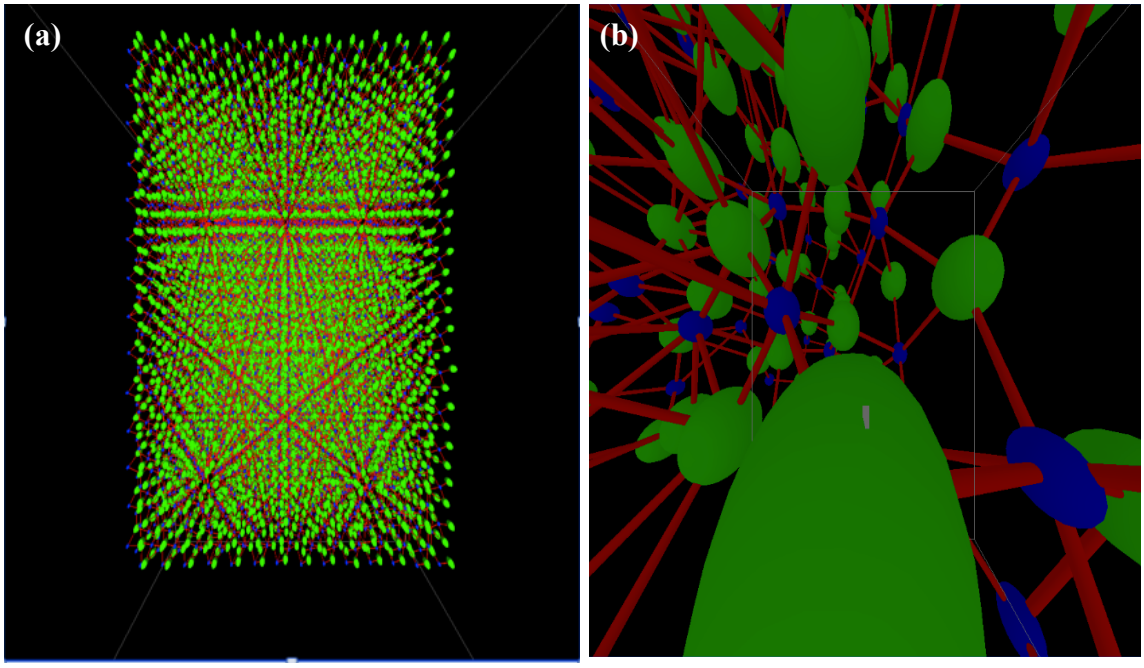

Figure 7. Application of the code to an Alumina system. (a) shows a full view with ~47,000 atoms. Blue spheres represent Aluminum, green spheres represent Oxygen, and bonds are in red. (b) shows a close-up view of the system. The structure is complex with 6-bonded Aluminum atoms and 4-bonded Oxygen atoms.

CONCLUSION AND FUTURE WORK

Linked lists make the execution time to determine the bonded atoms scale as the number of atoms. For an interactive display of atoms and bonds the current limit on the number of atoms is 11,200 (with 8-bonded atoms in a BCC lattice). Possible future optimization steps are off loading the task of determining bonded atoms from the display

loop to the main loop of the CAVE library, and Automatic lower resolution for atoms and bonds at a large distance from the viewer [5].

ACKNOWLEDGEMENTS

This work was funded by the Louisiana Board of Regents, through LASIGMA [Award Nos. EPS-1003897, and NSF (2010-15)-RII-SUBR]. One of the authors (Jasmine Jones) thanks Dr. Diola Bagayoko for the opportunity to conduct research with the LA-SiGMA REU.

REFERENCES

1. C. Vanderlick, S. Kodiyalam, A. Jana, "Optimizing Stereographic Visualization of Atomistic Configurations", LA-SiGMA REU Paper, 2012.
2. S. Kodiyalam, M. Benissan, S. Akwaboa, P. Mensah, A. Jana, and D. Bagayoko, "Molecular Dynamics Simulation and Visualization of Thermal Barrier Coatings," Proceedings of the 2012 RII LA-SiGMA Symposium, July 23rd, Baton Rouge, Louisiana.
3. G.R. Wright, S. Kodiyalam, A. Jana, "Stereographic Visualization of Molecular Configurations in a CAVE" LA-SiGMA 2011 REU Report.
4. Figure from http.cs.uic.edu/~kenyon/conference/GILKY/CAVE_DOD.html
5. A. Sharma, A. Nakano, R. K. Kalia, P. Vashishta, S. Kodiyalam, P. Miller, W. Zhao, X. Liu, T. J. Campbell, and A. Hass, *"Immersive and Interactive Exploration of Billion-Atom Systems,"* Presence: Teleoperators and Virtual Environments 12, 85 (2003).