dcdplugin.c

Go to the documentation of this file.

```
00001 /***************************************************************************
00002  *cr
00003  *cr            (C) Copyright 1995-2006 The Board of Trustees of the
00004  *cr                        University of Illinois
00005  *cr                         All Rights Reserved
00006  *cr
00007  ***************************************************************************/
00008
00009 /***************************************************************************
00010  * RCS INFORMATION:
00011  *
00012  *      $RCSfile: dcdplugin.c,v $
00013  *      $Author: johns $       $Locker:  $             $State: Exp $
00014  *      $Revision: 1.71 $       $Date: 2006/06/19 16:38:21 $
00015  *
00016  ***************************************************************************
00017  * DESCRIPTION:
00018  *   Code for reading and writing CHARMM, NAMD, and X-PLOR format
00019  *   molecular dynamic trajectory files.
00020  *
00021  * TODO:
00022  *   Integrate improvements from the NAMD source tree
00023  *    - NAMD's writer code has better type-correctness for the sizes
00024  *      of "int".  NAMD uses "int32" explicitly, which is required on
00025  *      machines like the T3E.  VMD's version of the code doesn't do that
00026  *      presently.
00027  *
00028  *  Try various alternative I/O API options:
00029  *   - use mmap(), with read-once flags
00030  *   - use O_DIRECT open mode on new revs of Linux kernel
00031  *   - use directio() call on a file descriptor to enable on Solaris
00032  *   - use aio_open()/read()/write()
00033  *   - use readv/writev() etc.
00034  *
00035  *  Standalone test binary compilation flags:
00036  *  cc -fast -xarch=v9a -I../../include -DTEST_DCDPLUGIN dcdplugin.c \
00037  *    -o ~/bin/readdcd -lm
00038  *
00039  *  Profiling flags:
00040  *  cc -xpg -fast -xarch=v9a -g -I../../include -DTEST_DCDPLUGIN dcdplugin.c \
00041  *    -o ~/bin/readdcd -lm
00042  *
00043  ***************************************************************************/
00044
00045 #include "largefiles.h"   /* platform dependent 64-bit file I/O defines */
00046
00047 #include <stdio.h>
00048 #include <sys/stat.h>
00049 #include <sys/types.h>
00050 #include <stdlib.h>
00051 #include <string.h>
00052 #include <math.h>
00053 #include <time.h>
00054 #include "endianswap.h"
00055 #include "fastio.h"
00056 #include "molfile_plugin.h"
00057
00058 #ifndef M_PI_2
00059 #define M_PI_2 1.57079632679489661922
00060 #endif
00061
00062 #define RECSCALE32BIT 1
00063 #define RECSCALE64BIT 2
00064 #define RECSCALEMAX   2
00065
00066 typedef struct {
00067   fio_fd fd;
00068   int natoms;
00069   int nsets;
00070   int setsread;
00071   int istart;
00072   int nsavc;
00073   double delta;
00074   int nfixed;
00075   float *x, *y, *z;
00076   int *freeind;
00077   float *fixedcoords;
00078   int reverse;
00079   int charmm;
00080   int first;
00081   int with_unitcell;
00082 } dcdhandle;
00083
00084 /* Define error codes that may be returned by the DCD routines */
00085 #define DCD_SUCCESS      0  /* No problems                       */
00086 #define DCD_EOF         -1  /* Normal EOF                        */
00087 #define DCD_DNE         -2  /* DCD file does not exist           */
00088 #define DCD_OPENFAILED  -3  /* Open of DCD file failed           */
```

```
00089 #define DCD_BADREAD     -4  /* read call on DCD file failed    */
00090 #define DCD_BADEOF      -5  /* premature EOF found in DCD file */
00091 #define DCD_BADFORMAT   -6  /* format of DCD file is wrong     */
00092 #define DCD_FILEEXISTS  -7  /* output file already exists      */
00093 #define DCD_BADMALLOC   -8  /* malloc failed                   */
00094 #define DCD_BADWRITE    -9  /* write call on DCD file failed   */
00095
00096 /* Define feature flags for this DCD file */
00097 #define DCD_IS_XPLOR        0x00
00098 #define DCD_IS_CHARMM       0x01
00099 #define DCD_HAS_4DIMS       0x02
00100 #define DCD_HAS_EXTRA_BLOCK 0x04
00101 #define DCD_HAS_64BIT_REC   0x08
00102
00103 /* defines used by write_dcdstep */
00104 #define NFILE_POS 8L
00105 #define NSTEP_POS 20L
00106
00107 /* READ Macro to make porting easier */
00108 #define READ(fd, buf, size)  fio_fread(((void *) buf), (size), 1, (fd))
00109
00110 /* WRITE Macro to make porting easier */
00111 #define WRITE(fd, buf, size) fio_fwrite(((void *) buf), (size), 1, (fd))
00112
00113 /* XXX This is broken - fread never returns -1 */
00114 #define CHECK_FREAD(X, msg) if (X==-1) { return(DCD_BADREAD); }
00115 #define CHECK_FEOF(X, msg)  if (X==0)  { return(DCD_BADEOF); }
00116
00117
00118 /* print DCD error in a human readable way */
00119 static void print_dcderror(const char *func, int errcode) {
00120   const char *errstr;
00121
00122   switch (errcode) {
00123     case DCD_EOF:         errstr = "end of file"; break;
00124     case DCD_DNE:         errstr = "file not found"; break;
00125     case DCD_OPENFAILED:  errstr = "file open failed"; break;
00126     case DCD_BADREAD:     errstr = "error during read"; break;
00127     case DCD_BADEOF:      errstr = "premature end of file"; break;
00128     case DCD_BADFORMAT:   errstr = "corruption or unrecognized file structure"; break;
00129     case DCD_FILEEXISTS:  errstr = "output file already exists"; break;
00130     case DCD_BADMALLOC:   errstr = "memory allocation failed"; break;
00131     case DCD_BADWRITE:    errstr = "error during write"; break;
00132     case DCD_SUCCESS:
00133     default:
00134       errstr = "no error";
00135       break;
00136   }
00137   printf("dcdplugin) %s: %s\n", func, errstr);
00138 }
00139
00140
00141 /*
00142  * Read the header information from a dcd file.
00143  * Input: fd - a file struct opened for binary reading.
00144  * Output: 0 on success, negative error code on failure.
00145  * Side effects: *natoms set to number of atoms per frame
00146  *               *nsets set to number of frames in dcd file
00147  *               *istart set to starting timestep of dcd file
00148  *               *nsavc set to timesteps between dcd saves
00149  *               *delta set to value of trajectory timestep
00150  *               *nfixed set to number of fixed atoms
00151  *               *freeind may be set to heap-allocated space
00152  *               *reverse set to one if reverse-endian, zero if not.
00153  *               *charmm set to internal code for handling charmm data.
00154  */
00155 static int read_dcdheader(fio_fd fd, int *N, int *NSET, int *ISTART,
00156                   int *NSAVC, double *DELTA, int *NAMNF,
00157                   int **FREEINDEXES, float **fixedcoords, int *reverseEndian,
00158                   int *charmm)
00159 {
00160   unsigned int input_integer[2];  /* buffer space */
00161   int i, ret_val, rec_scale;
00162   char hdrbuf[84];    /* char buffer used to store header */
00163   int NTITLE;
00164   int dcdcordmagic;
00165   char *corp = (char *) &dcdcordmagic;
00166
00167   /* coordinate dcd file magic string 'CORD' */
00168   corp[0] = 'C';
00169   corp[1] = 'O';
00170   corp[2] = 'R';
00171   corp[3] = 'D';
00172
00173   /* First thing in the file should be an 84.
00174    * some 64-bit compiles have a 64-bit record length indicator,
00175    * so we have to read two ints and check in a more complicated
00176    * way. :-( */
00177   ret_val = READ(fd, input_integer, 2*sizeof(unsigned int));
00178   CHECK_FREAD(ret_val, "reading first int from dcd file");
00179   CHECK_FEOF(ret_val, "reading first int from dcd file");
00180
00181   /* Check magic number in file header and determine byte order*/
00182   if ((input_integer[0]+input_integer[1]) == 84) {
00183     *reverseEndian=0;
00184     rec_scale=RECSCALE64BIT;
00185     printf("dcdplugin) detected CHARMM -i8 64-bit DCD file of native endianness\n");
```

```
00186   } else if (input_integer[0] == 84 && input_integer[1] == dcdcordmagic) {
00187     *reverseEndian=0;
00188     rec_scale=RECSCALE32BIT;
00189     printf("dcdplugin) detected standard 32-bit DCD file of native endianness\n");
00190   } else {
00191     /* now try reverse endian */
00192     swap4_aligned(input_integer, 2); /* will have to unswap magic if 32-bit */
00193     if ((input_integer[0]+input_integer[1]) == 84) {
00194       *reverseEndian=1;
00195       rec_scale=RECSCALE64BIT;
00196       printf("dcdplugin) detected CHARMM -i8 64-bit DCD file of opposite endianness\n");
00197     } else {
00198       swap4_aligned(&input_integer[1], 1); /* unswap magic (see above) */
00199       if (input_integer[0] == 84 && input_integer[1] == dcdcordmagic) {
00200         *reverseEndian=1;
00201         rec_scale=RECSCALE32BIT;
00202         printf("dcdplugin) detected standard 32-bit DCD file of opposite endianness\n");
00203       } else {
00204         /* not simply reversed endianism or -i8, something rather more evil */
00205         printf("dcdplugin) unrecognized DCD header:\n");
00206         printf("dcdplugin)   [0]: %10d  [1]: %10d\n", input_integer[0], input_integer[1]);
00207         printf("dcdplugin)   [0]: 0x%08x  [1]: 0x%08x\n", input_integer[0], input_integer[1]);
00208         return DCD_BADFORMAT;
00209 
00210       }
00211     }
00212   }
00213 
00214   /* check for magic string, in case of long record markers */
00215   if (rec_scale == RECSCALE64BIT) {
00216     ret_val = READ(fd, input_integer, sizeof(unsigned int));
00217     if (input_integer[0] != dcdcordmagic) {
00218       printf("dcdplugin) failed to find CORD magic in CHARMM -i8 64-bit DCD file\n");
00219       return DCD_BADFORMAT;
00220     }
00221   }
00222 
00223   /* Buffer the entire header for random access */
00224   ret_val = READ(fd, hdrbuf, 80);
00225   CHECK_FREAD(ret_val, "buffering header");
00226   CHECK_FEOF(ret_val, "buffering header");
00227 
00228   /* CHARMm-genereate DCD files set the last integer in the     */
00229   /* header, which is unused by X-PLOR, to its version number.  */
00230   /* Checking if this is nonzero tells us this is a CHARMm file */
00231   /* and to look for other CHARMm flags.                        */
00232   if (*((int *) (hdrbuf + 76)) != 0) {
00233     (*charmm) = DCD_IS_CHARMM;
00234     if (*((int *) (hdrbuf + 40)) != 0)
00235       (*charmm) |= DCD_HAS_EXTRA_BLOCK;
00236 
00237     if (*((int *) (hdrbuf + 44)) == 1)
00238       (*charmm) |= DCD_HAS_4DIMS;
00239 
00240     if (rec_scale == RECSCALE64BIT)
00241       (*charmm) |= DCD_HAS_64BIT_REC;
00242 
00243   } else {
00244     (*charmm) = DCD_IS_XPLOR; /* must be an X-PLOR format DCD file */
00245   }
00246 
00247   if (*charmm & DCD_IS_CHARMM) {
00248     /* CHARMM and NAMD versions 2.1b1 and later */
00249     printf("dcdplugin) CHARMM format DCD file (also NAMD 2.1 and later)\n");
00250   } else {
00251     /* CHARMM and NAMD versions prior to 2.1b1  */
00252     printf("dcdplugin) X-PLOR format DCD file (also NAMD 2.0 and earlier)\n");
00253   }
00254 
00255   /* Store the number of sets of coordinates (NSET) */
00256   (*NSET) = *((int *) (hdrbuf));
00257   if (*reverseEndian) swap4_unaligned(NSET, 1);
00258 
00259   /* Store ISTART, the starting timestep */
00260   (*ISTART) = *((int *) (hdrbuf + 4));
00261   if (*reverseEndian) swap4_unaligned(ISTART, 1);
00262 
00263   /* Store NSAVC, the number of timesteps between dcd saves */
00264   (*NSAVC) = *((int *) (hdrbuf + 8));
00265   if (*reverseEndian) swap4_unaligned(NSAVC, 1);
00266 
00267   /* Store NAMNF, the number of fixed atoms */
00268   (*NAMNF) = *((int *) (hdrbuf + 32));
00269   if (*reverseEndian) swap4_unaligned(NAMNF, 1);
00270 
00271   /* Read in the timestep, DELTA */
00272   /* Note: DELTA is stored as a double with X-PLOR but as a float with CHARMm */
00273   if ((*charmm) & DCD_IS_CHARMM) {
00274     float ftmp;
00275     ftmp = *((float *)(hdrbuf+36)); /* is this safe on Alpha? */
00276     if (*reverseEndian)
00277       swap4_aligned(&ftmp, 1);
00278 
00279     *DELTA = (double)ftmp;
00280   } else {
00281     (*DELTA) = *((double *)(hdrbuf + 36));
00282     if (*reverseEndian) swap8_unaligned(DELTA, 1);
```

```
00283   }
00284
00285   /* Get the end size of the first block */
00286   ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00287   CHECK_FREAD(ret_val, "reading second 84 from dcd file");
00288   CHECK_FEOF(ret_val, "reading second 84 from dcd file");
00289   if (*reverseEndian) swap4_aligned(input_integer, rec_scale);
00290
00291   if (rec_scale == RECSCALE64BIT) {
00292     if ((input_integer[0]+input_integer[1]) != 84) {
00293       return DCD_BADFORMAT;
00294     }
00295   } else {
00296     if (input_integer[0] != 84) {
00297       return DCD_BADFORMAT;
00298     }
00299   }
00300
00301   /* Read in the size of the next block */
00302   input_integer[1] = 0;
00303   ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00304   CHECK_FREAD(ret_val, "reading size of title block");
00305   CHECK_FEOF(ret_val, "reading size of title block");
00306   if (*reverseEndian) swap4_aligned(input_integer, rec_scale);
00307
00308   if ((((input_integer[0]+input_integer[1])-4) % 80) == 0) {
00309     /* Read NTITLE, the number of 80 character title strings there are */
00310     ret_val = READ(fd, &NTITLE, sizeof(int));
00311     CHECK_FREAD(ret_val, "reading NTITLE");
00312     CHECK_FEOF(ret_val, "reading NTITLE");
00313     if (*reverseEndian) swap4_aligned(&NTITLE, 1);
00314
00315     for (i=0; i<NTITLE; i++) {
00316       fio_fseek(fd, 80, FIO_SEEK_CUR);
00317       CHECK_FEOF(ret_val, "reading TITLE");
00318     }
00319
00320     /* Get the ending size for this block */
00321     ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00322     CHECK_FREAD(ret_val, "reading size of title block");
00323     CHECK_FEOF(ret_val, "reading size of title block");
00324   } else {
00325     return DCD_BADFORMAT;
00326   }
00327
00328   /* Read in an integer '4' */
00329   input_integer[1] = 0;
00330   ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00331
00332   CHECK_FREAD(ret_val, "reading a '4'");
00333   CHECK_FEOF(ret_val, "reading a '4'");
00334   if (*reverseEndian) swap4_aligned(input_integer, rec_scale);
00335
00336   if ((input_integer[0]+input_integer[1]) != 4) {
00337     return DCD_BADFORMAT;
00338   }
00339
00340   /* Read in the number of atoms */
00341   ret_val = READ(fd, N, sizeof(int));
00342   CHECK_FREAD(ret_val, "reading number of atoms");
00343   CHECK_FEOF(ret_val, "reading number of atoms");
00344   if (*reverseEndian) swap4_aligned(N, 1);
00345
00346   /* Read in an integer '4' */
00347   input_integer[1] = 0;
00348   ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00349   CHECK_FREAD(ret_val, "reading a '4'");
00350   CHECK_FEOF(ret_val, "reading a '4'");
00351   if (*reverseEndian) swap4_aligned(input_integer, rec_scale);
00352
00353   if ((input_integer[0]+input_integer[1]) != 4) {
00354     return DCD_BADFORMAT;
00355   }
00356
00357   *FREEINDEXES = NULL;
00358   *fixedcoords = NULL;
00359   if (*NAMNF != 0) {
00360     (*FREEINDEXES) = (int *) calloc(((*N)-(*NAMNF)), sizeof(int));
00361     if (*FREEINDEXES == NULL)
00362       return DCD_BADMALLOC;
00363
00364     *fixedcoords = (float *) calloc((*N)*4 - (*NAMNF), sizeof(float));
00365     if (*fixedcoords == NULL)
00366       return DCD_BADMALLOC;
00367
00368     /* Read in index array size */
00369     input_integer[1]=0;
00370     ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00371     CHECK_FREAD(ret_val, "reading size of index array");
00372     CHECK_FEOF(ret_val, "reading size of index array");
00373     if (*reverseEndian) swap4_aligned(input_integer, rec_scale);
00374
00375     if ((input_integer[0]+input_integer[1]) != ((*N)-(*NAMNF))*4) {
00376       return DCD_BADFORMAT;
00377     }
00378
00379     ret_val = READ(fd, (*FREEINDEXES), ((*N)-(*NAMNF))*sizeof(int));
```

```
00380     CHECK_FREAD(ret_val, "reading size of index array");
00381     CHECK_FEOF(ret_val, "reading size of index array");
00382 
00383     if (*reverseEndian)
00384       swap4_aligned((*FREEINDEXES), ((*N)-(*NAMNF)));
00385 
00386     input_integer[1]=0;
00387     ret_val = READ(fd, input_integer, rec_scale*sizeof(int));
00388     CHECK_FREAD(ret_val, "reading size of index array");
00389     CHECK_FEOF(ret_val, "reading size of index array");
00390     if (*reverseEndian) swap4_aligned(input_integer, rec_scale);
00391 
00392     if ((input_integer[0]+input_integer[1]) != ((*N)-(*NAMNF))*4) {
00393       return DCD_BADFORMAT;
00394     }
00395   }
00396 
00397   return DCD_SUCCESS;
00398 }
00399 
00400 static int read_charmm_extrablock(fio_fd fd, int charmm, int reverseEndian,
00401                                   float *unitcell) {
00402   int i, input_integer[2], rec_scale;
00403 
00404   if (charmm & DCD_HAS_64BIT_REC) {
00405     rec_scale = RECSCALE64BIT;
00406   } else {
00407     rec_scale = RECSCALE32BIT;
00408   }
00409 
00410   if ((charmm & DCD_IS_CHARMM) && (charmm & DCD_HAS_EXTRA_BLOCK)) {
00411     /* Leading integer must be 48 */
00412     input_integer[1] = 0;
00413     if (fio_fread(input_integer, sizeof(int), rec_scale, fd) != rec_scale)
00414       return DCD_BADREAD;
00415     if (reverseEndian) swap4_aligned(input_integer, rec_scale);
00416     if ((input_integer[0]+input_integer[1]) == 48) {
00417       double tmp[6];
00418       if (fio_fread(tmp, 48, 1, fd) != 1) return DCD_BADREAD;
00419       if (reverseEndian)
00420         swap8_aligned(tmp, 6);
00421       for (i=0; i<6; i++) unitcell[i] = (float)tmp[i];
00422     } else {
00423       /* unrecognized block, just skip it */
00424       if (fio_fseek(fd, (input_integer[0]+input_integer[1]), FIO_SEEK_CUR)) return DCD_BADREAD;
00425     }
00426     if (fio_fread(input_integer, sizeof(int), rec_scale, fd) != rec_scale) return DCD_BADREAD;
00427   }
00428 
00429   return DCD_SUCCESS;
00430 }
00431 
00432 static int read_fixed_atoms(fio_fd fd, int N, int num_free, const int *indexes,
00433                             int reverseEndian, const float *fixedcoords,
00434                             float *freeatoms, float *pos, int charmm) {
00435   int i, input_integer[2], rec_scale;
00436 
00437   if(charmm & DCD_HAS_64BIT_REC) {
00438     rec_scale=RECSCALE64BIT;
00439   } else {
00440     rec_scale=RECSCALE32BIT;
00441   }
00442 
00443   /* Read leading integer */
00444   input_integer[1]=0;
00445   if (fio_fread(input_integer, sizeof(int), rec_scale, fd) != rec_scale) return DCD_BADREAD;
00446   if (reverseEndian) swap4_aligned(input_integer, rec_scale);
00447   if ((input_integer[0]+input_integer[1]) != 4*num_free) return DCD_BADFORMAT;
00448 
00449   /* Read free atom coordinates */
00450   if (fio_fread(freeatoms, 4*num_free, 1, fd) != 1) return DCD_BADREAD;
00451   if (reverseEndian)
00452     swap4_aligned(freeatoms, num_free);
00453 
00454   /* Copy fixed and free atom coordinates into position buffer */
00455   memcpy(pos, fixedcoords, 4*N);
00456   for (i=0; i<num_free; i++)
00457     pos[indexes[i]-1] = freeatoms[i];
00458 
00459   /* Read trailing integer */
00460   input_integer[1]=0;
00461   if (fio_fread(input_integer, sizeof(int), rec_scale, fd) != rec_scale) return DCD_BADREAD;
00462   if (reverseEndian) swap4_aligned(input_integer, rec_scale);
00463   if ((input_integer[0]+input_integer[1]) != 4*num_free) return DCD_BADFORMAT;
00464 
00465   return DCD_SUCCESS;
00466 }
00467 
00468 static int read_charmm_4dim(fio_fd fd, int charmm, int reverseEndian) {
00469   int input_integer[2],rec_scale;
00470 
00471   if (charmm & DCD_HAS_64BIT_REC) {
00472     rec_scale=RECSCALE64BIT;
00473   } else {
00474     rec_scale=RECSCALE32BIT;
00475   }
00476 
```

```
00477   /* If this is a CHARMm file and contains a 4th dimension block, */
00478   /* we must skip past it to avoid problems                       */
00479   if ((charmm & DCD_IS_CHARMM) && (charmm & DCD_HAS_4DIMS)) {
00480     input_integer[1]=0;
00481     if (fio_fread(input_integer, sizeof(int), rec_scale, fd) != rec_scale) return DCD_BADREAD;
00482     if (reverseEndian) swap4_aligned(input_integer, rec_scale);
00483     if (fio_fseek(fd, (input_integer[0]+input_integer[1]), FIO_SEEK_CUR)) return DCD_BADREAD;
00484     if (fio_fread(input_integer, sizeof(int), rec_scale, fd) != rec_scale) return DCD_BADREAD;
00485   }
00486
00487   return DCD_SUCCESS;
00488 }
00489
00490 /*
00491  * Read a dcd timestep from a dcd file
00492  * Input: fd - a file struct opened for binary reading, from which the
00493  *             header information has already been read.
00494  *        natoms, nfixed, first, *freeind, reverse, charmm - the corresponding
00495  *             items as set by read_dcdheader
00496  *        first - true if this is the first frame we are reading.
00497  *        x, y, z: space for natoms each of floats.
00498  *        unitcell - space for six floats to hold the unit cell data.
00499  *                   Not set if no unit cell data is present.
00500  * Output: 0 on success, negative error code on failure.
00501  * Side effects: x, y, z contain the coordinates for the timestep read.
00502  *               unitcell holds unit cell data if present.
00503  */
00504 static int read_dcdstep(fio_fd fd, int N, float *X, float *Y, float *Z,
00505                         float *unitcell, int num_fixed,
00506                         int first, int *indexes, float *fixedcoords,
00507                         int reverseEndian, int charmm) {
00508   int ret_val, rec_scale;   /* Return value from read */
00509
00510   if (charmm & DCD_HAS_64BIT_REC) {
00511     rec_scale=RECSCALE64BIT;
00512   } else {
00513     rec_scale=RECSCALE32BIT;
00514   }
00515
00516   if ((num_fixed==0) || first) {
00517     /* temp storage for reading formatting info */
00518     /* note: has to be max size we'll ever use  */
00519     int tmpbuf[6*RECSCALEMAX];
00520
00521     fio_iovec iov[7];   /* I/O vector for fio_readv() call          */
00522     fio_size_t readlen; /* number of bytes actually read            */
00523     int i;
00524
00525     /* if there are no fixed atoms or this is the first timestep read */
00526     /* then we read all coordinates normally.                         */
00527
00528     /* read the charmm periodic cell information */
00529     /* XXX this too should be read together with the other items in a */
00530     /*     single fio_readv() call in order to prevent lots of extra  */
00531     /*     kernel/user context switches.                              */
00532     ret_val = read_charmm_extrablock(fd, charmm, reverseEndian, unitcell);
00533     if (ret_val) return ret_val;
00534
00535     /* setup the I/O vector for the call to fio_readv() */
00536     iov[0].iov_base = (fio_caddr_t) &tmpbuf[0]; /* read format integer    */
00537     iov[0].iov_len  = rec_scale*sizeof(int);
00538
00539     iov[1].iov_base = (fio_caddr_t) X;          /* read X coordinates     */
00540     iov[1].iov_len  = sizeof(float)*N;
00541
00542     iov[2].iov_base = (fio_caddr_t) &tmpbuf[1*rec_scale]; /* read 2 format integers */
00543     iov[2].iov_len  = rec_scale*sizeof(int) * 2;
00544
00545     iov[3].iov_base = (fio_caddr_t) Y;          /* read Y coordinates     */
00546     iov[3].iov_len  = sizeof(float)*N;
00547
00548     iov[4].iov_base = (fio_caddr_t) &tmpbuf[3*rec_scale]; /* read 2 format integers */
00549     iov[4].iov_len  = rec_scale*sizeof(int) * 2;
00550
00551     iov[5].iov_base = (fio_caddr_t) Z;          /* read Y coordinates     */
00552     iov[5].iov_len  = sizeof(float)*N;
00553
00554     iov[6].iov_base = (fio_caddr_t) &tmpbuf[5*rec_scale]; /* read format integer    */
00555     iov[6].iov_len  = rec_scale*sizeof(int);
00556
00557     readlen = fio_readv(fd, &iov[0], 7);
00558
00559     if (readlen != (rec_scale*6*sizeof(int) + 3*N*sizeof(float)))
00560       return DCD_BADREAD;
00561
00562     /* convert endianism if necessary */
00563     if (reverseEndian) {
00564       swap4_aligned(&tmpbuf[0], rec_scale*6);
00565       swap4_aligned(X, N);
00566       swap4_aligned(Y, N);
00567       swap4_aligned(Z, N);
00568     }
00569
00570     /* double-check the fortran format size values for safety */
00571     if(rec_scale == 1) {
00572       for (i=0; i<6; i++) {
00573         if (tmpbuf[i] != sizeof(float)*N) return DCD_BADFORMAT;
```

```
00574       }
00575     } else {
00576       for (i=0; i<6; i++) {
00577           if ((tmpbuf[2*i]+tmpbuf[2*i+1]) != sizeof(float)*N) return DCD_BADFORMAT;
00578       }
00579     }
00580 
00581     /* copy fixed atom coordinates into fixedcoords array if this was the */
00582     /* first timestep, to be used from now on.  We just copy all atoms.   */
00583     if (num_fixed && first) {
00584       memcpy(fixedcoords, X, N*sizeof(float));
00585       memcpy(fixedcoords+N, Y, N*sizeof(float));
00586       memcpy(fixedcoords+2*N, Z, N*sizeof(float));
00587     }
00588 
00589     /* read in the optional charmm 4th array */
00590     /* XXX this too should be read together with the other items in a */
00591     /*     single fio_readv() call in order to prevent lots of extra  */
00592     /*     kernel/user context switches.                              */
00593     ret_val = read_charmm_4dim(fd, charmm, reverseEndian);
00594     if (ret_val) return ret_val;
00595   } else {
00596     /* if there are fixed atoms, and this isn't the first frame, then we */
00597     /* only read in the non-fixed atoms for all subsequent timesteps.    */
00598     ret_val = read_charmm_extrablock(fd, charmm, reverseEndian, unitcell);
00599     if (ret_val) return ret_val;
00600     ret_val = read_fixed_atoms(fd, N, N-num_fixed, indexes, reverseEndian,
00601                                fixedcoords, fixedcoords+3*N, X, charmm);
00602     if (ret_val) return ret_val;
00603     ret_val = read_fixed_atoms(fd, N, N-num_fixed, indexes, reverseEndian,
00604                                fixedcoords+N, fixedcoords+3*N, Y, charmm);
00605     if (ret_val) return ret_val;
00606     ret_val = read_fixed_atoms(fd, N, N-num_fixed, indexes, reverseEndian,
00607                                fixedcoords+2*N, fixedcoords+3*N, Z, charmm);
00608     if (ret_val) return ret_val;
00609     ret_val = read_charmm_4dim(fd, charmm, reverseEndian);
00610     if (ret_val) return ret_val;
00611   }
00612 
00613   return DCD_SUCCESS;
00614 }
00615 
00616 
00617 /*
00618  * Skip past a timestep.  If there are fixed atoms, this cannot be used with
00619  * the first timestep.
00620  * Input: fd - a file struct from which the header has already been read
00621  *        natoms - number of atoms per timestep
00622  *        nfixed - number of fixed atoms
00623  *        charmm - charmm flags as returned by read_dcdheader
00624  * Output: 0 on success, negative error code on failure.
00625  * Side effects: One timestep will be skipped; fd will be positioned at the
00626  *               next timestep.
00627  */
00628 static int skip_dcdstep(fio_fd fd, int natoms, int nfixed, int charmm) {
00629 
00630   int seekoffset = 0;
00631   int rec_scale;
00632 
00633   if (charmm & DCD_HAS_64BIT_REC) {
00634     rec_scale=RECSCALE64BIT;
00635   } else {
00636     rec_scale=RECSCALE32BIT;
00637   }
00638 
00639   /* Skip charmm extra block */
00640   if ((charmm & DCD_IS_CHARMM) && (charmm & DCD_HAS_EXTRA_BLOCK)) {
00641     seekoffset += 4*rec_scale + 48 + 4*rec_scale;
00642   }
00643 
00644   /* For each atom set, seek past an int, the free atoms, and another int. */
00645   seekoffset += 3 * (2*rec_scale + natoms - nfixed) * 4;
00646 
00647   /* Assume that charmm 4th dim is the same size as the other three. */
00648   if ((charmm & DCD_IS_CHARMM) && (charmm & DCD_HAS_4DIMS)) {
00649     seekoffset += (2*rec_scale + natoms - nfixed) * 4;
00650   }
00651 
00652   if (fio_fseek(fd, seekoffset, FIO_SEEK_CUR)) return DCD_BADEOF;
00653 
00654   return DCD_SUCCESS;
00655 }
00656 
00657 
00658 /*
00659  * Write a timestep to a dcd file
00660  * Input: fd - a file struct for which a dcd header has already been written
00661  *       curframe: Count of frames written to this file, starting with 1.
00662  *       curstep: Count of timesteps elapsed = istart + curframe * nsavc.
00663  *        natoms - number of elements in x, y, z arrays
00664  *        x, y, z: pointers to atom coordinates
00665  * Output: 0 on success, negative error code on failure.
00666  * Side effects: coordinates are written to the dcd file.
00667  */
00668 static int write_dcdstep(fio_fd fd, int curframe, int curstep, int N,
00669                          const float *X, const float *Y, const float *Z,
00670                          const double *unitcell, int charmm) {
```

```
00671   int out_integer;
00672
00673   if (charmm) {
00674     /* write out optional unit cell */
00675     if (unitcell != NULL) {
00676       out_integer = 48; /* 48 bytes (6 doubles) */
00677       fio_write_int32(fd, out_integer);
00678       WRITE(fd, unitcell, out_integer);
00679       fio_write_int32(fd, out_integer);
00680     }
00681   }
00682
00683   /* write out coordinates */
00684   out_integer = N*4; /* N*4 bytes per X/Y/Z array (N floats per array) */
00685   fio_write_int32(fd, out_integer);
00686   if (fio_fwrite((void *) X, out_integer, 1, fd) != 1) return DCD_BADWRITE;
00687   fio_write_int32(fd, out_integer);
00688   fio_write_int32(fd, out_integer);
00689   if (fio_fwrite((void *) Y, out_integer, 1, fd) != 1) return DCD_BADWRITE;
00690   fio_write_int32(fd, out_integer);
00691   fio_write_int32(fd, out_integer);
00692   if (fio_fwrite((void *) Z, out_integer, 1, fd) != 1) return DCD_BADWRITE;
00693   fio_write_int32(fd, out_integer);
00694
00695   /* update the DCD header information */
00696   fio_fseek(fd, NFILE_POS, FIO_SEEK_SET);
00697   fio_write_int32(fd, curframe);
00698   fio_fseek(fd, NSTEP_POS, FIO_SEEK_SET);
00699   fio_write_int32(fd, curstep);
00700   fio_fseek(fd, 0, FIO_SEEK_END);
00701
00702   return DCD_SUCCESS;
00703 }
00704
00705 /*
00706  * Write a header for a new dcd file
00707  * Input: fd - file struct opened for binary writing
00708  *        remarks - string to be put in the remarks section of the header.
00709  *                  The string will be truncated to 70 characters.
00710  *        natoms, istart, nsavc, delta - see comments in read_dcdheader
00711  * Output: 0 on success, negative error code on failure.
00712  * Side effects: Header information is written to the dcd file.
00713  */
00714 static int write_dcdheader(fio_fd fd, const char *remarks, int N,
00715                            int ISTART, int NSAVC, double DELTA, int with_unitcell,
00716                            int charmm) {
00717   int out_integer;
00718   float out_float;
00719   char title_string[200];
00720   time_t cur_time;
00721   struct tm *tmbuf;
00722   char time_str[81];
00723
00724   out_integer = 84;
00725   WRITE(fd, (char *) & out_integer, sizeof(int));
00726   strcpy(title_string, "CORD");
00727   WRITE(fd, title_string, 4);
00728   fio_write_int32(fd, 0);      /* Number of frames in file, none written yet   */
00729   fio_write_int32(fd, ISTART); /* Starting timestep                            */
00730   fio_write_int32(fd, NSAVC);  /* Timesteps between frames written to the file */
00731   fio_write_int32(fd, 0);      /* Number of timesteps in simulation            */
00732   fio_write_int32(fd, 0);      /* NAMD writes NSTEP or ISTART - NSAVC here?    */
00733   fio_write_int32(fd, 0);
00734   fio_write_int32(fd, 0);
00735   fio_write_int32(fd, 0);
00736   fio_write_int32(fd, 0);
00737   if (charmm) {
00738     out_float = DELTA;
00739     WRITE(fd, (char *) &out_float, sizeof(float));
00740     if (with_unitcell) {
00741       fio_write_int32(fd, 1);
00742     } else {
00743       fio_write_int32(fd, 0);
00744     }
00745   } else {
00746     WRITE(fd, (char *) &DELTA, sizeof(double));
00747   }
00748   fio_write_int32(fd, 0);
00749   fio_write_int32(fd, 0);
00750   fio_write_int32(fd, 0);
00751   fio_write_int32(fd, 0);
00752   fio_write_int32(fd, 0);
00753   fio_write_int32(fd, 0);
00754   fio_write_int32(fd, 0);
00755   fio_write_int32(fd, 0);
00756   if (charmm) {
00757     fio_write_int32(fd, 24); /* Pretend to be CHARMM version 24 */
00758   } else {
00759     fio_write_int32(fd, 0);
00760   }
00761   fio_write_int32(fd, 84);
00762   fio_write_int32(fd, 164);
00763   fio_write_int32(fd, 2);
00764
00765   strncpy(title_string, remarks, 80);
00766   title_string[79] = '\0';
00767   WRITE(fd, title_string, 80);
```

```
00768
00769   cur_time=time(NULL);
00770   tmbuf=localtime(&cur_time);
00771   strftime(time_str, 80, "REMARKS Created %d %B, %Y at %R", tmbuf);
00772   WRITE(fd, time_str, 80);
00773
00774   fio_write_int32(fd, 164);
00775   fio_write_int32(fd, 4);
00776   fio_write_int32(fd, N);
00777   fio_write_int32(fd, 4);
00778
00779   return DCD_SUCCESS;
00780 }
00781
00782
00783 /*
00784  * clean up dcd data
00785  * Input: nfixed, freeind - elements as returned by read_dcdheader
00786  * Output: None
00787  * Side effects: Space pointed to by freeind is freed if necessary.
00788  */
00789 static void close_dcd_read(int *indexes, float *fixedcoords) {
00790   free(indexes);
00791   free(fixedcoords);
00792 }
00793
00794
00795
00796
00797
00798 static void *open_dcd_read(const char *path, const char *filetype,
00799     int *natoms) {
00800   dcdhandle *dcd;
00801   fio_fd fd;
00802   int rc;
00803   struct stat stbuf;
00804
00805   if (!path) return NULL;
00806
00807   /* See if the file exists, and get its size */
00808   memset(&stbuf, 0, sizeof(struct stat));
00809   if (stat(path, &stbuf)) {
00810     printf("dcdplugin) Could not access file '%s'.\n", path);
00811     return NULL;
00812   }
00813
00814   if (fio_open(path, FIO_READ, &fd) < 0) {
00815     printf("dcdplugin) Could not open file '%s' for reading.\n", path);
00816     return NULL;
00817   }
00818
00819   dcd = (dcdhandle *)malloc(sizeof(dcdhandle));
00820   memset(dcd, 0, sizeof(dcdhandle));
00821   dcd->fd = fd;
00822
00823   if ((rc = read_dcdheader(dcd->fd, &dcd->natoms, &dcd->nsets, &dcd->istart,
00824         &dcd->nsavc, &dcd->delta, &dcd->nfixed, &dcd->freeind,
00825         &dcd->fixedcoords, &dcd->reverse, &dcd->charmm))) {
00826     print_dcderror("read_dcdheader", rc);
00827     fio_fclose(dcd->fd);
00828     free(dcd);
00829     return NULL;
00830   }
00831
00832   /*
00833    * Check that the file is big enough to really hold the number of sets
00834    * it claims to have.  Then we'll use nsets to keep track of where EOF
00835    * should be.
00836    */
00837   {
00838     fio_size_t ndims, firstframesize, framesize, extrablocksize;
00839     fio_size_t trjsize, filesize, curpos;
00840     int newnsets;
00841
00842     extrablocksize = dcd->charmm & DCD_HAS_EXTRA_BLOCK ? 48 + 8 : 0;
00843     ndims = dcd->charmm & DCD_HAS_4DIMS ? 4 : 3;
00844     firstframesize = (dcd->natoms+2) * ndims * sizeof(float) + extrablocksize;
00845     framesize = (dcd->natoms-dcd->nfixed+2) * ndims * sizeof(float)
00846       + extrablocksize;
00847
00848     /*
00849      * It's safe to use ftell, even though ftell returns a long, because the
00850      * header size is < 4GB.
00851      */
00852
00853     curpos = fio_ftell(dcd->fd); /* save current offset (end of header) */
00854
00855 #if defined(_MSC_VER) && defined(FASTIO_NATIVEWIN32)
00856     /* the stat() call is not 64-bit savvy on Windows             */
00857     /* so we have to use the fastio fseek/ftell routines for this */
00858     /* until we add a portable filesize routine for this purpose  */
00859     fio_fseek(dcd->fd, 0, FIO_SEEK_END);       /* seek to end of file */
00860     filesize = fio_ftell(dcd->fd);
00861     fio_fseek(dcd->fd, curpos, FIO_SEEK_SET);  /* return to end of header */
00862 #else
00863     filesize = stbuf.st_size; /* this works ok on Unix machines */
00864 #endif
```

```
00865     trjsize = filesize - curpos - firstframesize;
00866     if (trjsize < 0) {
00867       printf("dcdplugin) file '%s' appears to contain no timesteps.\n", path);
00868       fio_fclose(dcd->fd);
00869       free(dcd);
00870       return NULL;
00871     }
00872 
00873     newnsets = trjsize / framesize + 1;
00874 
00875     if (dcd->nsets > 0 && newnsets != dcd->nsets) {
00876       printf("dcdplugin) Warning: DCD header claims %d frames, file size indicates there are actually %d frames\n", dcd->nsets, newnsets);
00877     }
00878 
00879     dcd->nsets = newnsets;
00880     dcd->setsread = 0;
00881   }
00882 
00883   dcd->first = 1;
00884   dcd->x = (float *)malloc(dcd->natoms * sizeof(float));
00885   dcd->y = (float *)malloc(dcd->natoms * sizeof(float));
00886   dcd->z = (float *)malloc(dcd->natoms * sizeof(float));
00887   if (!dcd->x || !dcd->y || !dcd->z) {
00888     printf("dcdplugin) Unable to allocate space for %d atoms.\n", dcd->natoms);
00889     if (dcd->x)
00890       free(dcd->x);
00891     if (dcd->y)
00892       free(dcd->y);
00893     if (dcd->z)
00894       free(dcd->z);
00895     fio_fclose(dcd->fd);
00896     free(dcd);
00897     return NULL;
00898   }
00899   *natoms = dcd->natoms;
00900   return dcd;
00901 }
00902 
00903 
00904 static int read_next_timestep(void *v, int natoms, molfile_timestep_t *ts) {
00905   dcdhandle *dcd;
00906   int i, j, rc;
00907   float unitcell[6];
00908   unitcell[0] = unitcell[2] = unitcell[5] = 1.0f;
00909   unitcell[1] = unitcell[3] = unitcell[4] = 90.0f;
00910   dcd = (dcdhandle *)v;
00911 
00912   /* Check for EOF here; that way all EOF's encountered later must be errors */
00913   if (dcd->setsread == dcd->nsets) return MOLFILE_EOF;
00914   dcd->setsread++;
00915   if (!ts) {
00916     if (dcd->first && dcd->nfixed) {
00917       /* We can't just skip it because we need the fixed atom coordinates */
00918       rc = read_dcdstep(dcd->fd, dcd->natoms, dcd->x, dcd->y, dcd->z,
00919           unitcell, dcd->nfixed, dcd->first, dcd->freeind, dcd->fixedcoords,
00920              dcd->reverse, dcd->charmm);
00921       dcd->first = 0;
00922       return rc; /* XXX this needs to be updated */
00923     }
00924     dcd->first = 0;
00925     /* XXX this needs to be changed */
00926     return skip_dcdstep(dcd->fd, dcd->natoms, dcd->nfixed, dcd->charmm);
00927   }
00928   rc = read_dcdstep(dcd->fd, dcd->natoms, dcd->x, dcd->y, dcd->z, unitcell,
00929             dcd->nfixed, dcd->first, dcd->freeind, dcd->fixedcoords,
00930             dcd->reverse, dcd->charmm);
00931   dcd->first = 0;
00932   if (rc < 0) {
00933     print_dcderror("read_dcdstep", rc);
00934     return MOLFILE_ERROR;
00935   }
00936 
00937   /* copy timestep data from plugin-local buffers to VMD's buffer */
00938   /* XXX
00939    *   This code is still the root of all evil.  Just doing this extra copy
00940    *   cuts the I/O rate of the DCD reader from 728 MB/sec down to
00941    *   394 MB/sec when reading from a ram filesystem.
00942    *   For a physical disk filesystem, the I/O rate goes from
00943    *   187 MB/sec down to 122 MB/sec.  Clearly this extra copy has to go.
00944    */
00945   {
00946     int natoms = dcd->natoms;
00947     float *nts = ts->coords;
00948     const float *bufx = dcd->x;
00949     const float *bufy = dcd->y;
00950     const float *bufz = dcd->z;
00951 
00952     for (i=0, j=0; i<natoms; i++, j+=3) {
00953       nts[j    ] = bufx[i];
00954       nts[j + 1] = bufy[i];
00955       nts[j + 2] = bufz[i];
00956     }
00957   }
00958 
00959   ts->A = unitcell[0];
00960   ts->B = unitcell[2];
00961   ts->C = unitcell[5];
```

```
00962
00963   if (unitcell[1] >= -1.0 && unitcell[1] <= 1.0 &&
00964       unitcell[3] >= -1.0 && unitcell[3] <= 1.0 &&
00965       unitcell[4] >= -1.0 && unitcell[4] <= 1.0) {
00966     /* This file was generated by CHARMM, or by NAMD > 2.5, with the angle */
00967     /* cosines of the periodic cell angles written to the DCD file.        */
00968     /* This formulation improves rounding behavior for orthogonal cells    */
00969     /* so that the angles end up at precisely 90 degrees, unlike acos().   */
00970     ts->alpha = 90.0 - asin(unitcell[4]) * 90.0 / M_PI_2; /* cosBC */
00971     ts->beta  = 90.0 - asin(unitcell[3]) * 90.0 / M_PI_2; /* cosAC */
00972     ts->gamma = 90.0 - asin(unitcell[1]) * 90.0 / M_PI_2; /* cosAB */
00973   } else {
00974     /* This file was likely generated by NAMD 2.5 and the periodic cell    */
00975     /* angles are specified in degrees rather than angle cosines.          */
00976     ts->alpha = unitcell[4]; /* angle between B and C */
00977     ts->beta  = unitcell[3]; /* angle between A and C */
00978     ts->gamma = unitcell[1]; /* angle between A and B */
00979   }
00980
00981   return MOLFILE_SUCCESS;
00982 }
00983
00984
00985 static void close_file_read(void *v) {
00986   dcdhandle *dcd = (dcdhandle *)v;
00987   close_dcd_read(dcd->freeind, dcd->fixedcoords);
00988   fio_fclose(dcd->fd);
00989   free(dcd->x);
00990   free(dcd->y);
00991   free(dcd->z);
00992   free(dcd);
00993 }
00994
00995
00996 static void *open_dcd_write(const char *path, const char *filetype,
00997     int natoms) {
00998   dcdhandle *dcd;
00999   fio_fd fd;
01000   int rc;
01001   int istart, nsavc;
01002   double delta;
01003   int with_unitcell;
01004   int charmm;
01005
01006   if (fio_open(path, FIO_WRITE, &fd) < 0) {
01007     printf("dcdplugin) Could not open file '%s' for writing\n", path);
01008     return NULL;
01009   }
01010
01011   dcd = (dcdhandle *)malloc(sizeof(dcdhandle));
01012   memset(dcd, 0, sizeof(dcdhandle));
01013   dcd->fd = fd;
01014
01015   istart = 0;             /* starting timestep of DCD file                  */
01016   nsavc = 1;              /* number of timesteps between written DCD frames */
01017   delta = 1.0;            /* length of a timestep                           */
01018
01019   if (getenv("VMDDCDWRITEXPLORFORMAT") != NULL) {
01020     with_unitcell = 0;      /* no unit cell info */
01021     charmm = DCD_IS_XPLOR;  /* X-PLOR format */
01022     printf("dcdplugin) WARNING: Writing DCD file in X-PLOR format, \n");
01023     printf("dcdplugin) WARNING: unit cell information will be lost!\n");
01024   } else {
01025     with_unitcell = 1;      /* contains unit cell infor (Charmm format) */
01026     charmm = DCD_IS_CHARMM; /* charmm-formatted DCD file                */
01027     if (with_unitcell)
01028       charmm |= DCD_HAS_EXTRA_BLOCK;
01029   }
01030
01031   rc = write_dcdheader(dcd->fd, "Created by DCD plugin", natoms,
01032                        istart, nsavc, delta, with_unitcell, charmm);
01033
01034   if (rc < 0) {
01035     print_dcderror("write_dcdheader", rc);
01036     fio_fclose(dcd->fd);
01037     free(dcd);
01038     return NULL;
01039   }
01040
01041   dcd->natoms = natoms;
01042   dcd->nsets = 0;
01043   dcd->istart = istart;
01044   dcd->nsavc = nsavc;
01045   dcd->with_unitcell = with_unitcell;
01046   dcd->charmm = charmm;
01047   dcd->x = (float *)malloc(natoms * sizeof(float));
01048   dcd->y = (float *)malloc(natoms * sizeof(float));
01049   dcd->z = (float *)malloc(natoms * sizeof(float));
01050   return dcd;
01051 }
01052
01053
01054 static int write_timestep(void *v, const molfile_timestep_t *ts) {
01055   dcdhandle *dcd = (dcdhandle *)v;
01056   int i, rc, curstep;
01057   float *pos = ts->coords;
01058   double unitcell[6];
```

```
01059   unitcell[0] = unitcell[2] = unitcell[5] = 1.0f;
01060   unitcell[1] = unitcell[3] = unitcell[4] = 90.0f;
01061
01062   /* copy atom coords into separate X/Y/Z arrays for writing */
01063   for (i=0; i<dcd->natoms; i++) {
01064     dcd->x[i] = *(pos++);
01065     dcd->y[i] = *(pos++);
01066     dcd->z[i] = *(pos++);
01067   }
01068   dcd->nsets++;
01069   curstep = dcd->istart + dcd->nsets * dcd->nsavc;
01070
01071   unitcell[0] = ts->A;
01072   unitcell[2] = ts->B;
01073   unitcell[5] = ts->C;
01074   unitcell[1] = sin((M_PI_2 / 90.0) * (90.0 - ts->gamma)); /* cosAB */
01075   unitcell[3] = sin((M_PI_2 / 90.0) * (90.0 - ts->beta));  /* cosAC */
01076   unitcell[4] = sin((M_PI_2 / 90.0) * (90.0 - ts->alpha)); /* cosBC */
01077
01078   rc = write_dcdstep(dcd->fd, dcd->nsets, curstep, dcd->natoms,
01079                      dcd->x, dcd->y, dcd->z,
01080                      dcd->with_unitcell ? unitcell : NULL,
01081                      dcd->charmm);
01082   if (rc < 0) {
01083     print_dcderror("write_dcdstep", rc);
01084     return MOLFILE_ERROR;
01085   }
01086
01087   return MOLFILE_SUCCESS;
01088 }
01089
01090 static void close_file_write(void *v) {
01091   dcdhandle *dcd = (dcdhandle *)v;
01092   fio_fclose(dcd->fd);
01093   free(dcd->x);
01094   free(dcd->y);
01095   free(dcd->z);
01096   free(dcd);
01097 }
01098
01099
01100 /*
01101  * Initialization stuff here
01102  */
01103 static molfile_plugin_t plugin;
01104
01105 VMDPLUGIN_API int VMDPLUGIN_init() {
01106   memset(&plugin, 0, sizeof(molfile_plugin_t));
01107   plugin.abiversion = vmdplugin_ABIVERSION;
01108   plugin.type = MOLFILE_PLUGIN_TYPE;
01109   plugin.name = "dcd";
01110   plugin.prettyname = "CHARMM,NAMD,XPLOR DCD Trajectory";
01111   plugin.author = "Justin Gullingsrud, John Stone";
01112   plugin.majorv = 1;
01113   plugin.minorv = 10;
01114   plugin.is_reentrant = VMDPLUGIN_THREADSAFE;
01115   plugin.filename_extension = "dcd";
01116   plugin.open_file_read = open_dcd_read;
01117   plugin.read_next_timestep = read_next_timestep;
01118   plugin.close_file_read = close_file_read;
01119   plugin.open_file_write = open_dcd_write;
01120   plugin.write_timestep = write_timestep;
01121   plugin.close_file_write = close_file_write;
01122   return VMDPLUGIN_SUCCESS;
01123 }
01124
01125 VMDPLUGIN_API int VMDPLUGIN_register(void *v, vmdplugin_register_cb cb) {
01126   (*cb)(v, (vmdplugin_t *)&plugin);
01127   return VMDPLUGIN_SUCCESS;
01128 }
01129
01130 VMDPLUGIN_API int VMDPLUGIN_fini() {
01131   return VMDPLUGIN_SUCCESS;
01132 }
01133
01134
01135 #ifdef TEST_DCDPLUGIN
01136
01137 #include <sys/time.h>
01138
01139 /* get the time of day from the system clock, and store it (in seconds) */
01140 double time_of_day(void) {
01141 #if defined(_MSC_VER)
01142   double t;
01143
01144   t = GetTickCount();
01145   t = t / 1000.0;
01146
01147   return t;
01148 #else
01149   struct timeval tm;
01150   struct timezone tz;
01151
01152   gettimeofday(&tm, &tz);
01153   return((double)(tm.tv_sec) + (double)(tm.tv_usec)/1000000.0);
01154 #endif
01155 }
```

```
01156
01157 int main(int argc, char *argv[]) {
01158   molfile_timestep_t timestep;
01159   void *v;
01160   dcdhandle *dcd;
01161   int i, natoms;
01162   float sizeMB =0.0, totalMB = 0.0;
01163   double starttime, endtime, totaltime = 0.0;
01164
01165   while (--argc) {
01166     ++argv;
01167     natoms = 0;
01168     v = open_dcd_read(*argv, "dcd", &natoms);
01169     if (!v) {
01170       fprintf(stderr, "main) open_dcd_read failed for file %s\n", *argv);
01171       return 1;
01172     }
01173     dcd = (dcdhandle *)v;
01174     sizeMB = ((natoms * 3.0) * dcd->nsets * 4.0) / (1024.0 * 1024.0);
01175     totalMB += sizeMB;
01176     printf("main) file: %s\n", *argv);
01177     printf("  %d atoms, %d frames, size: %6.1fMB\n", natoms, dcd->nsets, sizeMB);
01178
01179     starttime = time_of_day();
01180     timestep.coords = (float *)malloc(3*sizeof(float)*natoms);
01181     for (i=0; i<dcd->nsets; i++) {
01182       int rc = read_next_timestep(v, natoms, &timestep);
01183       if (rc) {
01184         fprintf(stderr, "error in read_next_timestep on frame %d\n", i);
01185         return 1;
01186       }
01187     }
01188     endtime = time_of_day();
01189     close_file_read(v);
01190     totaltime += endtime - starttime;
01191     printf("  Time: %5.1f seconds\n", endtime - starttime);
01192     printf("  Speed: %5.1f MB/sec, %5.1f timesteps/sec\n", sizeMB / (endtime - starttime), (dcd->nsets / (endtime - starttime)));
01193   }
01194   printf("Overall Size: %6.1f MB\n", totalMB);
01195   printf("Overall Time: %6.1f seconds\n", totaltime);
01196   printf("Overall Speed: %5.1f MB/sec\n", totalMB / totaltime);
01197   return 0;
01198 }
01199
01200 #endif
01201
```
