```
/*
DISCLAIMER: The source code for all file, graphic, string, and memory primitives such as FileAddLine,
Gcircle, StrDelimit, and MemClear used herein is not included but could easily be created for the platform
for which this source code will be compiled
*/

/*---------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                        */
/*                                                                                                                        */
/*          SUPPORT FOR SPHERICAL 3D VARIABLE PARTICLE CORE ROUTINES                                                      */
/*                                                                                                                        */
/*                                                                                                                        */
/*---------------------------------------------------------------------------------------------------------------------- */
/* (c) Copyright Terence Witt 2007, All Rights Reserved */
/* NULLPHYSICS.COM */
/* SUPPORT FOR PARTICLE CORE RECESSION */

/* master includes */
#include "witt.h"
#include "resource.h"
#include "zproject.h"
/*---------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                        */
/*                                          SHOW CORE FILE                                                                */
/*                                                                                                                        */
/*---------------------------------------------------------------------------------------------------------------------- */
/* show core file */
void Discores(long numpar, double numpos, double numneg)
{

        COLORREF        colormap[]={0, COLORBLUE, COLORGREEN, COLORSKY, COLORRED, COLORPURPLE,
                COLORYELLOW, COLORWHITE, COLORORANGE, COLORGRAY};
        COLORREF        parcolor;
        char            af[300];
        long            shwsiz, rr, xx, yy, nn;
        double          corespread=0.5;

        shwsiz = (long)(400/pow(numpar, 0.235));
        if(numpar>10000)
        {
                shwsiz = (long)(380/pow(numpar, 0.235));
        }
        if(numpar>20000)
        {
                shwsiz = (long)(360/pow(numpar, 0.235));
        }
        corespread = 5.0/(2.0*log(numpar));
        for(nn=1; nn<=numpar; nn++)
        {
                /* use z to set core size */
                xx = (long)((MaxViewX/2)+(shwsiz*xc[nn]));
                yy = (long)((MaxViewX/2)+(shwsiz*yc[nn]) - shwsiz);
                corespread = 0.5;
                if(numpar>2000)
                {
                        if(numpar>20000)
                        {
                                xx += (shwsiz/2);
                                yy -= (long)(2*shwsiz);
                        }else{
                                if(numpar>10000)
                                {
                                        xx += (shwsiz/2);
                                        yy -= (long)(1.5*shwsiz);
                                }else{
                                        xx += (shwsiz/2);
                                        yy -= (long)(1.2*shwsiz);
                                }
                        }
                }else{
                        if(numpar<20)
                        {
                                yy += (shwsiz/2);
                                corespread = 3.0;
                        }
                }
                rr = (long)(corespread*zc[nn] + shwsiz/2);
                parcolor = COLORWHITE;
                if(colorc[nn]<=9)
                {
                        parcolor = colormap[colorc[nn]];
                }
                Gwidth(2);
                Gcircle(VIDEOFRAME, parcolor, xx, yy, rr);
        }
        Gframe(MESSAGEFRAME);
        sprintf(af, "Cores: %ld     Atomic number: %5.0f     Mass Number: %5.0f     Bound e/p: %6.4f",
                numpar, numpos-numneg, numpos, numneg/numpos);
        Gbold(1);
        Gjust(TA_CENTER);
        Gtextframesmall(MESSAGEFRAME, COLORBLUE, "", af, 21);
}

/*---------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                       */
/*                              MULTINUCLEON COULOMB REPULSION                                                           */
/*                                                                                                                       */
/*---------------------------------------------------------------------------------------------------------------------- */
/* calculate Coulomb potential energy for multiple particles */
double SpaceEnergyCoulomb(long coulombmode, long numpar, double d)
{
        double  Crepulse=0.0, rr;
        char    af[200];
        long    i, j;

        /* for each particle in the system, calculate the potential involved in adding another */
        for(i=2; i<=numpar; i++)
        {
                for(j=1; j<i; j++)
                {
                        rr = sqrt((((xc[i]*d)-(xc[j]*d))*((xc[i]*d)-(xc[j]*d)))
                                + (((yc[i]*d)-(yc[j]*d))*((yc[i]*d)-(yc[j]*d)))
                                + (((zc[i]*d)-(zc[j]*d))*((zc[i]*d)-(zc[j]*d))));
                        if(rr==0.0)
                        {
                                sprintf(af, "Coincident Particles: %ld and %ld", i, j);
                                PopWarning("Particle File Error", af);
                        }else{
                                switch(coulombmode)
                                {
                                        /* normal coulomb potential */
                                        case 1:
                                                Crepulse += ((chargec[i]*q*chargec[j]*q)/(4.0*PI*e0*rr));
                                                break;
                                        /* uniformly repulsive for core degeneracy since all bound electrons are
                                                at a potential greater than maxfield */
                                        case 2:
                                                Crepulse += fabs((chargec[i]*q*chargec[j]*q)/(4.0*PI*e0*rr));
                                                break;
                                }
                        }
                }
        }

        return(Crepulse);
}

/* keep precision updated in case of abort */
void TestPrecision(long numpar, double rmid, double rscan, double Ecenter1, double Ecenter2, double corevolume,
        double *Vexpect, double *Eexpect1, double *Eexpect2, double *Vexpectfract, double *Eexpectfract1,
        double *Eexpectfract2)
{
        double     rcen;
        long       nn;

        rcen = coresizec[1];

        *Vexpect = 0.0;
        *Eexpect1 = KKp*((1/rcen) - (1/rscan));
        *Eexpect2 = KKp*((1/rcen) - (1/rmid));
        for(nn=2; nn<=numpar; nn++)
        {
                *Vexpect += (4.0*PI/3.0)*(coresizec[nn]*coresizec[nn]*coresizec[nn]);
        }

        if((*Eexpect1)>0.0)
        {
                *Eexpectfract1 = fabs(100.0*(1.0-(Ecenter1/(*Eexpect1))));
        }
        if((*Eexpect2)>0.0)
        {
                *Eexpectfract2 = fabs(100.0*(1.0-(Ecenter2/(*Eexpect2))));
        }
        if((*Vexpect)>0.0)
        {
                *Vexpectfract = fabs(100.0*(1.0-(corevolume/(*Vexpect))));
        }
}

/* keep precision updated in case of abort, energy only */
void TestPrecisionEnergy(double rmid, double rscan, double Ecenter1, double Ecenter2,
        double *Eexpect1, double *Eexpect2, double *Eexpectfract1, double *Eexpectfract2)
{
        double     rcen;

        rcen = coresizec[1];

        *Eexpect1 = KKp*((1/rcen) - (1/rscan));
        *Eexpect2 = KKp*((1/rcen) - (1/rmid));

        if((*Eexpect1)>0.0)
        {
                *Eexpectfract1 = fabs(100.0*(1.0-(Ecenter1/(*Eexpect1))));
        }
        if((*Eexpect2)>0.0)
```

```
        {
                *Eexpectfract2 = fabs(100.0*(1.0-(Ecenter2/(*Eexpect2))));
        }
}



/*-------------------------------------------------------------------------------------------------------------------- */
/*                                                                                                                      */
/*                                      PARTICLE FILE ROUTINES                                                          */
/*                                                                                                                      */
/*-------------------------------------------------------------------------------------------------------------------- */
/* parse a position line from a nucleon file */
double ParseCorePosition(long parnum, char *textvalue)
{
        char    af[200], txt[200];
        long    ptr, mult;
        double  value=0.0;
        double  ystep, zstep;
        double  ya, yb, yc;
        BOOL    notfound=FALSE;

        strncpy(txt, textvalue, 20);
        txt[20] = 0;
        strupr(txt);
        StrTrim(txt);

        /* various hexagonal closest packing constants */
        ystep = sqrt(3.0)/2.0;
        ya = 1.0/(2.0*sqrt(3.0));
        yb = 1.0/sqrt(3.0);
        yc = 2.0*yb;
        zstep = sqrt(2.0/3.0);

        /* assume number */
        value = atof(txt);
        if(value==0.0)
        {
                switch(txt[0])
                {
                        case 'Y':
                                switch(txt[1])
                                {
                                        case 'S':
                                                value = ystep;
                                                mult = atol(&txt[5]);
                                                if((mult>0) && (mult<=MAX_PARWIDTH))
                                                {
                                                        value *= mult;
                                                }
                                                if(strstr(txt, "-"))
                                                {
                                                        value *= -1.0;
                                                }
                                                break;

                                        case 'A':
                                                value = ya;
                                                ptr = atol(&txt[2]);
                                                if((ptr>=1) && (ptr<=MAX_PARWIDTH))
                                                {
                                                        value += (ptr*ystep);
```

```
						}
						if(strstr(txt, "-"))
						{
							value *= -1.0;
						}
						break;

					case 'B':
						value = yb;
						ptr = atol(&txt[2]);
						if((ptr>=1) && (ptr<=MAX_PARWIDTH))
						{
							value += (ptr*ystep);
						}
						if(strstr(txt, "-"))
						{
							value *= -1.0;
						}
						break;

					default:
						notfound = TRUE;
						break;
				}
				break;

			case 'Z':
				value = zstep;
				mult = atol(&txt[5]);
				if((mult>0) && (mult<=MAX_PARWIDTH))
				{
					value *= mult;
				}
				if(strstr(txt, "-"))
				{
					value *= -1.0;
				}
				break;
		}
	}

	if(notfound)
	{
		sprintf(af, "Particle %ld Position Not Found", parnum);
		PopWarning(af, textvalue);
	}

	return(value);
}


/* load particles from file */
long LoadParticles(char *parfile, BOOL *onepass, BOOL *vary_d, BOOL *vary_ereso, long *conmode, long *coulombmode,
	double rbasis, double *reso, double *Ereso, double *d, double *standoff, double *rscan, double *conrate,

	double *recessrate, double *rc, double *corrlimit, double *corrpow, double *numvary, double *numfixed,
	double *numneg, double *numpos, double *numneutral, double *totalparE, double *totalneutralE)
{
	WFILE	chan;
	char	af[300], txt[300];
	double	xpar=0.0, ypar=0.0, zpar=0.0, radpar, thetapar, phipar, chargepar, fixedpar, energypar, corepar;
	long	colorpar, numpar=0, n;
```

```
BOOL     startrc=FALSE;

/* initial condition of recessing core */
*rc = rbasis;

/* get particle file */
if(!parfile[0])
{
        if(GetText(hWndMain, "Enter Nucleon File [.DAT]"))
        {
                StrTrim(note_text);
                if(!strstr(note_text, "."))
                {
                        strcat(note_text, ".DAT");
                }
                if(!strstr(note_text, ":"))
                {
                        strcpy(af, note_text);
                        strcpy(note_text, NUCPATH);
                        strcat(note_text, af);
                }
                strcpy(parfile, note_text);
        }
}

if(parfile[0])
{
        if(!FileExist(parfile))
        {
                PopWarning("File Not Found", parfile);
        }else{
                Gframe(MESSAGEFRAME);
                sprintf(af, "Loading  %s", parfile);
                Gbold(1);
                Gjust(TA_CENTER);
                Gtextframesmall(MESSAGEFRAME, COLORBLUE, "", af, 21);

                FileGetLine(parfile, 3, af, sizeof(af));

                /* processing parameters keyed to first line, integration resolution, convergence resolution,
                and fixed distance */
                StrDelimit(RESO_TAB, CHAR_PIPE, af, txt, sizeof(txt));
                StrTrim(txt);
                if(atof(txt)>0.0)
                {
                        *reso = atof(txt);
                        if(StrString(txt, "o", FALSE))
                        {
                                *onepass = TRUE;
                        }
                        if(StrString(txt, "c", FALSE))
                        {

                                *coulombmode = 1;
                        }
                        if(StrString(txt, "r", FALSE))
                        {
                                *coulombmode = 2;
                        }
                }
                StrDelimit(ENERGYRESO_TAB, CHAR_PIPE, af, txt, sizeof(txt));
                StrTrim(txt);
```

```
if(atof(txt)>0.0)
{
        *Ereso = atof(txt);
        if(StrString(txt, "f", FALSE))
        {
                *vary_ereso = FALSE;
        }
}
StrDelimit(D_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>0.0)
{
        if(StrString(txt, "f", FALSE))
        {
                *vary_d = FALSE;
        }
        *d = (atof(txt)*rbasis);
}
StrDelimit(STANDOFF_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>0.0)
{
        *standoff = atof(txt);
        if(*standoff>=1.0)
        {
                *standoff = 0.0;
        }
}
StrDelimit(RSCAN_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>0.0)
{
        *rscan = (atof(txt)*rbasis);
}
StrDelimit(CONVERGRATE_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>1.0)
{
        *conrate = atof(txt);
}
StrDelimit(RECESSRATE_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>0.0)
{
        *recessrate = atof(txt);
        if(StrString(txt, "f", FALSE))
        {
                *conmode = *conmode + 1;
        }
        if(StrString(txt, "x", FALSE))
        {
                *conmode = *conmode + 2;
        }
}
StrDelimit(INITRADIUS_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>0.0)
{
        *rc = (atof(txt)*rbasis);
        startrc = TRUE;
}
StrDelimit(CORRLIMIT_TAB, CHAR_PIPE, af, txt, sizeof(txt));
```

```
StrTrim(txt);
if(atof(txt)>0.0)
{
	*corrlimit = atof(txt);
}
StrDelimit(CORRPOW_TAB, CHAR_PIPE, af, txt, sizeof(txt));
StrTrim(txt);
if(atof(txt)>0.0)
{
	*corrpow = atof(txt);
}

/* load x, y and z coordinates for all particles.  This is a parser that
interprets a short set of commands for root three values */
chan = FileOpenRead(parfile);
if(FileOK(chan))
{
	FileInput(chan, af, sizeof(af));
	FileInput(chan, af, sizeof(af));
	FileInput(chan, af, sizeof(af));
	while(FileInput(chan, af, sizeof(af)))
	{
		if(af[0]!=';')
		{
			StrDelimit(X_TAB, CHAR_PIPE, af, txt, sizeof(txt));
			StrTrim(txt);
			if(txt[0])
			{
				if(StrIsNumeric(txt))
				{
					/* input relative coordinates in cartesian
					or spherical coordinates */
					if(StrString(txt, "~", FALSE))
					{
						radpar = ParseCorePosition(numpar, txt);
						StrDelimit(Y_TAB, CHAR_PIPE, af,
							txt, sizeof(txt));
						thetapar = RADIANS*
							ParseCorePosition(numpar, txt);
						StrDelimit(Z_TAB, CHAR_PIPE, af,
							txt, sizeof(txt));
						phipar = RADIANS*
							ParseCorePosition(numpar, txt);
						xpar = radpar*cos(thetapar)*sin(phipar);
						ypar = radpar*sin(thetapar)*sin(phipar);
						zpar = radpar*cos(phipar);
					}else{
						xpar = ParseCorePosition(numpar, txt);
						StrDelimit(Y_TAB, CHAR_PIPE, af,
							txt, sizeof(txt));
						ypar = ParseCorePosition(numpar, txt);
						StrDelimit(Z_TAB, CHAR_PIPE, af,
							txt, sizeof(txt));
						zpar = ParseCorePosition(numpar, txt);
					}
					if(fabs(xpar)<(1.0e-4))
					{
						xpar = 0.0;
					}
					if(fabs(ypar)<(1.0e-4))
					{
						ypar = 0.0;
```

```
}
if(fabs(zpar)<(1.0e-4))
{
        zpar = 0.0;
}

StrDelimit(CHARGE_TAB, CHAR_PIPE, af,
        txt, sizeof(txt));
chargepar = atof(txt);
StrDelimit(CORE_TAB, CHAR_PIPE, af,
        txt, sizeof(txt));
if(atof(txt)>0.0)
{
        corepar = atof(txt);
}else{
        corepar = 1.0;
}
if(StrString(txt, "f", FALSE))
{
        fixedpar = TRUE;
}else{
        if((!startrc) && (corepar!=1.0))
        {
                *rc = corepar*rbasis;
        }
        fixedpar = FALSE;
}
StrDelimit(ENERGY_TAB, CHAR_PIPE, af,
        txt, sizeof(txt));
energypar = atof(txt);
StrDelimit(COLOR_TAB, CHAR_PIPE, af,
        txt, sizeof(txt));
colorpar = atol(txt);

if(numpar<MAX_NUCLEON)
{
        numpar++;
        xc[numpar] = xpar;
        yc[numpar] = ypar;
        zc[numpar] = zpar;
        chargec[numpar] = chargepar;
        coresizec[numpar] = corepar*rbasis;
        fixcorec[numpar] = fixedpar;
        Ebasec[numpar] = energypar;
        colorc[numpar] = colorpar;
        *totalparE += energypar;

        /* for recursion estimate */
        if(fixedpar)
        {
                *numfixed += 1.0;
        }else{
                *numvary += 1.0;
        }
        if(chargepar<0)
        {

                *numneg += 1.0;
        }else{
                if(chargepar>0)
                {
                        *numpos += 1.0;
```

```
                                        }else{
                                            *numneutral += 1.0;
                                            *totalneutralE += energypar;
                                        }
                                    }
                                }else{
                                    sprintf(af,
                                        "Max Nucleon [%ld] Exceeded: %ld",
                                        MAX_NUCLEON, numpar);
                                    PopWarning("Load Nucleons", af);
                                    break;
                                }
                            }
                        }
                    }
                }
                FileClose(chan);
            }

            if(*rc<=0.0)
            {
                sprintf(af, "File: %s   Rc: %5.3e   Rscan: %5.3e", parfile, *rc, *rscan);
                PopWarning("Particle File Error", af);
                numpar = 0;
            }

            /* initialize core sizes */
            if(startrc)
            {
                for(n=1; n<=numpar; n++)
                {
                    coresizec[n] = (*rc);
                }
            }
        }
    }

    return(numpar);
}
```