/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Author: Charles Boyd (cboyd@mailandnews.com) - Texas Tech University

File: atomizer.cpp

Project: Atomizer

Date: 12-07-2000

- note -

Anaglyph Code adapted from Chris Stanley (cstanley@ttacs.ttu.edu) - Texas Tech Univeristy

Project: Real-Time Shared Environment

- Description -

This program displays the shell structure of all known elements found in the periodic table.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <GL/glut.h>

#include <stdlib.h>

#include <stdio.h>

#include <fstream.h>

#include <math.h>

#include <conio.h>

#include <windows.h>

#define PI 3.1415926535898

//function declarations

void display(void);

void idle(void);

void reshape(int w, int h);

void init(void);

void keyboard(unsigned char key, int x, int y);

void build\_atom(int atom\_val);

void build\_stage\_x(int stage\_num);

void build\_stage\_y(int stage\_num);

void build\_stage\_z(int stage\_num);

void set\_rotate\_method();

void load\_atom(int i);

void main\_menu(int value);

void sub\_menu(int value);

void display\_type\_menu(int value);

//global variables

float rotate=0;

int drawn=0, stage\_distance=6, speed=1, flat=1, animation=1, current\_atom=1, chosen=0;

const double PIXELS\_PER\_INCH = 50;

GLfloat ruby\_ambient[] = {0.1745, 0.01175, 0.01175, 0.55};

GLfloat ruby\_diffuse[] = {0.61424, 0.04136, 0.04136, 0.55};

GLfloat ruby\_specular[] = {0.727811, 0.626959, 0.626959, 0.55};

GLfloat emerald\_ambient[] = {0.0215, 0.1745, 0.0215, 0.55};

GLfloat emerald\_diffuse[] = {0.07568, 0.61424, 0.07568, 0.55};

GLfloat emerald\_specular[] = {0.633, 0.727811, 0.633, 0.55};

GLfloat grey\_ambient[] = {0.3, 0.3, 0.3, 0.55};

GLfloat grey\_diffuse[] = {0.3, 0.3, 0.3, 0.55};

GLfloat grey\_specular[] = {0.3, 0.3, 0.3, 0.55};

GLfloat red\_light\_ambient[]= {1.0,0.0,0.0,1.0};

GLfloat red\_light\_diffuse[]= {1.0,0.0,0.0,1.0};

GLfloat red\_light\_specular[]={1.0,0.0,0.0,1.0};

GLfloat blue\_light\_ambient[]= {0.0,1.0,0.0,1.0};

GLfloat blue\_light\_diffuse[]= {0.0,1.0,0.0,1.0};

GLfloat blue\_light\_specular[]={0.0,1.0,0.0,1.0};

GLfloat white\_light[] = {1.0, 1.0, 1.0, 1.0};

GLfloat light\_position[] = {25.0, 25.0, 50.0, 0.0};

GLfloat Camera[] = {0.0,0.0,-65.0,0.0,0.0,0.0}; //X,Y,Z,X-Rotate,Y-Rotate,Z-Rotate

//struct used to hold stage coordinates

struct element\_stage\_info

{

float elements;

float rot\_x;

float rot\_y;

float rot\_z;

float x[120];

float y[120];

float z[120];

};

struct ProgramState

{

int w;

int h;

GLdouble RotationY;

double eye;

double zscreen;

double znear;

double zfar;

double RotationIncrement;

int solidmodel;

bool bStereo;

bool bInterlaced;

bool bServerMode;

};

struct element\_stage\_info stage[9];

struct ProgramState ps;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: main()

Description: standard main

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main (int argc, char \*argv[])

{

int choice;

glutInit (&argc, argv);

glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGB);

choice = MessageBox(NULL,"Would You Like To Run In Fullscreen Mode?", "Start FullScreen?",MB\_YESNO|MB\_ICONQUESTION);

if (choice == IDNO)

{

glutInitWindowSize(500, 500); // Windowed Mode

glutInitWindowPosition(100, 100);

glutCreateWindow ("ATOMIZER");

}

else

{

glutCreateWindow ("ATOMIZER");

glutFullScreen();

}

init();

glutDisplayFunc (display);

glutKeyboardFunc (keyboard);

glutIdleFunc (idle);

glutReshapeFunc (reshape);

glutMainLoop ();

return 0;

}//end main()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: display()

Description: openGL standard display function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display (void)

{

double xfactor=1.0, yfactor=1.0;

double Eye =0.0;

int i, test;

if(ps.bStereo)

test = 2;

else

test = 1;

if(ps.w < ps.h)

{

xfactor = 1.0;

yfactor = ps.h/ps.w;

}

else if(ps.h < ps.w)

{

xfactor = ps.w/ps.h;

yfactor = 1.0;

}

glClear (GL\_COLOR\_BUFFER\_BIT);

for(i=0;i<test;i++)

{

glClear(GL\_DEPTH\_BUFFER\_BIT);

if(!ps.bStereo)

{

glEnable(GL\_LIGHT0);

glDisable(GL\_LIGHT1);

glDisable(GL\_LIGHT2);

Eye = 0.0;

glStencilFunc(GL\_ALWAYS,1,1);

}

else //anaglyph mode

{

glEnable(GL\_LIGHT0+ i+1);

glDisable(GL\_LIGHT0);

glStencilFunc(GL\_ALWAYS,1,1);

if(i==0) /\* left eye - RED \*/

{

Eye = ps.eye;

glColorMask(GL\_TRUE,GL\_FALSE,GL\_FALSE,GL\_TRUE);

}

else /\* right eye - GREEN <--Change to BLUE for blue lenses \*/

{

Eye = -ps.eye;

glColorMask(GL\_FALSE,GL\_TRUE,GL\_FALSE,GL\_TRUE);

}

}

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(

(-(ps.w/(2.0\*PIXELS\_PER\_INCH))+Eye)\*(ps.znear/ps.zscreen)\*xfactor,

(ps.w/(2.0\*PIXELS\_PER\_INCH)+Eye)\*(ps.znear/ps.zscreen)\*xfactor,

-(ps.h/(2.0\*PIXELS\_PER\_INCH))\*(ps.znear/ps.zscreen)\*yfactor,

(ps.h/(2.0\*PIXELS\_PER\_INCH))\*(ps.znear/ps.zscreen)\*yfactor,

ps.znear, ps.zfar);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(Eye,0.0,0.0);

glTranslated(0,0,-ps.zscreen);

glTranslatef(Camera[0],Camera[1],Camera[2]);

glRotatef(Camera[3],.1,0,0);

glRotatef(Camera[4],0,.1,0);

glRotatef(Camera[5],0,0,.1);

if (chosen == 1)

{

glPushMatrix();

if(!ps.bStereo)

{

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, emerald\_specular);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, emerald\_ambient);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, emerald\_diffuse);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 76.0);

}

else

{

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, grey\_specular);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, grey\_ambient);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, grey\_diffuse);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 76.0);

}

glutSolidSphere(4.5, 15, 15);

glPopMatrix();

}

//enable electron material

if(!ps.bStereo)

{

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, ruby\_specular);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ruby\_ambient);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, ruby\_diffuse);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 76.0);

}

else

{

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, grey\_specular);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, grey\_ambient);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, grey\_diffuse);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 76.0);

}

for (int i=1; stage[i].elements != 0; i++)

{

glPushMatrix();

glRotatef(rotate, stage[i].rot\_x, stage[i].rot\_y, stage[i].rot\_z);

for (int j=0; j < stage[i].elements; j++)

{

glTranslatef(stage[i].x[j], stage[i].y[j], stage[i].z[j]);

glutSolidSphere(2.0, 10, 10);

glTranslatef(stage[i].x[j]\*(-1), stage[i].y[j]\*(-1), stage[i].z[j]\*(-1));

}//end for

glPopMatrix();

}//end for

glDisable(GL\_LIGHT0 + i+1);

}

glColorMask(GL\_TRUE,GL\_TRUE,GL\_TRUE,GL\_TRUE);

glutSwapBuffers ();

}//end display()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: idle()

Description: openGL idle function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void idle (void)

{

if (animation == 1)

rotate = rotate + speed;

glutPostRedisplay();

}//end idle()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: reshape()

Description: openGL reshape function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void reshape (int w, int h)

{

ps.w = w;

ps.h = h;

glViewport (0.0, 0.0, (GLfloat) w, (GLfloat) h);

glMatrixMode (GL\_PROJECTION);

glLoadIdentity ();

gluPerspective (45.0, (GLfloat) w / (GLfloat) h, 1.0, 1000.0);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glLightfv(GL\_LIGHT1, GL\_POSITION, light\_position);

glLightfv(GL\_LIGHT2, GL\_POSITION, light\_position);

glMatrixMode (GL\_MODELVIEW);

glLoadIdentity();

}//end reshape()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: init()

Description: openGL init function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init(void)

{

int menuId, sub1, sub2, sub3, sub4;

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_FLAT);

glEnable(GL\_DEPTH\_TEST);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, white\_light);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, white\_light);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, white\_light);

glLightfv(GL\_LIGHT1, GL\_AMBIENT, red\_light\_ambient);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, red\_light\_diffuse);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, red\_light\_specular);

glLightfv(GL\_LIGHT2, GL\_AMBIENT, blue\_light\_ambient);

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, blue\_light\_diffuse);

glLightfv(GL\_LIGHT2, GL\_SPECULAR, blue\_light\_specular);

//attenuation setup

glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, 0.3);

glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, 0.0);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glShadeModel(GL\_SMOOTH);

ps.eye=0.80;

ps.zscreen = 10.0;

ps.znear = 0.1;

ps.zfar = 1000000.0;

ps.RotationY = 0.0;

ps.RotationIncrement = 4.0;

ps.solidmodel = 1;

ps.bInterlaced = true;

ps.bStereo = false;

glutCreateMenu(display\_type\_menu);

glutAddMenuEntry ("2-D", 1);

glutAddMenuEntry ("3-D", 2);

glutAttachMenu (GLUT\_LEFT\_BUTTON);

sub1 = glutCreateMenu(sub\_menu);

glutAddMenuEntry ("Hydrogen", 1);

glutAddMenuEntry ("Helium", 2);

glutAddMenuEntry ("Lithium", 3);

glutAddMenuEntry ("Beryllium", 4);

glutAddMenuEntry ("Boron", 5);

glutAddMenuEntry ("Carbon", 6);

glutAddMenuEntry ("Nitrogen", 7);

glutAddMenuEntry ("Oxygen", 8);

glutAddMenuEntry ("Fluorine", 9);

glutAddMenuEntry ("Neon", 10);

glutAddMenuEntry ("Sodium", 11);

glutAddMenuEntry ("Magnesium", 12);

glutAddMenuEntry ("Aluminum", 13);

glutAddMenuEntry ("Silicon", 14);

glutAddMenuEntry ("Phosphorus", 15);

glutAddMenuEntry ("Sulfur", 16);

glutAddMenuEntry ("Chlorine", 17);

glutAddMenuEntry ("Argon", 18);

glutAddMenuEntry ("Potassium", 19);

glutAddMenuEntry ("Calcium", 20);

glutAddMenuEntry ("Scandium", 21);

glutAddMenuEntry ("Titanium", 22);

glutAddMenuEntry ("Vanadium", 23);

glutAddMenuEntry ("Chromium", 24);

glutAddMenuEntry ("Manganese", 25);

glutAddMenuEntry ("Iron", 26);

glutAddMenuEntry ("Cobalt", 27);

glutAddMenuEntry ("Nickel", 28);

glutAddMenuEntry ("Copper", 29);

glutAddMenuEntry ("Zinc", 30);

sub2 = glutCreateMenu(sub\_menu);

glutAddMenuEntry ("Gallium", 31);

glutAddMenuEntry ("Germanium", 32);

glutAddMenuEntry ("Arsenic", 33);

glutAddMenuEntry ("Selenium", 34);

glutAddMenuEntry ("Bromine", 35);

glutAddMenuEntry ("Krypton", 36);

glutAddMenuEntry ("Rubidium", 37);

glutAddMenuEntry ("Strontium", 38);

glutAddMenuEntry ("Yttrium", 39);

glutAddMenuEntry ("Zirconium", 40);

glutAddMenuEntry ("Niobium", 41);

glutAddMenuEntry ("Molybdenum", 42);

glutAddMenuEntry ("Technetium", 43);

glutAddMenuEntry ("Ruthenium", 44);

glutAddMenuEntry ("Rhodium", 45);

glutAddMenuEntry ("Palladium", 46);

glutAddMenuEntry ("Silver", 47);

glutAddMenuEntry ("Cadmium", 48);

glutAddMenuEntry ("Indium", 49);

glutAddMenuEntry ("Tin", 50);

glutAddMenuEntry ("Antimony", 51);

glutAddMenuEntry ("Tellurium", 52);

glutAddMenuEntry ("Iodine", 53);

glutAddMenuEntry ("Xenon", 54);

glutAddMenuEntry ("Cesium", 55);

glutAddMenuEntry ("Barium", 56);

glutAddMenuEntry ("Lanthanum", 57);

glutAddMenuEntry ("Cerium", 58);

glutAddMenuEntry ("Praseodymium", 59);

glutAddMenuEntry ("Neodymium", 60);

sub3 = glutCreateMenu(sub\_menu);

glutAddMenuEntry ("Promethium", 61);

glutAddMenuEntry ("Samarium", 62);

glutAddMenuEntry ("Europium", 63);

glutAddMenuEntry ("Gadolinium", 64);

glutAddMenuEntry ("Terbium", 65);

glutAddMenuEntry ("Dysprosium", 66);

glutAddMenuEntry ("Holmium", 67);

glutAddMenuEntry ("Erbium", 68);

glutAddMenuEntry ("Thulium", 69);

glutAddMenuEntry ("Ytterbium", 70);

glutAddMenuEntry ("Lutetium", 71);

glutAddMenuEntry ("Hafnium", 72);

glutAddMenuEntry ("Tantalum", 73);

glutAddMenuEntry ("Tungsten", 74);

glutAddMenuEntry ("Rhenium", 75);

glutAddMenuEntry ("Osmium", 76);

glutAddMenuEntry ("Iridium", 77);

glutAddMenuEntry ("Platinum", 78);

glutAddMenuEntry ("Gold", 79);

glutAddMenuEntry ("Mercury", 80);

glutAddMenuEntry ("Thallium", 81);

glutAddMenuEntry ("Lead", 82);

glutAddMenuEntry ("Bismuth", 83);

glutAddMenuEntry ("Polonium", 84);

glutAddMenuEntry ("Astatine", 85);

glutAddMenuEntry ("Radon", 86);

glutAddMenuEntry ("Francium", 87);

glutAddMenuEntry ("Radium", 88);

glutAddMenuEntry ("Actinium", 89);

glutAddMenuEntry ("Thorium", 90);

sub4 = glutCreateMenu(sub\_menu);

glutAddMenuEntry ("Protactinium", 91);

glutAddMenuEntry ("Uranium", 92);

glutAddMenuEntry ("Neptunium", 93);

glutAddMenuEntry ("Plutonium", 94);

glutAddMenuEntry ("Americium", 95);

glutAddMenuEntry ("Curium", 96);

glutAddMenuEntry ("Berkelium", 97);

glutAddMenuEntry ("Californium", 98);

glutAddMenuEntry ("Einsteinium", 99);

glutAddMenuEntry ("Fermium", 100);

glutAddMenuEntry ("Mendelevium", 101);

glutAddMenuEntry ("Nobelium", 102);

glutAddMenuEntry ("Lawrencium", 103);

glutAddMenuEntry ("Rutherfordium", 104);

glutAddMenuEntry ("Dubnium", 105);

glutAddMenuEntry ("Bohrium", 106);

glutAddMenuEntry ("Seaborgium", 107);

glutAddMenuEntry ("Hassium", 108);

glutAddMenuEntry ("Meitnerium", 109);

glutAddMenuEntry ("Ununnilium", 110);

glutAddMenuEntry ("Unununium", 111);

glutAddMenuEntry ("Ununbium", 112);

glutAddMenuEntry ("Ununquadium", 113);

glutAddMenuEntry ("Ununhexium", 114);

glutAddMenuEntry ("Ununoctium", 115);

glutAddMenuEntry ("Not Real Element", 116);

menuId = glutCreateMenu(main\_menu);

glutAddSubMenu( "Elements 1-30", sub1);

glutAddSubMenu( "Elements 31-60", sub2);

glutAddSubMenu( "Elements 61-90", sub3);

glutAddSubMenu( "Elements 90-117", sub4);

glutSetMenu(menuId);

glutAttachMenu (GLUT\_RIGHT\_BUTTON);

}//end init()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: keyboard()

Description: openGL reshape function

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void keyboard(unsigned char key, int x, int y)

{

switch (key)

{

case 'q':

case 'Q':

exit(0);

case '.':

speed=speed+1;

break;

case ',':

if (speed > 0)

speed--;

break;

case 'u':

animation=1;

break;

case 'p':

animation=0;

break;

case 'M':

case 'm': /\* mono \*/

ps.bStereo = false;

ps.eye = 0.0;

reshape(ps.w, ps.h);

//reshape(ps.h, ps.w);

break;

case 'S':

case 's': /\* stereo \*/

ps.bStereo = true;

ps.eye = 0.80;

reshape(ps.w, ps.h);

//reshape(ps.h, ps.w);

break;

case '-': /\* Camera focus plane foward \*/

ps.eye -=.1;

break;

case '+': /\* Camera focus plane foward \*/

ps.eye +=.1;

break;

case 'F':

case 'f': /\* Camera focus plane foward \*/

ps.zscreen+=.1;

break;

case 'V':

case 'v': /\* Camera focus plane foward \*/

ps.zscreen-=.1;

break;

case 'A': /\* Z double step \*/

Camera[2]+=.5;

break;

case 'a': /\* Camera Z foward \*/

Camera[2]+= 0.1;

break;

case 'Z': /\* Z double step backward \*/

Camera[2]-= 0.5;

break;

case 'z': /\* Camera Z backward \*/

Camera[2]-= 0.1;

break;

case '[': /\* Camera Y counter clockwise rotate \*/

Camera[4]-= 1;

break;

case ']': /\* Camera Y clockwise rotate \*/

Camera[4]+= 1;

break;

}

}//end keyboard

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: build\_atom()

Description: calls build\_stage() to build each stage of the atom.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void build\_atom(int atom\_val)

{

chosen = 1; //variable set to 1, to let program know an element

//has been chosen

set\_rotate\_method(); //sets the rotation axis for each stage

load\_atom(atom\_val); //loads in the specified element parameters

current\_atom = atom\_val; //sets the current atom value

if (flat == 0)

{

if (stage[1].elements != 0)

build\_stage\_y(1);

if (stage[2].elements != 0)

build\_stage\_x(2);

if (stage[3].elements != 0)

build\_stage\_z(3);

if (stage[4].elements != 0)

build\_stage\_y(4);

if (stage[5].elements != 0)

build\_stage\_x(5);

if (stage[6].elements != 0)

build\_stage\_y(6);

if (stage[7].elements != 0)

build\_stage\_x(7);

}

if (flat == 1)

{

if (stage[1].elements != 0)

build\_stage\_z(1);

if (stage[2].elements != 0)

build\_stage\_z(2);

if (stage[3].elements != 0)

build\_stage\_z(3);

if (stage[4].elements != 0)

build\_stage\_z(4);

if (stage[5].elements != 0)

build\_stage\_z(5);

if (stage[6].elements != 0)

build\_stage\_z(6);

if (stage[7].elements != 0)

build\_stage\_z(7);

}

}//end build atom()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: build\_stage\_x()

Description: computes the value of each 'electron' in a stage around the x-axis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void build\_stage\_x(int n)

{

float angle = .001, angle\_size;

int radius;

double two\_pi = 3.1415926535\*2;

angle\_size = 360/stage[n].elements;

radius = n \* stage\_distance;

for (int i=0; i < stage[n].elements; i++)

{

if (angle > 180)

{

stage[n].y[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].z[i] = sqrt(( (radius\*radius) - (stage[n].y[i]\*stage[n].y[i]) ));

stage[n].x[i] = 0;

}

else

{

stage[n].y[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].z[i] = (-1)\*sqrt(( (radius\*radius) - (stage[n].y[i]\*stage[n].y[i]) ));

stage[n].x[i] = 0;

}

angle = angle + angle\_size;

}//end for

}//end build\_stage\_x()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: build\_stage\_y()

Description: computes the value of each 'electron' in a stage around the y-axis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void build\_stage\_y(int n)

{

float angle = .001, angle\_size;

int radius;

double two\_pi = 3.1415926535\*2;

angle\_size = 360/stage[n].elements;

radius = n \* stage\_distance;

for (int i=0; i < stage[n].elements && angle <= 360; i++)

{

if (angle > 180)

{

stage[n].x[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].z[i] = sqrt(( (radius\*radius) - (stage[n].x[i]\*stage[n].x[i]) ));

stage[n].y[i] = 0;

}

else

{

stage[n].x[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].z[i] = (-1)\*sqrt(( (radius\*radius) - (stage[n].x[i]\*stage[n].x[i]) ));

stage[n].y[i] = 0;

}

angle = angle + angle\_size;

}//end for

}//end build\_stage\_y()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: build\_stage\_z()

Description: computes the value of each 'electron' in a stage around the z-axis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void build\_stage\_z(int n)

{

float angle = 0.0, angle\_size;

int radius;

double two\_pi = 3.1415926535\*2;

angle\_size = 360/stage[n].elements;

radius = n \* stage\_distance;

for (int i=0; i < stage[n].elements; i++)

{

if (angle > 180)

{

stage[n].y[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].x[i] = sqrt(( (radius\*radius) - (stage[n].y[i]\*stage[n].y[i]) ));

stage[n].z[i] = 0;

}

else

{

stage[n].y[i] = (cos((two\_pi\*angle)/360)) \* radius;

stage[n].x[i] = (-1)\*sqrt(( (radius\*radius) - (stage[n].y[i]\*stage[n].y[i]) ));

stage[n].z[i] = 0;

}

angle = angle + angle\_size;

}//end for

}//end build\_stage\_z()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: set\_rotate\_method()

Description: sets the rotation of each stage

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void set\_rotate\_method()

{

if (flat == 1)

{

stage[1].rot\_x = 0.0; stage[1].rot\_y = 0.0; stage[1].rot\_z = 1.0;

stage[2].rot\_x = 0.0; stage[2].rot\_y = 0.0; stage[2].rot\_z = -1.0;

stage[3].rot\_x = 0.0; stage[3].rot\_y = 0.0; stage[3].rot\_z = 1.0;

stage[4].rot\_x = 0.0; stage[4].rot\_y = 0.0; stage[4].rot\_z = -1.0;

stage[5].rot\_x = 0.0; stage[5].rot\_y = 0.0; stage[5].rot\_z = 1.0;

stage[6].rot\_x = 0.0; stage[6].rot\_y = 0.0; stage[6].rot\_z = -1.0;

stage[7].rot\_x = 0.0; stage[7].rot\_y = 0.0; stage[7].rot\_z = 1.0;

}

if (flat == 0)

{

stage[1].rot\_x = 0.0; stage[1].rot\_y = 1.0; stage[1].rot\_z = 0.0;

stage[2].rot\_x = 1.0; stage[2].rot\_y = 0.0; stage[2].rot\_z = 0.0;

stage[3].rot\_x = 0.0; stage[3].rot\_y = 0.0; stage[3].rot\_z = 1.0;

stage[4].rot\_x = 0.0; stage[4].rot\_y = 1.0; stage[4].rot\_z = 1.0;

stage[5].rot\_x = 1.0; stage[5].rot\_y = 0.0; stage[5].rot\_z = 1.0;

stage[6].rot\_x = 0.0; stage[6].rot\_y = -1.0; stage[6].rot\_z = -1.0;

stage[7].rot\_x = -1.0; stage[7].rot\_y = 0.0; stage[7].rot\_z = -1.0;

}

}//end set\_rotate\_method();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: load\_atom()

Description: loads the specified atom from the database text file 'elements.db'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void load\_atom(int i)

{

char name[18], symbol[4];

ifstream infile;

infile.open("elements.db");

for (int j=1; j != i; j++)

{

infile>>name;

infile>>symbol;

infile>>stage[1].elements;

infile>>stage[2].elements;

infile>>stage[3].elements;

infile>>stage[4].elements;

infile>>stage[5].elements;

infile>>stage[6].elements;

infile>>stage[7].elements;

}

infile>>name;

infile>>symbol;

infile>>stage[1].elements;

infile>>stage[2].elements;

infile>>stage[3].elements;

infile>>stage[4].elements;

infile>>stage[5].elements;

infile>>stage[6].elements;

infile>>stage[7].elements;

infile.close();

}//end load\_atom()

void main\_menu(int value)

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: sub\_menu()

Description: sub menus to select elements

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void sub\_menu(int value)

{

build\_atom(value);

}//end main\_menu

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: main\_menu()

Description: secondary menu toggles 2-D and 3-D

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display\_type\_menu(int value)

{

switch(value)

{

case 1:

flat = 1;

build\_atom(current\_atom);

break;

case 2:

flat = 0;

build\_atom(current\_atom);

break;

}

}//end menu\_two