SETI Visualizations:

Development of Graphical Utilities for Explaining SETI

Immanuel Buder

Computer Systems Lab

June 12, 2003

Abstract

The SETI (Search for Extra-Terrestrial Intelligence) program has been active since 1960[1]. It publicises itself well with many text-based sites. However, few graphical utilities exist to explain the program. The purpose of this project is to develop such utilities using OpenGL and possibly POV-Ray.

1 List of Terms

- SETI
- OpenGL
- POV-Ray

- Drake Equation
- C++
- Project Phoenix
- Project Argus
- SETI@home
- milky
- explorer
- strategy
- home
- breakup

2 Introduction

The SETI program is an effort to detect communication or other signals sent by intelligent civilizations from other star systems. It has been active since 1960[1] and uses mostly radio astronomy. Many SETI organizations depend on donations to keep running. Therefore, it is important for them to make the public aware of their efforts. There are many text-based explanations of the SETI program, its philosophy, its method, what it hopes to accomplish.

For example, Carl Sagan, a famous SETI advocate, presented his cast for the SETI program in "Can SETI Succeed"[8]. However, few of these explanations are supplemented with graphics. The purpose of this project is to develop graphical utilities to help SETI explain its efforts to the public. Such utilities could be used by SETI organizations for fundraising and public information activities. The project will avoid more technical aspects not needed for a basic understanding. Current data will not be discussed.

2.1 Background

"In early November, 1961, a group of scientists and engineers met at the Green Bank Observatory to discuss the possibility of using the techniques of radio astronomy to detect evidence of intelligent life outside our Solar System."[2] The Drake Equation, a tool used to estimate the number of communicating civilizations in the galaxy, was first discussed here. At the same time, the first SETI search, Project Ozma, was conducted at the National Radio Astronomy Observatory.[1]

2.2 Other SETI Utilities

There are very few graphical utilities capable of explaining the goals and reasoning of the SETI program. The SETI@home[4] program does have an advanced graphical display; however, it displays current SETI data. Data display is not what is needed, nor is it in the scope of this project. Before raw or even analyzed data can be understood, a user must grasp the basic principles of SETI.

3 Proceedure

3.1 Timeplan

- Sept. 9-20: Basic Project Design and Proposal
- Sept. 23-Oct. 4: Develop milky
- Oct. 7-Nov. 22: Develop explorer
- Nov. 25-Dec. 13: Develop strategy
- Dec. 16-Jan.24: Construct poster
- Jan. 27-Feb. 28: Develop home
- Mar. 3-21: Develop breakup
- Apr. 6-May 16: Analysis and Conclusions
- May 27-June 17: Write Technical Paper

3.2 Resources

The following programming tools were used:

- GNU C++
- OpenGL
- POV-Ray

The construction of each program will be described separately.

3.3 milky: The Drake Equation Graphical Simulator

The Drake Equation graphical simulator (milky.cpp, see Appendix A) was developed in C++ using the OpenGL graphics library. The first step was to create a graphics shell, a standard C++ program that creates a graphics environment. Next, special coding was needed to impliment textures which were used to display the galaxy in background. Then, the Drake Equation was added in the form of an alien civilization density. The probability of finding an alien civilization in a given grid square was set equal to the density. Finally, a control window was added to allow the user to change Drake factors.

3.3.1 The Drake Equation

The Drake Equation[2], often written as

$$N = R f_p n_e f_l f_i f_c L,$$

is a way to estimate the number of sources that SETI might detect. The purpose of the drake equation is to identify the factors that influence this number:

- R is the rate at which new stars are formed
- f_p is the fraction of such starts that have planets
- n_e is the fraction of such planets which can support life
- f_l is the fraction of such planets on which live develops

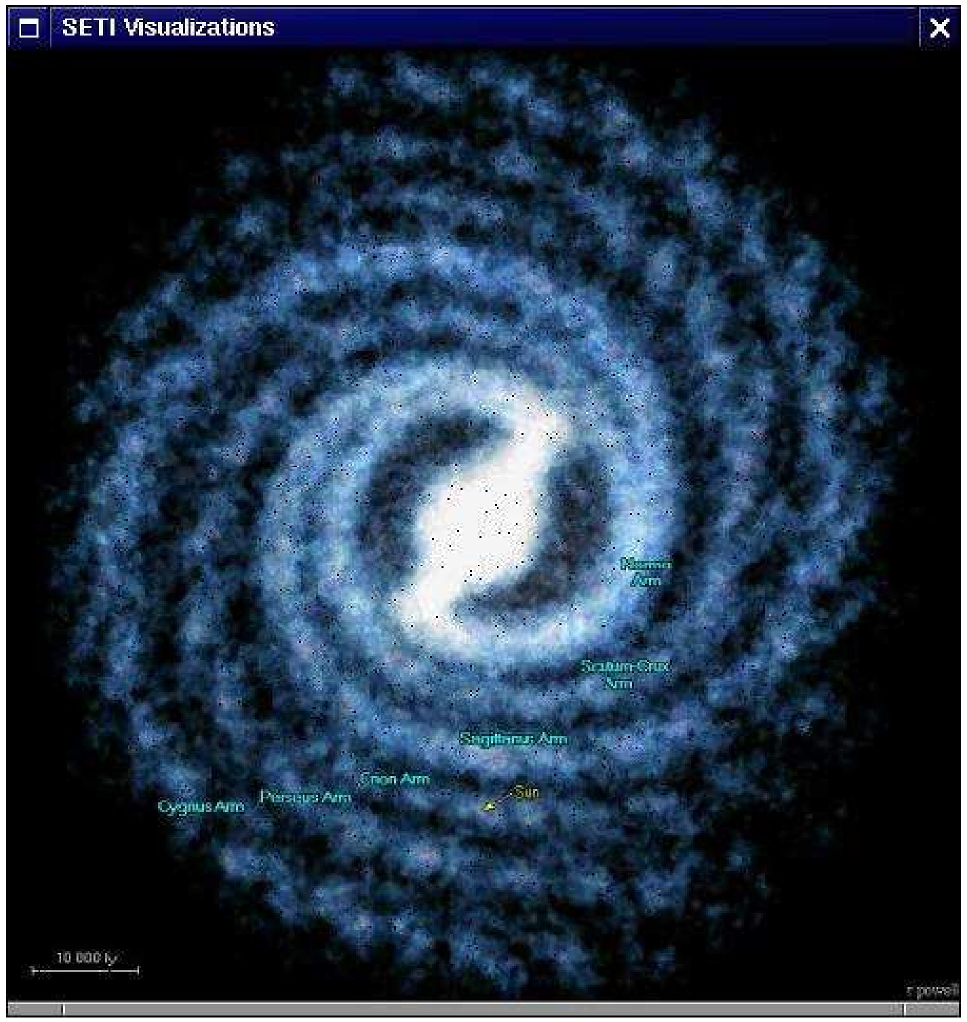

Figure 1: The black dots represent squares with alien civilizations.

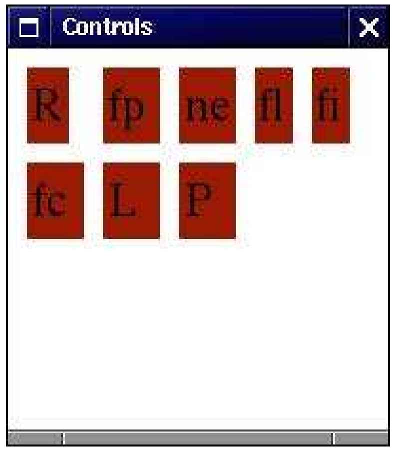

Figure 2: The control panel for milky.

- f_i is the fraction of biospheres where intelligence arises
- f_C is the fraction of intelligent civilizations which develop interstellar communications
- L is the length of time that such a civilizaion remains in communication

3.3.2 Code Analysis

The code for milky can be broken down by purpose:

- Texture Loading
- Control Panel
- OpenGL Initialization
- Drake Equation Display

3.3.3 Texture Loading

The actual code to convert an image file in targa format to an OpenGL texture (tgaload.h) is not shown here. It was gratefully provided by classmate Joey Turner. The implimentation was then relatively easy, since the texture was a rectangle being mapped to another rectangle. Although the code is relatively short, it took several weeks to create, mostly spent finding the correct texture loading header.

3.3.4 Control Panel

This contains the largest number of lines of code; however, they are largely routine. The ”display” and ”mouse” functions (see Appendix A) all have the same format, and most of the lines could be reused. Positioning the buttons in the control window was the hardest part since this programmer does not think visually in pixels.

3.3.5 OpenGL Initialization

This code is perhaps the least understandable since OpenGL often requires strange and unusual initialization patterns. Several weeks were spent in experimentation to find an initialization sequence that would allow textures and colors. Textures were implimented; however, colors proved to be impossible without an unreasonable expenditure of time. Therefore they were abandoned. The initialization code, with a few modifications, was used for all OpenGL parts of the project.

3.3.6 Drake Equation Display

The actual display code is rather short because it only prints dots on the screen. The background is already drawn as part of the texture code. Some time was spent modifying this code in an attempt to draw the dots in color. It failed.

3.4 explorer: The Expanding Civilization Simulation

The expanding civilization simulation (explorer.cpp, see Appendix B) used the same shell developed for milky. Modifications (such as double buffering) had to be made to allow for animation. Then, the various expansion models (see move1...move4, Appendix B) were introduced. Finally, a counter was added to compare the various models.

3.4.1 Code Analysis

The major parts of the explorer code are

- OpenGL Initialization
- Texture Loading
- Evolution
- Movement Algorithm

The first two were taken with a few modifications from milky.

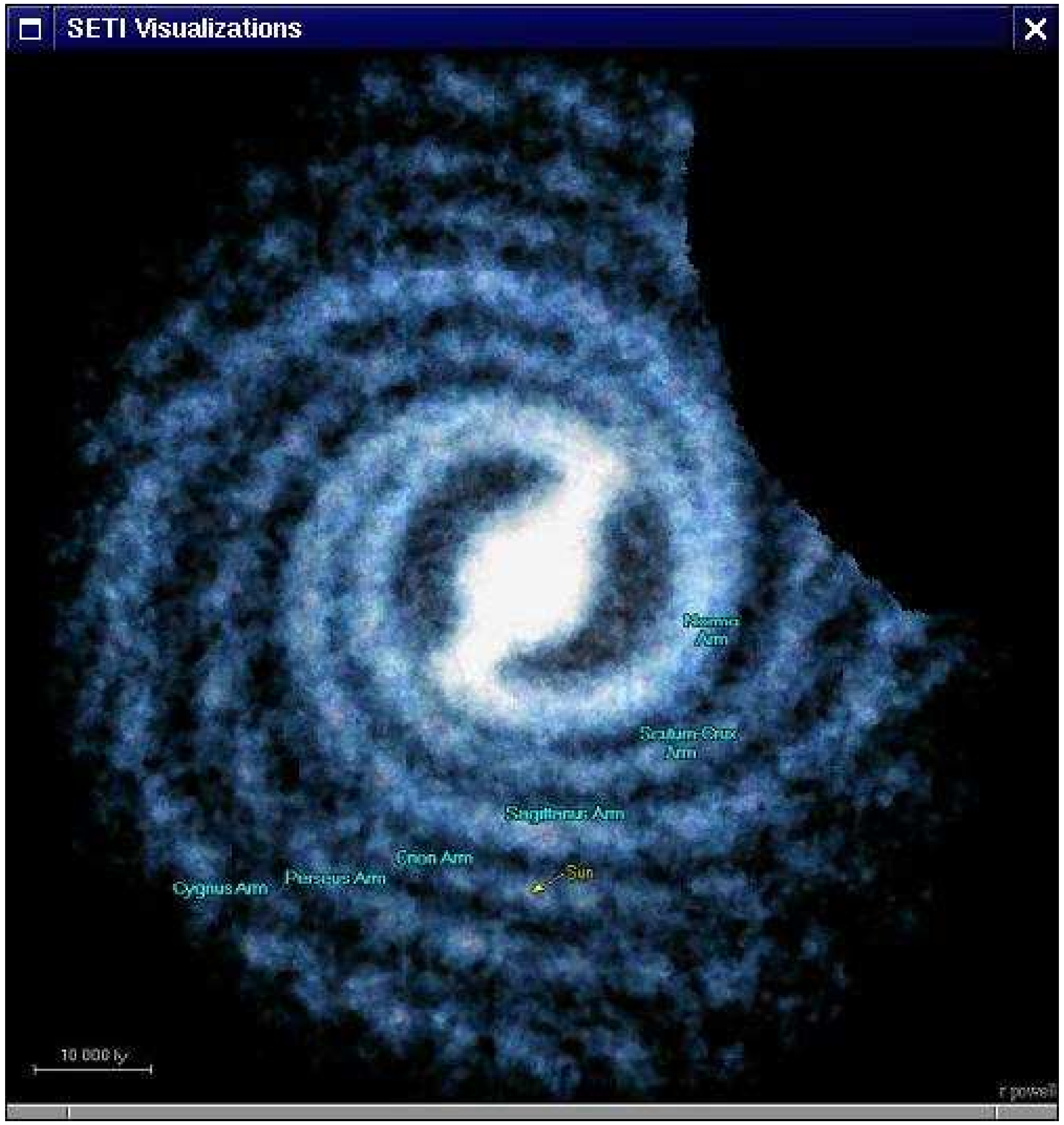

Figure 3: The black area has been taken over by an explorer civilization.

3.4.2 Evolution

The evolution code uses a simple algorithm: Move across the screen. For each planet where there are aliens, call the movement algorithm. If the movement algorithm decides that those aliens will colonize, it will set a special flag in the planet where they are colonizing. Once all planets have moved, the flags will be converted to new aliens.

3.4.3 Movement Algorithm

There are four separate movement algorithms. The first simulates aliens who move in a random direction, but always move if possible. The second simulates aliens who choose a random direction and move only in that direction. The third simulates aliens who always move in the same direction if possible, then move in other directions when otherwise. The fourth simulates the same aliens as the first, except adds a hesitation factor such that the aliens may decide not the colonize each turn.

3.5 strategy: Two Ways to Do SETI

The strategy demonstration uses POV-Ray. Two images, strat1 and strat2 (see strat1.pov, Appendix C and strat2.pov, Appendix D) were created. The first deals with a targeted search like Project Phoenix[3], the second with an all-sky search like Project Argus[1]. A method of symbolically representing the telescopes was required. The programmer chose cones since they naturally correspond to a telescope's field of view. Also, a new implimentation of textures was required because POV-Ray treats textures differently from OpenGL. Once the components were created, several tests were needed to optimize their arrangement.

3.5.1 Targeted Search

The targeted search strategy identifies a few candidate stars as having a high probability of supporting communicating life. Age, brightness, and possibility of planets are among the criteria used to select the candidates. The targeted search uses a small number of very large telescopes. They have very high sensitivity; that is they can detect very weak signals. However, they have a small angle of view; they cannot see much of the sky at once. When the location of sky to be studied is known, small angle of view is not a great limitation.

3.5.2 All-Sky Search

The all-sky search makes no a priori assumptions about where communicating life might be found. Because the entire sky must be considered, small angle of view becomes a limiting factor. The amount of sky visible at one time determines how long it takes to search the entire sky. Therefore a larger number of smaller telescopes with larget angles of view is the optimal method for conducting an all-sky search. Additonally, smaller telescopes are cheaper so many more can be purchased with the same amount of money used to buy one larget telescope (e.g. for a targeted search). However, the lower sensitivity of smaller telescopes is still limiting. Weak signals that a targeted search could detect might be missed by an all-sky search.

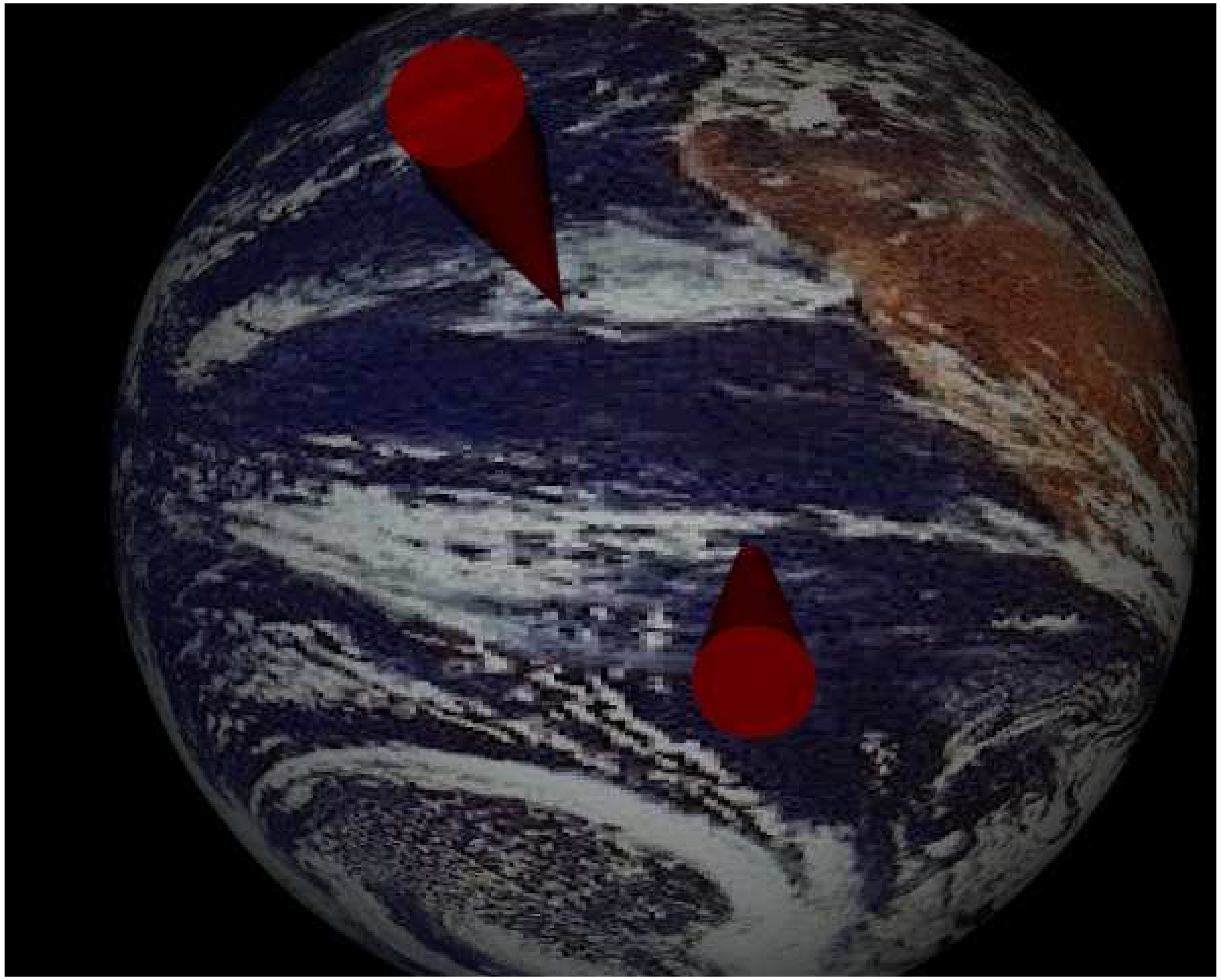

Figure 4: The image for a targeted search.

3.6 home: Why Does SETI@home Look for Gaussians

home, the first in a series of SETI@home[4] utilities, also uses POV-Ray. The symbolic representation developed for strategy was reused here. The new element, a signal source, is represented by a blue sphere. Since the purpose of this program is to show how signal strength changes with time, static images like those of strategy were inadequate. A different technique was used to develop animation in POV-Ray. A clock variable was invluded in the source, causing one of the image parameters to vary with time. The povray renderer was invoked multiple times with different values of the clock variable. An auxiliary C++ program was used to auotmate this task. The POV-Ray source home.pov is given in Appendix E.

3.7 breakup: How Is SETI@home Data Distributed

breakup, the second in the series op SETI@home[4] utilities, uses C++ and OpenGL. The OpenGL framework is the same that was developed for milky and explorer. breakup shows how the data gathered by the Arecibo telescope is divided into work units for distribution to individual participants. The main display is a frequency vs. time chart with the data selected for a work unit highlighted. The user can input new work unit specifications, and the program will calculate the new work unit size. The C++ source breakup.cpp is given in Appendix F.

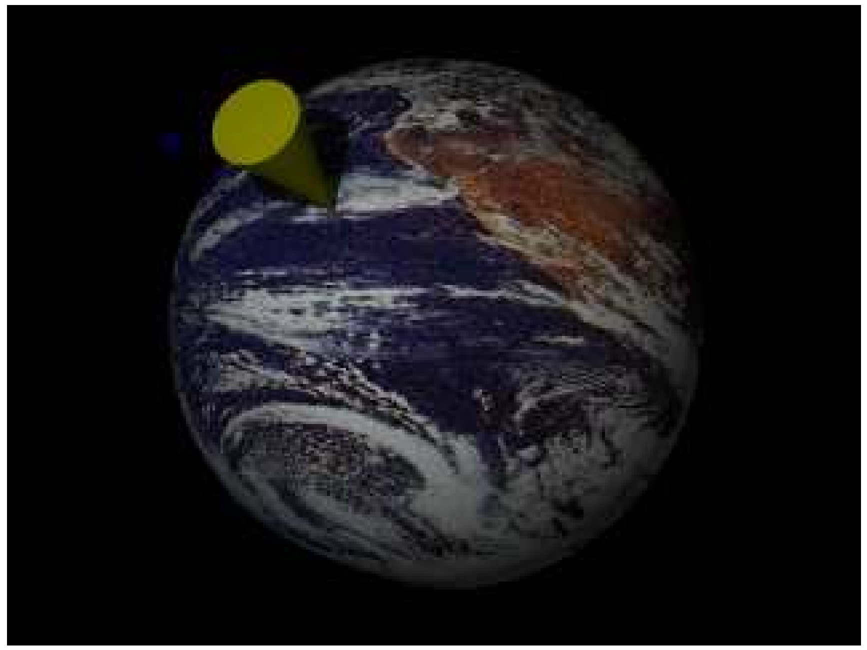

Figure 5: One frame from the home animation

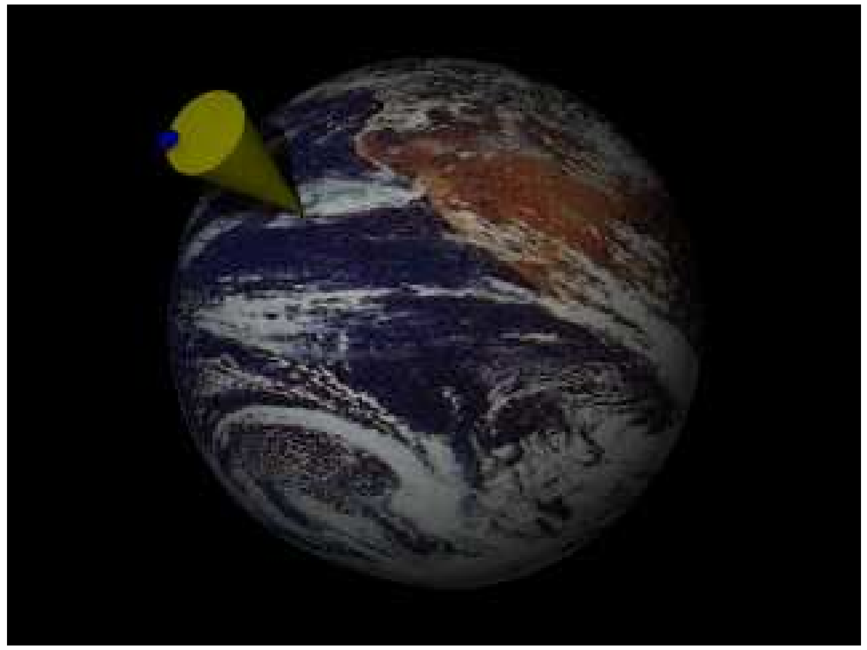

Figure 6: One frame from the home animation

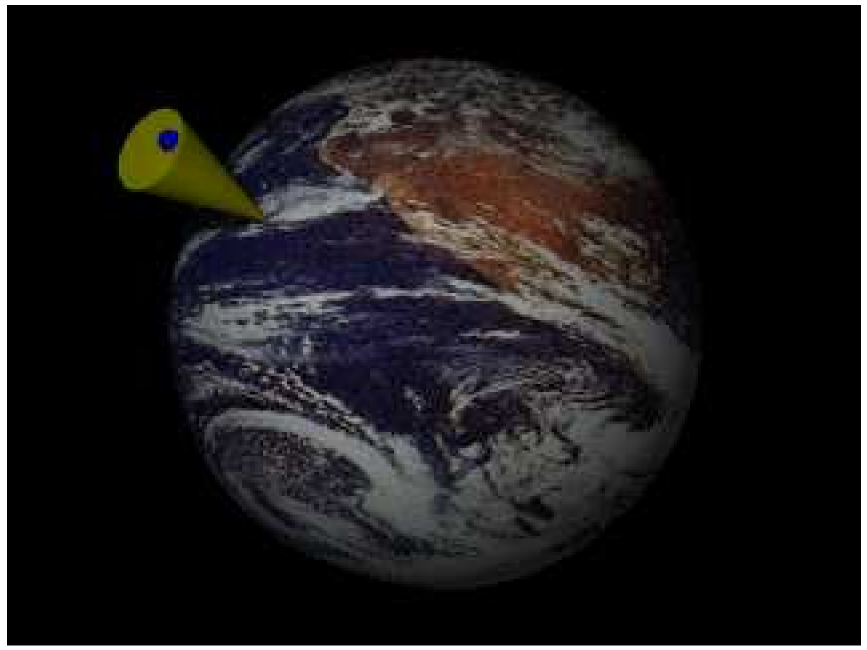

Figure 7: One frame from the home animation

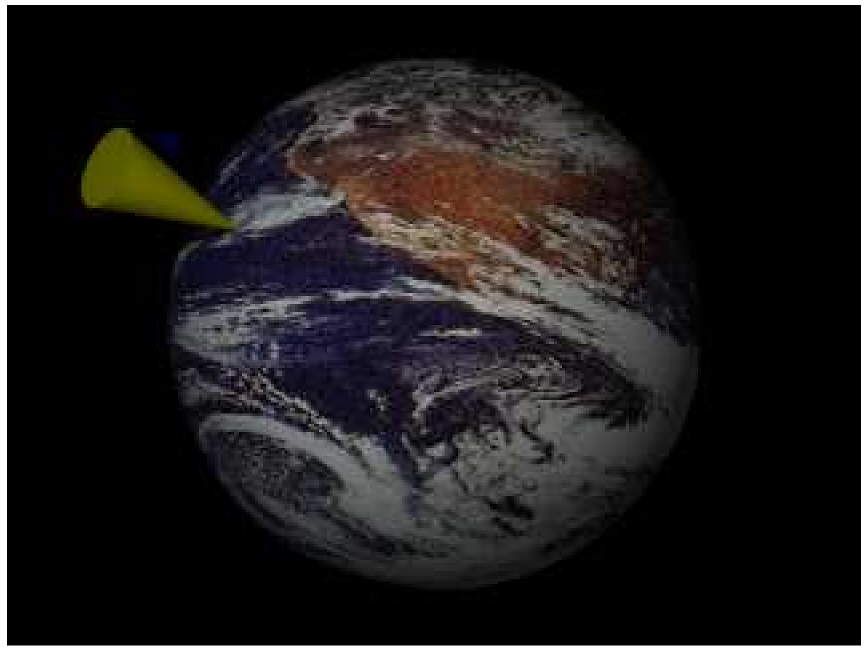

Figure 8: One frame from the home animation

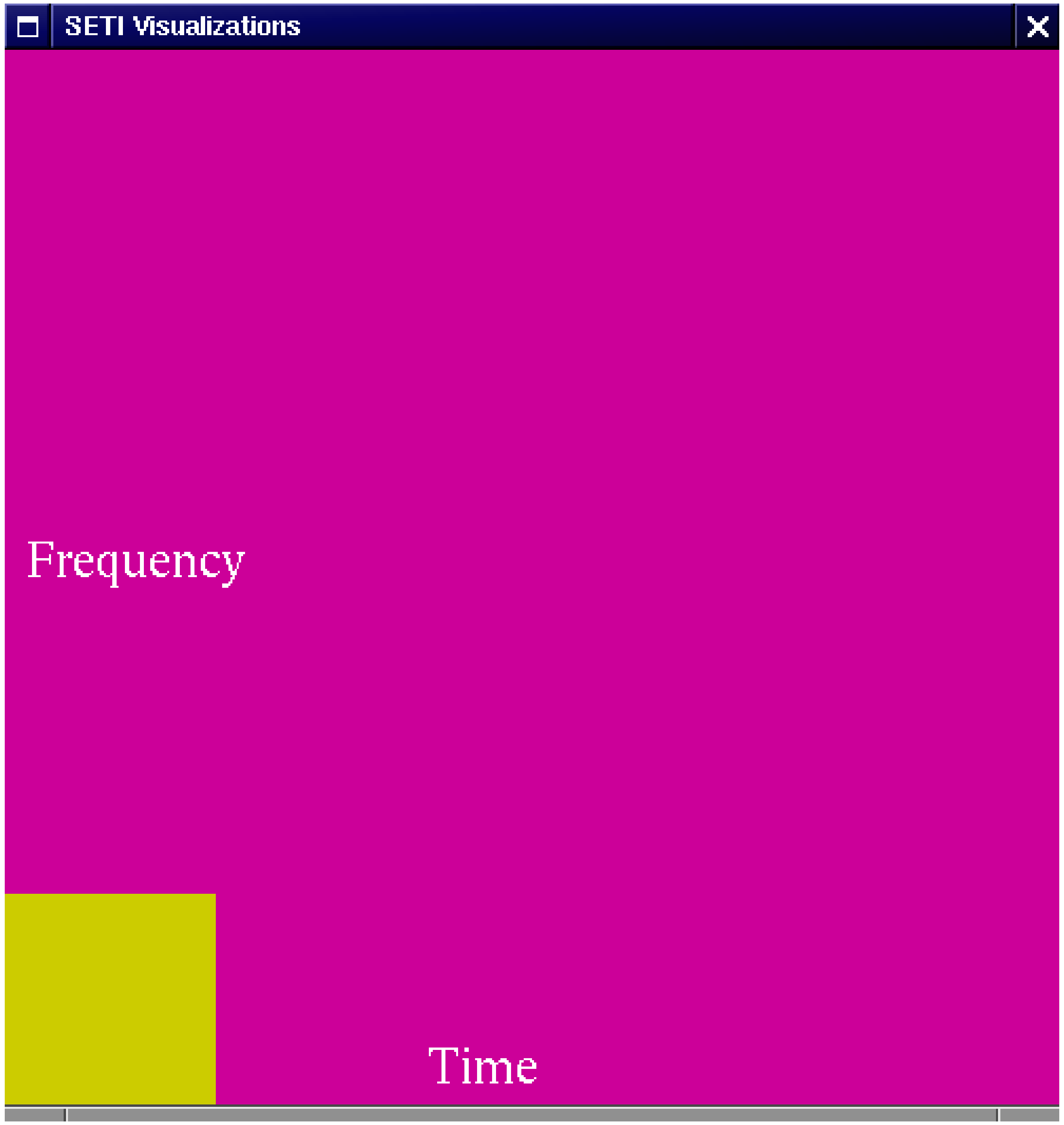

Figure 9: The frequency vs. time chart for breakup.

4 Results

Several utilities were developed. milky gives a graphical depiction of the Drake Equation. explorer shows how a colonizing alien civilization might develop. strategy shows the different strategies used by different SETI programs. home explains the gaussian waveform that SETI@home searches for. breakup shows how data is divided into work units. These should help SETI explain itself.

5 Conclusion

It is unlikely that any SETI organization would use SETI Visualizations utilities for its public information campaign. The quality of the programs is not an issue here. The programs need to be packaged like commercial software, with manuals and technical support. To provide extensive services, a small company might be formed, perhaps with seed money from interested SETI organizations. This company would continue to develop the programs to the level generally expected of commercial software.

References

[1] The SETI League, Inc., "General Information", 27 July 2002. http://www.setileague.org/general/general.htm (September 12, 2002)

[2] Jones, Douglas S., "Beyond the Drake Equation, 26 September 2001. http://www.station1.net/DouglasJones/drake.htm

[3] SETI Institute, "Frequently Asked Questions", 5 December 2002.
http://www.seti.org/faq.html (December 12, 2002)

[4] SETI@home, "Learn About SETI@home", 2 July 2002
http://setiathome.ssl.berkeley.edu/learnmore.html (February 25, 2003)

[5] Powell, Richard, "A Map of the Milky Way", 1 March 2003
http://www.anzwers.org/free/universe/index.html (April 8, 2003)

[6] Anderson, David P. et al., "SETI@home: An Experiment in Public-Resource Computin", November 2002
http://setiathome.ssl.berkeley.edu/cacm/cacm.html (April 8, 2003)

[7] NeHe Productions, "OpenGL Tutorials", 22 April 2003
http://nehe.gamedev.net/ (April 22, 2003)

[8] Sagan, Carl and Mayr, Ernst, "Can SETI Succeed", 1995
http://www.planetary.org/html/UPDATES/seti/Contact/debate/default.html (April 22, 2003)

[9] Sagan, Carl. Cosmos. Random House: New York. 1980.

[10] Korpela, Eric et al., "SETI@home: Massively Distributed Computing for SETI".
http://www.computer.org/cise/articles/seti.htm (May 29, 2003)

6 Appendices

A milky.cpp

```
//      SETI Visualizations

// by Immanuel Buder


#include <stdlib.h>

#include <iostream.h>

#include <GL/glut.h>

#include "tgaload.cpp"      //routine for loading texture from targa

#include <time.h>


int mainwindow,controlwindow, Rbutton, fpbutton, nebutton;

double R, fp, ne, fl, fi, fc, L;        //Drake factors

double P;                   //civilization density

GLint height, width; //window size

GLuint map;                 //texture, galaxy map

GLint cheight, cwidth;      //controol window size


void display ();            //main display function

void outtext (double x, double y, double z, char *string, void *font);
```

```
void drakeinit() {

    R= 10;          //known with some accuracy

    fp = .2;         //reasonable estimate

    ne = 1;             // good... if we are average

    fl=  1;         //optimistic

    fi = .9;        //optimistic

    fc = .1;        //blatant guess

    L = 1000;        //low side of Dolphin range

}


void loadtextures () {          //load texture

    glPixelStorei (GL_UNPACK_ALIGNMENT,1);

    glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);

    map = tgaLoadAndBind ("milky.tga",TGA_DEFAULT); //map identifies texture

}


void controldisplay () {

    glClear(GL_COLOR_BUFFER_BIT);       //reset to background

}
```

```
void Rdisplay () {
    glClearColor (.6,.1,0,0);           //background color
    glClear (GL_COLOR_BUFFER_BIT);      //set background
    glLoadIdentity();                   //put basic matrix into memory
    glOrtho (0,10,0,10,-1,1);           //dimensions of view
    glColor3f (0,0,0);                  //write in black
    outtext (1,3,0,"R",GLUT_BITMAP_TIMES_ROMAN_24);//label
    glFlush();                          //force drawing
}

void fldisplay () {
    glClearColor (.6,.1,0,0);
    glClear (GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glOrtho (0,10,0,10,-1,1);
    glColor3f (0,0,0);
    outtext (1,3,0,"fl",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush();
}
```

```
void fpdisplay () {

    glClearColor (.6,.1,0,0);

    glClear (GL_COLOR_BUFFER_BIT);

    glLoadIdentity();

    glOrtho (0,10,0,10,-1,1);

    glColor3f (0,0,0);

    outtext (1,3,0,"fp",GLUT_BITMAP_TIMES_ROMAN_24);

    glFlush();

}


void Ldisplay () {

    glClearColor (.6,.1,0,0);

    glClear (GL_COLOR_BUFFER_BIT);

    glLoadIdentity();

    glOrtho (0,10,0,10,-1,1);

    glColor3f (0,0,0);

    outtext (1,3,0,"L",GLUT_BITMAP_TIMES_ROMAN_24);

    glFlush();

}
```

```
void fidisplay () {
    glClearColor (.6,.1,0,0);
    glClear (GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glOrtho (0,10,0,10,-1,1);
    glColor3f (0,0,0);
    outtext (1,3,0,"fi",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush();
}

void fcdisplay () {
    glClearColor (.6,.1,0,0);
    glClear (GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glOrtho (0,10,0,10,-1,1);
    glColor3f (0,0,0);
    outtext (1,3,0,"fc",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush();
}

void Pdisplay () {
```

```
    glClearColor (.6,.1,0,0);
    glClear (GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glOrtho (0,10,0,10,-1,1);
    glColor3f (0,0,0);
    outtext (1,3,0,"P",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush();
}


void nedisplay () {
    glClearColor (.6,.1,0,0);
    glClear (GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glOrtho (0,10,0,10,-1,1);
    glColor3f (0,0,0);
    outtext (1,3,0,"ne",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush();
}

void nemouse (int button, int state, int x, int y) {
```

```
    if (state == GLUT_DOWN) {                   //if pressed

        cout<<"Enter a new value for ne, the number of earthlike planets per star."<<endl;

        cout<<"The old value is "<<ne<<endl;

        cout<<"This parameter is not well known; however, if the solar system is representative of other star systems, this number is between 1 and 5."<<endl;

        cin>>ne;

        glutPostWindowRedisplay(mainwindow);    //redraw main window

        cout<<"ne changed to "<<ne<<endl;       //inform user of change

        glFlush();                              //force drawing

    }

}




void Rmouse (int button, int state, int x, int y) {

    if (state == GLUT_DOWN) {

        cout<<"Enter a new value for R, the number of stars formed per year."<<endl;

        cout<<"The old value is "<<R<<endl;

        cout<<"This parameter is believed to be between 1 and 20"<<endl;

        cin>>R;

        cout<<"R changed to "<<R<<endl;

        glutPostWindowRedisplay(mainwindow);

        glFlush();
```

```
    }
}


void fcmouse (int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {
        cout<<"Enter a new value for fc, the fraction of intelligent beings that communicate"<<endl;
        cout<<"The old value is "<<fc<<endl;
        cout<<"This parameter is largely unknown.  Some guess it is between .1 and .2, since many intelligent beings do not develop technology (e.g. dolphins)."<<endl;
        cin>>fc;
        cout<<"fc changed to "<<fc<<endl;       //inform user of change


        glutPostWindowRedisplay(mainwindow);
        glFlush();
    }
}



void flmouse (int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {
        cout<<"Enter a new value for fl, the fraction of earthlike planets on which life occurs"<<endl;
        cout<<"The old value is "<<fl<<endl;
```

```
        cout<<"This parameter is assumed to be close to 1, given how quickly life evolved on earth"<<endl;
        cin>>fl;
        cout<<"fl changed to "<<fl<<endl;       //inform user of change

        glutPostWindowRedisplay(mainwindow);
        glFlush();
    }
}

void fimouse (int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {
        cout<<"Enter a new value for fi, the fraction of life-bearing planets which develop intelligence."<<endl;
        cout<<"The old value is "<<fi<<endl;
        cout<<"This parameter is believed to be close to 1, since intelligence has great survival value."<<endl;
        cin>>fi;
        cout<<"fi changed to "<<fi<<endl;       //inform user of change

        glutPostWindowRedisplay(mainwindow);
        glFlush();
    }
}
```

```
void Pmouse (int button, int state, int x, int y) {

    if (state == GLUT_DOWN) {

        cout<<"The current civilization density is "<<P<<endl;

    }

}



void Lmouse (int button, int state, int x, int y) {

    if (state == GLUT_DOWN) {

        cout<<"Enter a new value for L, the length of time (in years) a communicating civilization remains detectable."<<endl;

        cout<<"The old value is "<<L<<endl;

        cout<<"This parameter is guessed to be between 1,000 and 100,000,000"<<endl;

        cin>>L;

        cout<<"L changed to "<<L<<endl;       //inform user of change


        glutPostWindowRedisplay(mainwindow);

        glFlush();

    }

}
```

```
void fpmouse (int button, int state, int x, int y) {

    if (state == GLUT_DOWN) {

        cout<<"Enter a new value for fp, the fraction of stars with planets."<<endl;

        cout<<"The old value is "<<fp<<endl;

        cout<<"This parameter is believed to be between .2 and .5"<<endl;

        cin>>fp;

        cout<<"fp changed to "<<fp<<endl;       //inform user of change


        glutPostWindowRedisplay(mainwindow);

        glFlush();

    }

}



void initialize () {

    height = width = 500;                   //set size

    glutInitDisplayMode(GLUT_DEPTH|GLUT_SINGLE|GLUT_RGB);   //single buffered mode

glutInitWindowSize(width, height);

mainwindow = glutCreateWindow("SETI Visualizations");    //create main window

glutDisplayFunc(display);   //function for main drawing
```

```
    glEnable(GL_RGB);           //RGB mode

    glEnable (GL_TEXTURE_2D);       //textures on

    glEnable(GL_DEPTH_TEST);        //

    glDepthFunc(GL_LEQUAL);        // more texture stuff

    glEnable(GL_BLEND);             //

    glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA); //

    loadtextures();         //load texture from milky.tga

    glEnable(GL_RGB);


    glClearColor(.3,.3,.6,0);

glMatrixMode(GL_PROJECTION); //create view

glLoadIdentity();

glOrtho(-1,1,-1,1,-1,1);


    cwidth= 200; cheight = 200;     //control window size

    glutInitWindowSize(cwidth, cheight);              //control windoe

    controlwindow = glutCreateWindow("Controls");

    glutDisplayFunc(controldisplay);            //drawing function for controls

    glClearColor (1,1,1,1);

    glClear(GL_COLOR_BUFFER_BIT);           //set background

    glMatrixMode(GL_PROJECTION);
```

```
glLoadIdentity();

glOrtho(-1,1,-1,1,-1,1);

//create buttons

Rbutton = glutCreateSubWindow(controlwindow, 10, 10, 22, 40);

glutDisplayFunc (Rdisplay);             //display function for buttons

glutMouseFunc(Rmouse);              //function for mouse click


fpbutton = glutCreateSubWindow(controlwindow, 50, 10, 30, 40);

glutDisplayFunc (fpdisplay);

glutMouseFunc(fpmouse);


nebutton = glutCreateSubWindow(controlwindow, 90, 10, 30, 40);

glutDisplayFunc (nedisplay);

glutMouseFunc(nemouse);


glutCreateSubWindow(controlwindow, 130, 10, 20, 40);

glutDisplayFunc(fldisplay);

glutMouseFunc(flmouse);


glutCreateSubWindow(controlwindow, 160, 10, 20, 40);

glutDisplayFunc(fidisplay);
```

```
    glutMouseFunc(fimouse);

    glutCreateSubWindow(controlwindow, 10, 60, 30, 40);
    glutDisplayFunc(fcdisplay);
    glutMouseFunc(fcmouse);

    glutCreateSubWindow(controlwindow, 50, 60, 30, 40);
    glutDisplayFunc(Ldisplay);
    glutMouseFunc(Lmouse);

    glutCreateSubWindow(controlwindow, 90, 60, 30, 40);
    glutDisplayFunc(Pdisplay);
    glutMouseFunc(Pmouse);
}

void outtext (double x, double y, double z, char *string, void *font) {
glRasterPos3f(x,y,z); //locate position for output
int len = strlen (string); //find length of output
    for (int i = 0; i < len; i++) {
glutBitmapCharacter(font,string[i]); //output by chars
    }
```

```
}

void display () {
glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT); //clear

    glBindTexture(GL_TEXTURE_2D, map);  //use texture
    glColor3f (1.0,1.0,1.0);
    glBegin(GL_QUADS);      //square, background
    glTexCoord2f (0,0);     //point on texture
    glVertex3f (-1,-1,.1);  //point on screen
    glTexCoord2f (0,1);
    glVertex3f (-1,1,.1);
    glTexCoord2f (1,1);
    glVertex3f (1,1,.1);
    glTexCoord2f (1,0);
    glVertex3f (1,-1,.1);
    glEnd();                //done with quads

    glColor3f(1.0,1.,1.0);      //set draw color
    glBegin(GL_POINTS);         //begin drawing points
    glColor3f(1.0,0,0);      //set draw color
```

```
    double x,y, xstep, ystep, points;

    xstep = ystep = .005;       //dist between points

    points = 4/ (xstep * ystep);    //number of possible points on screen

    P = R*fp*ne*fl*fi*fc*L/points;      //current point density

    if (P >= 1) cout<<"Warning: maximum saturation reached"<<endl;

    for (x = -1; x <1; x+= xstep)       //move across screen

        for (y = -1; y < 1; y += ystep)

            if (P > (double(rand())/double(RAND_MAX)))

                glVertex3f(x,y,0.2);        //draw point

    glEnd();        //done with points

    glFlush(); //force drawing

}


int main (int argc, char ** argv) {

srand(unsigned(time(NULL)));    //seed random numbers

    drakeinit();                //initialize Drake factors

    glutInit(&argc, argv); //initialize graphics

    initialize();               //more graphics initialization

glutMainLoop();             //hand control over to OpenGL

return 0;
```

```
}
```

B explorer.cpp

```
//      SETI Visualizations

// by Immanuel Buder


#include <stdlib.h>

#include <iostream.h>

#include <GL/glut.h>

#include "tgaload.cpp"      //routine for loading texture from targa

#include <time.h>


#define occupied 1;         //codes for data

#define vacant 0;

#define spawn 2;


int mainwindow,controlwindow,Twindow;       //window id numbers

GLint height, width; //window size

GLuint map;                 //texture, galaxy map

GLint cheight, cwidth;      //controol window size

long long count;            // number of run stages
```

```
int mode;                   //alien type

int pos[501][501];          //what at each point

void display ();            //main drawing function (prototype)
void outtext (double x, double y, double z, char *string, void *font); //not used

void move1 (int x, int y) {             //explorint function for type 1 alien
                int dir = rand() % 4;   //random mvmt
                switch (dir) {
                    case 0:
                        if (pos[x-1][y] == 0) {         //if no one there
                            pos[x-1][y] =pos[x][y] = spawn;     //go
                            break;
                        }               //if can't go, move to next
                    case 1:
                        if (pos[x+1][y] == 0) {
                            pos[x+1][y] = pos[x][y] = spawn;
                            break;
                        }
                    case 2:
```

```
                if (pos[x][y-1] == 0) {
                    pos[x][y-1]=pos[x][y] = 2;
                    break;
                }
            case 3:
                if (pos[x][y+1] == 0) {
                    pos[x][y+1] = pos[x][y] = spawn;
                    break;
                }
        }
}

void move2(int x, int y) {
        int dir = rand() % 4;
        switch (dir) {
            case 0:
                if (pos[x-1][y] == 0) {
                    pos[x-1][y] =pos[x][y] = spawn;
                }
                break;          //stop even if can't go
            case 1:
```

```
                if (pos[x+1][y] == 0) {
                    pos[x+1][y] = pos[x][y] = spawn;
                }
                break;
            case 2:
                if (pos[x][y-1] == 0) {
                    pos[x][y-1]=pos[x][y] = 2;
                }
                    break;


            case 3:
                if (pos[x][y+1] == 0) {
                    pos[x][y+1] = pos[x][y] = spawn;
                }
                    break;


        }
}


void move3(int x, int y) {
        switch (0) {            //always 0 direction first
```

```
    case 0:
        if (pos[x-1][y] == 0) {
            pos[x-1][y] =pos[x][y] = spawn;
            break;
        }   //next direction if this one fails
    case 1:
        if (pos[x+1][y] == 0) {
            pos[x+1][y] = pos[x][y] = spawn;
            break;
        }
    case 2:
        if (pos[x][y-1] == 0) {
            pos[x][y-1]=pos[x][y] = 2;
            break;
        }
    case 3:
        if (pos[x][y+1] == 0) {
            pos[x][y+1] = pos[x][y] = spawn;
            break;
        }
}
```

```
}

void move4 (int x, int y) {

                int dir = rand() % 4;
                switch (dir) {
                    case 0:
                        if (pos[x-1][y] == 0) {
  /* coin toss to stop */   if (rand() % 2) pos[x-1][y] =pos[x][y] = spawn;
                            break;
                        }   //next direction if can't go
                    case 1:
                        if (pos[x+1][y] == 0) {
                            if (rand() % 2) pos[x+1][y] = pos[x][y] = spawn;
                            break;
                        }
                    case 2:
                        if (pos[x][y-1] == 0) {
                            if (rand() %2) pos[x][y-1]=pos[x][y] = 2;
                            break;
                        }
```

```
                case 3:
                    if (pos[x][y+1] == 0) {
                        if (rand() % 2) pos[x][y+1] = pos[x][y] = spawn;
                        break;
                    }
            }
}


void loadtextures () {          //load texture
    glPixelStorei (GL_UNPACK_ALIGNMENT,1);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);
    map = tgaLoadAndBind ("milky.tga",TGA_DEFAULT);
}

void mouse (int button, int state, int x, int y) {
    if (state == GLUT_DOWN && button == 2) {        //if right click
        cout<<count<<" ticks"<<endl;        // say time before exit
        exit(0);
    }
}
```

```
/*
void Tdisplay () {                  //old control routine
    glClearColor (.6,.1,0,0);       //no longer used
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glOrtho(0,10,0,10, -1,1);
    glColor3f(1.0,1.0,1.0);
    outtext(1,3,0,"T",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush();

}

void Tmouse(int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {
        cout<<"The current time is "<<count<<endl;
    }
}

void controldisplay () {
    glClear(GL_COLOR_BUFFER_BIT);
```

```
}

*/

void initialize () {

    height = width = 500;                   //set size

    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGB);   //OpenGl mode

glutInitWindowSize(width, height);

mainwindow = glutCreateWindow("SETI Visualizations");    //create main window

glutDisplayFunc(display);       //set display function

    glutMouseFunc(mouse);           //set mouse function

    glEnable(GL_RGB);               //OpenGL options

    glEnable (GL_TEXTURE_2D);

    glEnable(GL_DEPTH_TEST);

    glDepthFunc(GL_LEQUAL);

    glEnable(GL_BLEND);

    glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);

    loadtextures();

    glEnable(GL_RGB);


    glClearColor(.3,.3,.6,0);       //background color

glMatrixMode(GL_PROJECTION); //create view

glLoadIdentity();               //
```

```
glOrtho(1,500,1,500,-1,1);      //drawspace size


//    cwidth= 200; cheight = 200;  //no longer used

    glutInitDisplayMode(GLUT_DEPTH|GLUT_SINGLE|GLUT_RGB);


    glutInitWindowSize(cwidth, cheight);              //control windoe

/*    controlwindow = glutCreateWindow("Controls");

    glutDisplayFunc(controldisplay);        //no longer used

    glClearColor (1,1,1,1);

    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();

    glOrtho(-1,1,-1,1,-1,1);


    Twindow = glutCreateSubWindow(controlwindow, 10, 10, 40, 40);

    glutDisplayFunc(Tdisplay);

    glutMouseFunc(Tmouse);*/

}


void outtext (double x, double y, double z, char *string, void *font) {

glRasterPos3f(x,y,z); //locate position for output
```

```
int len = strlen (string); //find length of output
    for (int i = 0; i < len; i++) {
glutBitmapCharacter(font,string[i]); //output by chars
    }
}

void display () {
glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT); //clear

    glBindTexture(GL_TEXTURE_2D, map);  //use texture
    glColor3f (1.0,1.0,1.0);            //draw color -- doesn't work
    glBegin(GL_QUADS);      //square, background
    glTexCoord2f (0,0);         //coordinate in texture file
    glVertex3f (1,1,.1);        //coordinate on screen
    glTexCoord2f (0,1);
    glVertex3f (1,500,.1);
    glTexCoord2f (1,1);
    glVertex3f (500,500,.1);
    glTexCoord2f (1,0);
    glVertex3f (500,1,.1);
    glEnd();                    //done with quads
```

```
glColor3f(1.0,1.,1.0);      //set draw color -- does not work

glBegin(GL_POINTS);         //draw points

glColor3f(1.0,0,0);      //set draw color -- does not work


int x,y;                                //

for (x = 1; x <= 500; x++)              //move across screen

    for (y = 1; y <= 500; y++)          //

        if (pos[x][y] == 1)

            glVertex3f(x,y,.1);     //plot aliens


for (x = 2; x<=499; x++)

    for (y=2; y <= 499; y++)

        if (pos[x][y] == 1) {


            if (mode == 1) move1(x,y);  //do movement

            if (mode == 2) move2(x,y);

            if (mode == 3) move3(x,y);

            if (mode == 4) move4(x,y);
```

```
            }
    for (x = 1; x <= 500; x++)
        for (y = 1; y<=500; y++){
            if (pos[x][y] == 2)             //update spawnms
                pos[x][y] = occupied;
            for (long long j = 0; j < 5; j++);      //waste time
        }
            glEnd();            //done with points
    glFlush(); //force drawing
    count++;                //count updates
    glutSwapBuffers();          //animate
//    glutPostWindowRedisplay(Twindow);
}

int main (int argc, char ** argv) {
```

```
    if (argc != 2) {
        cout<<"FATAL ERROR.  Incorrect number of arguments.  You die."<<endl;
        exit(0);
    }
    mode = strtol(argv[1],NULL,10);      //read in mode
    srand(unsigned(time(NULL)));         //seed random
    for (int x = 0; x <= 500; x++)
        for (int y = 0; y <= 500; y++)
            pos[x][y] = vacant;          //clear galaxy
    pos[(rand()%500)+1][(rand()%500)+1] = occupied;     //random start
    count = 0;                  //initialize counter
    glutInit(&argc, argv); //initialize graphics
    initialize();
    glutIdleFunc(display);      //
    glutMainLoop();             //hand control to OpenGL
    cout<<count;
    return 0;
}
```

C strat1.pov

```
camera {
```

```
    location <-1,-2,10>         //where are you
    look_at <0,0,0>             //where are you looking
}

light_source {
    <-1,-2,11>                  //where is light
    color rgbf<1,1,1>           //what color light
}

sphere {
    <0,0,0>,5                   //where is sphere, size
    pigment {
        image_map {
            gif "globe.gif"     //map texture
            map_type 1          //how texture is applied to surface
            interpolate 0
        }
    }
}

cone { <0,0,7>.25,              //cone base and radius
```

```
        <0,0,5>,0               //cone point
        pigment {color rgbf <1,0,0,.5>} //color

        }

cone { <-1,-2,6>, .25,
        <-1,-1,4>,0
        pigment {color rgbf<1,0,0,.5>}
    }
```

D strat2.pov

```
camera {
    location <-1,-2,10>     //where are you
    look_at <0,0,0>         //where are you looking
}

light_source {
    <-1,-2,11>              //where is light
    color rgbf<1,1,1>       //what color light
}
```

```
sphere {

    <0,0,0>,5               //sphere location + size

    pigment {

        image_map {

            gif "globe.gif"     //map texture to sphere

            map_type 1          //how texture is applied

            interpolate 0

        }

    }

}


cone { <.5,.5,5.5>.75,          //cone base and radius

        <0,0,5>,0               //cone point

        pigment {color rgbf <0,1,1,.5>}     //color


        }


cone { <-1,-2,6>, .5,

        <-1,-1,4>,0

        pigment {color rgbf<0,1,1,.5>}

    }
```

```
cone { <1.5,0.5,5.5>, .75,
        <1,0,5>,0
        pigment {color rgbf<0,1,1,.5>}
    }

cone { <.5,-3,5>, .75,
        <0,-2,4.5>,0
        pigment {color rgbf<0,1,1,.5>}
    }
```

E home.pov

```
camera {
    location <-3,-4,12>         //where are you
    look_at <0,0,0>             //where are you looking
}

light_source {
    <-1,-2,11>                  //where is light
    color rgbf<1,1,1>           //what color light
}
```

```
sphere {
    <0,0,0>,5                   //where is sphere, size
    pigment {
        image_map {
            gif "globe.gif"     //map texture
            map_type 1          //how texture is applied to surface
            interpolate 0
        }
        rotate <0, clock*20/16,0>   //spin with time


    }
}


cone { <0,0,7>.5,              //cone base and radius
        <0,0,5>,0               //cone point
        pigment {color rgbf <1,1,0,.3>} //color
        rotate <0, clock*20/16,0>


        }


sphere { <0,0,7>, .125
```

```
    pigment {color rgbf <0,0,1,1-sin(clock * 3.1415926535/16)>}   //change brightness
        rotate<0,8*20/16,0>
        }
```

F breakup.cpp

```
//      SETI Visualizations
// by Immanuel Buder

#include <stdlib.h>
#include <iostream.h>
#include <GL/glut.h>

double f,t;
int mainwindow;       //window id numbers
GLint height, width; //window size

void display ();            //main drawing function (prototype)
void outtext (double x, double y, double z, char *string, void *font); //not used

void initialize () {
    height = width = 500;                   //set size
```

```
    glutInitDisplayMode(GLUT_RGB);   //OpenGl mode
glutInitWindowSize(width, height);
mainwindow = glutCreateWindow("SETI Visualizations");    //create main window
glutDisplayFunc(display);       //set display function

    glClearColor(.3,.3,.6,0);       //background color
glMatrixMode(GL_PROJECTION); //create view
glLoadIdentity();               //
glOrtho(0,500,0,2500,-1,1);      //drawspace size

//    cwidth= 200; cheight = 200;  //no longer used
    glutInitDisplayMode(GLUT_DEPTH|GLUT_SINGLE|GLUT_RGB);

/*    controlwindow = glutCreateWindow("Controls");
    glutDisplayFunc(controldisplay);        //no longer used
    glClearColor (1,1,1,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1,1,-1,1,-1,1);
```

```
    Twindow = glutCreateSubWindow(controlwindow, 10, 10, 40, 40);

    glutDisplayFunc(Tdisplay);

    glutMouseFunc(Tmouse);*/

}


void outtext (double x, double y, double z, char *string, void *font) {

glRasterPos3f(x,y,z); //locate position for output

int len = strlen (string); //find length of output

    for (int i = 0; i < len; i++) {

glutBitmapCharacter(font,string[i]); //output by chars

    }

}


void display () {

glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT); //clear

glColor3f(.8,0,.6);             //color

    glBegin(GL_QUADS);              //draw background square

    glVertex3f(0,0,0);

    glVertex3f(0,2500,0);

    glVertex3f(500,2500,0);

    glVertex3f(500,0,0);
```

```
    glColor3f(.8,.8,0);
    glVertex3f(0,0,0);              //draw foreground rectangle
    glVertex3f(0,f,0);
    glVertex3f(t,f,0);
    glVertex3f(t,0,0);

    glEnd();                //done drawing quads

    glColor3f(1,1,1);
    outtext(200,50,0.1,"Time",GLUT_BITMAP_TIMES_ROMAN_24);  //label axes
    outtext(10,1250,.1,"Frequency",GLUT_BITMAP_TIMES_ROMAN_24);
    glFlush(); //force drawing
    cout<<"The current frequency width is "<<f<<" kHz"<<endl
        <<"The current time width is "<<t<<" s"<<endl
        <<"The current data size is "<<2*f*t<<" Kb"<<endl;
    cout<<"Enter new (f,t) in (kHz,s)"<<endl;
    cin>>f>>t;
    glutPostRedisplay();        //redraw after input
    //    glutPostWindowRedisplay(Twindow);
}
```

```
int main (int argc, char ** argv) {

    f = 10;         //initialize values = SETI@home current

    t=107;

    initialize();               //OpenGL stuff

    glutMainLoop();             //hand control to OpenGL

    return 0;

}
```