Stanford Exploration Project, Report 89, November 12, 1997, pages 227–??

Stanford Exploration Project, Report 89, November 12, 1997, pages 227–??

Interactive web pages with Java

Ken Lenga, Robert G. Clapp, and Jon F. Claerbout[1]

keywords: *interactive, algorithm, graphics*

ABSTRACT

Recently, Sun introduced Java, a new programming language with machine independent object files. Java, and the Java Browser, provides the next step in web browser technology by bringing over machine independent byte codes and interpreting them on the local machine rather than the host machine. This capability opens up a new realm of possibilities in information transfer and interactive displays. This summer SEP tested Java's capabilities by developing an interactive web-based demo of wave propagation, a frame viewer (similar to SEP's old Animator), and a cube viewer.

INTRODUCTION

SEP's purpose is to deliver fundamental research in reflection seismology to it's sponsors, and to some extent, the world at large. In the last few years SEP has attempted to use interactive CD-ROMs to facilitate this information passing (Claerbout and Schwab, 1995). This effort has proven to be somewhat successful, but has suffered from the requirement that the user must have one of our supported platforms (Sun, HP, and RS600). In today's world of powerful PC's and Macs this eliminates numerous possible users. In addition, SEP has suffered from a lack of interactivity in it's in-house plotting software vplot. Vplot has proven to be a powerful tool but is outdated. Java , a new programming language introduced by Sun in March, offers a potential solution to both these problems. To test the power of Java as an information passing and visualization tool we attempted to use Java to do what SEP tools have had trouble accomplishing. We used the Java language and Java Browser to: create an interactive demonstration of wavefields propagating through a medium; a web based slice viewer, similar to the old-style SEP `Animator`; and finally a `Cube Viewer`.

[1]**email:** krl@sep.stanford.edu, bob@sep.stanford.edu, jon@sep.stanford.edu

JAVA LANGUAGE

The Java language represents a giant step in web browsing technology and possibly the next revolution in object oriented programming. Currently Java is only available on Sun computers running Solaris and IBM PC's but in the future should be available on most platforms.

Machine Independent/ Java Browser

The major innovation of the Java language is its architecture neutral object files that can be interpreted on machines other then the type on which it was compiled. This feature allows a programmer to write code that is completely portable. The Java Browser, available at , currently provides the major link between platforms (Netscape plans to incorporate the Java technology in future versions). The machine independent nature of Java provides:

- Improved technology transfer: programs can be evaluated immediately. For SEP, even when a sponsor had one of our supported computer platforms, their first opportunity to see our latest research in a non-static form was after our semi-annual CD-ROM release.
- Cooperation: People working on completely different platforms can work together without the time consuming task of converting libraries and debugging incompatibility problems.
- Computer resource savings: The server is not bogged down by running applications locally (previously the only possible way to simulate the interactivity of our CD-ROMs over the web).

Object Oriented

Object oriented programming has many features that appeal to scientists: clear representation of purpose, reusable tool kits, etc.(Nichols et al., 1993). The speed issue, along with the intricacies in programming the C++ has slowed its acceptance into general use in the scientific community. Java has attempted to find some middle ground between the power of C++ and the ease of Fortran.

Java has followed most of C++ conventions (inheritances, public/private functions, etc.) but has broken away from C++ in some key areas that commonly cause programming bugs. Java's new features include

- automatic garbage collection: allocation and freeing of memory is done automatically, avoiding memory leakage problems.

- disallowing multiple inheritances: something that is sometimes useful, but more often causes headaches.

- removes (hides) pointers: referencing of arrays, even multi-dimensional arrays are done with Fortran style direct referencing, eliminating over-writing of memory, a common bug in C and C++ code that is difficult to track down.

FROG POND

As a first test of Java's capabilities we wanted to see how well it could handle a problem that SEPLIB could not do adequately. Therefore we simulated what a pond would look like when a frog jumping from lily pad to lily pad. The goal of the demo is to simulate wavefront propagation through a medium with a given viscosity at a given velocity. The algorithm is a simple finite difference program where a Laplacian template is applied to the data. Using Ratfor the coding is short (see Appendix A), and with a knowledge of the called subroutines, straightforward. The problem lies with our SEPLIB libraries which do not allow interactivity. The location of where the frog jumps, the source position of the wavefront, is determined by a random function (see Figure 1). In addition, the only possible way to display the results over the Internet is through pre-calculating the images, and saving them into a mpeg file.

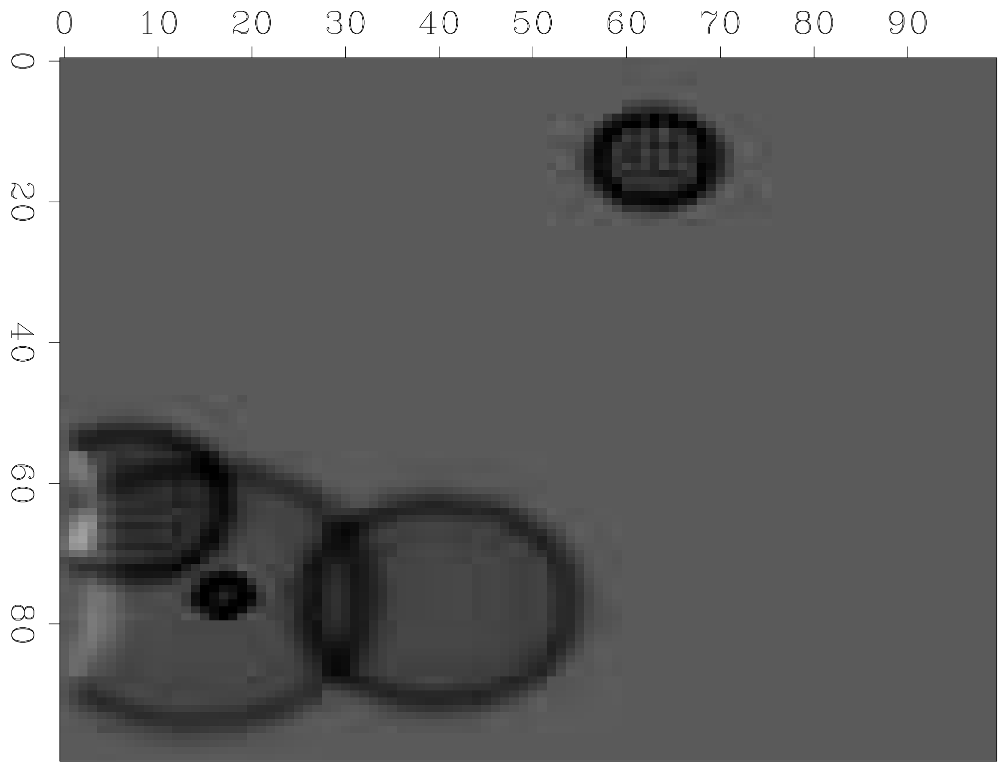

Figure 1: The Frog Pond, SEPLIB style. Select figure interaction to see a movie version. bob2-frogsep [ER]

Java opens up an entire new realm of possibilities (see Appendix A). By writing it as an applet, Java's convention for a web-based application, the demo becomes dynamic and interactive. By clicking on the pond the frog's course, and therefore the source of new wavefronts, can be guided. In addition features like the clip, the size of the display, and pausing the calculations become possible. Figure 2 shows the display in the Hot Java Browser.

Overall our initial test proved successful. The one drawback is speed. The machine independent byte codes appear to run at about 10% of the speed of the Fortran version.

ANIMATOR

The advent of the web has opened up new opportunities for researcher's to present their ideas, in both written and graphical forms. SEP's initial efforts to take advantage of this new medium was with text and either static gif images, or pre-created mpeg movies. This has proved effective, but extremely limiting. In addition, SEP's software to display images, vplot, suffers from a lack of interactivity.

The `Animator` offers a solution to both these problems. The Animator (see Figure 3) can display a series of images like vplot, but offers additional features:

- selective frame viewing - the user can go through the series of frames provided and choose a subset to view in a movie format.
- interactive windowing - a portion of the display can be zoomed in on.

The `Animator` reads in byte data from a SEPJava (see Appendix B) file, and creates images from this data. As each image is created, it is displayed and a graph at the bottom of the applet is updated. When all images have been created, the Animator enters an edit mode, in which you can create a custom movie by manipulating several parameters via mouse commands or keystrokes. A more detailed description can be found in Appendix C.

CUBE VIEWER

As more and more of SEP's research moves into the 3-D , 2-D viewers are no longer sufficient. Often the researcher can gain new insight by looking at orthogonal planes of the data simultaneously, and having the ability to run through the 3-D cube along any of the three planes. As a result SEP has developed the in-house 3-D viewers, `X11movie` and `Cubeplot`, that provide this extra level of sophistication. These programs have proven immensely useful, but they:

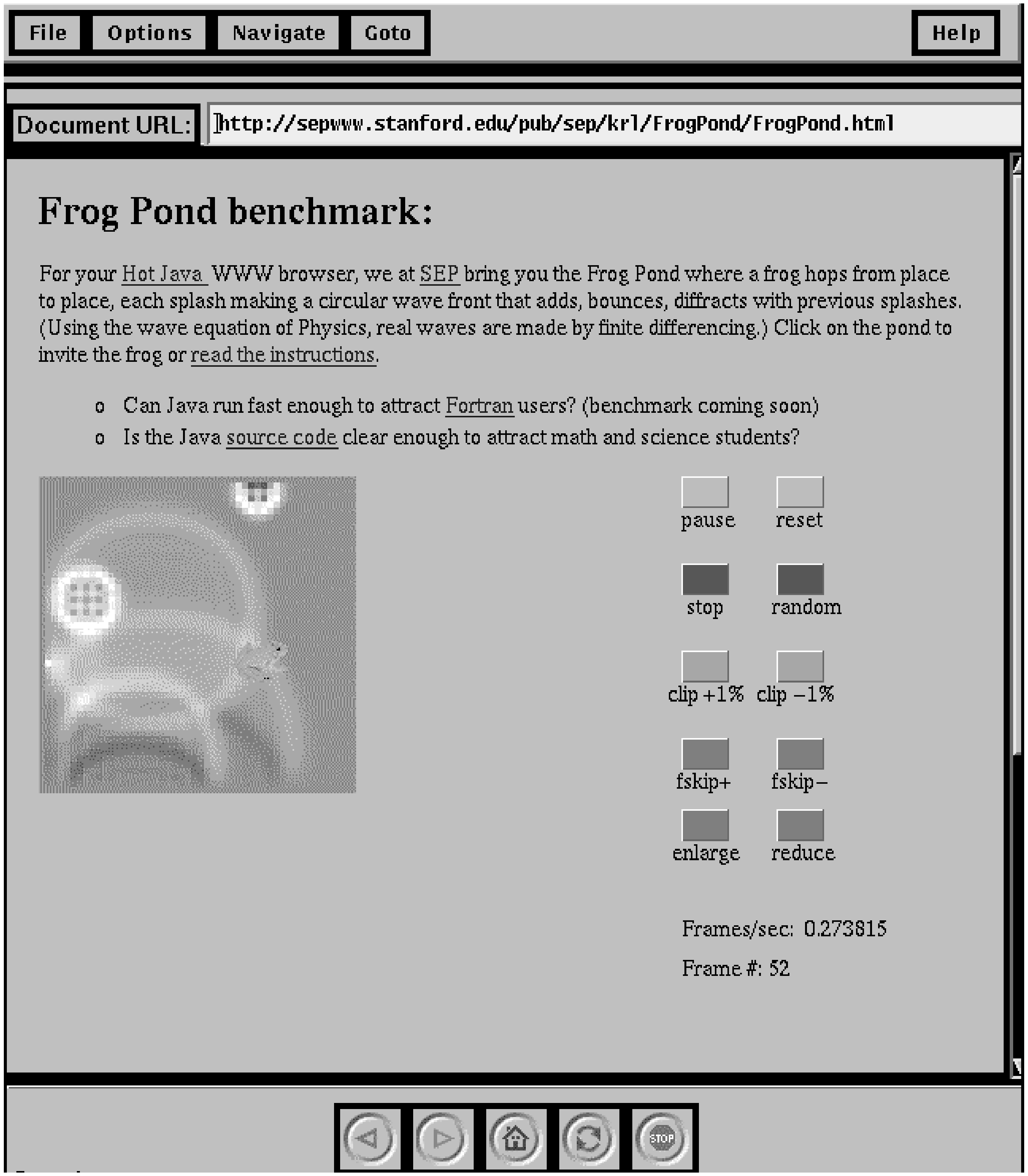

Figure 2: The Frog Pond, wherever the frog jumps, wavefronts emerge. If you have Java capabilities, click on the figure interact button to go to the Frog Pond. bob2-frogjava [NR]

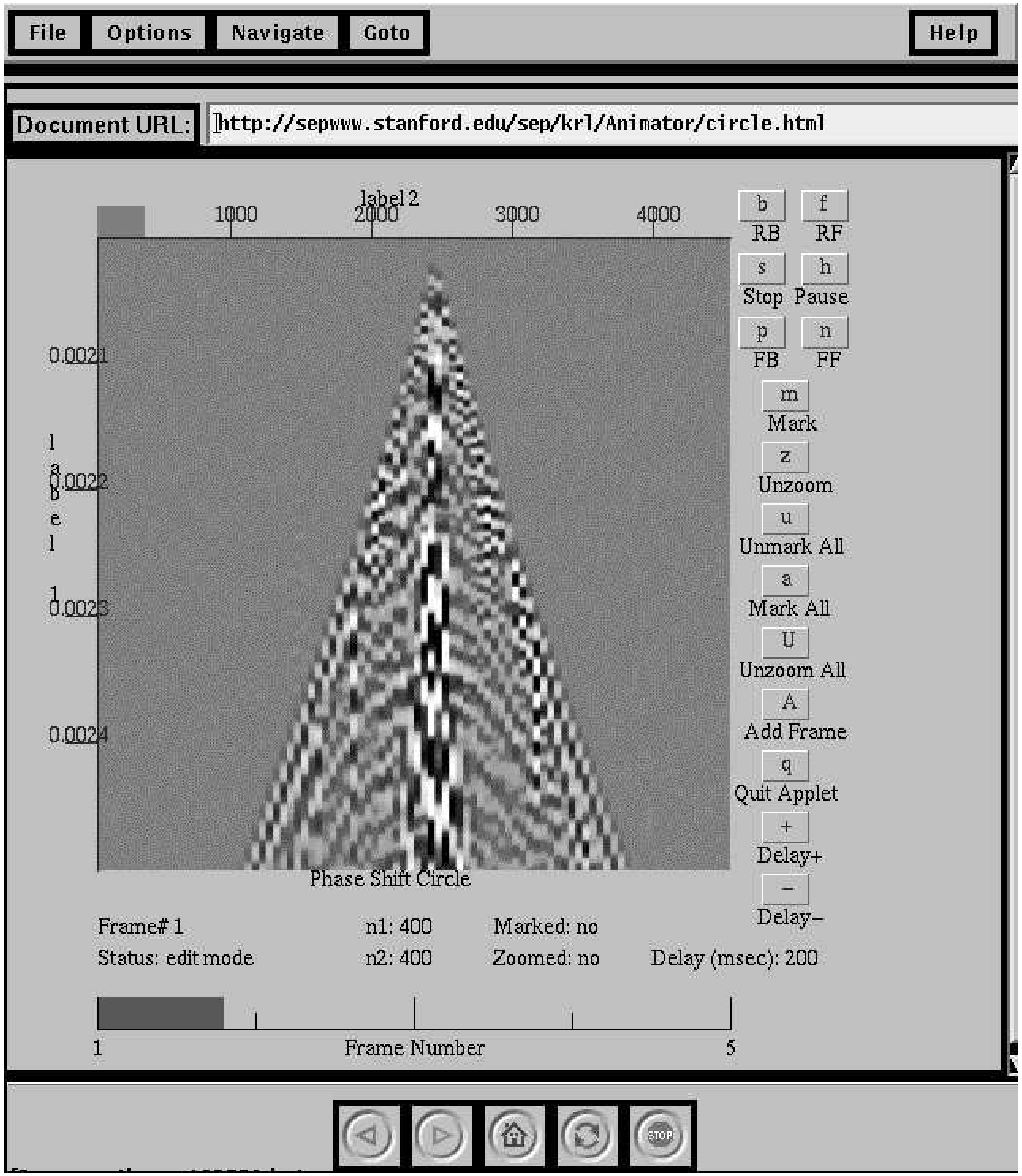

Figure 3: The animator showing a frame. If you have Java capabilities, click on the figure interact button to go to the `Animator`. bob2-animjava [NR]

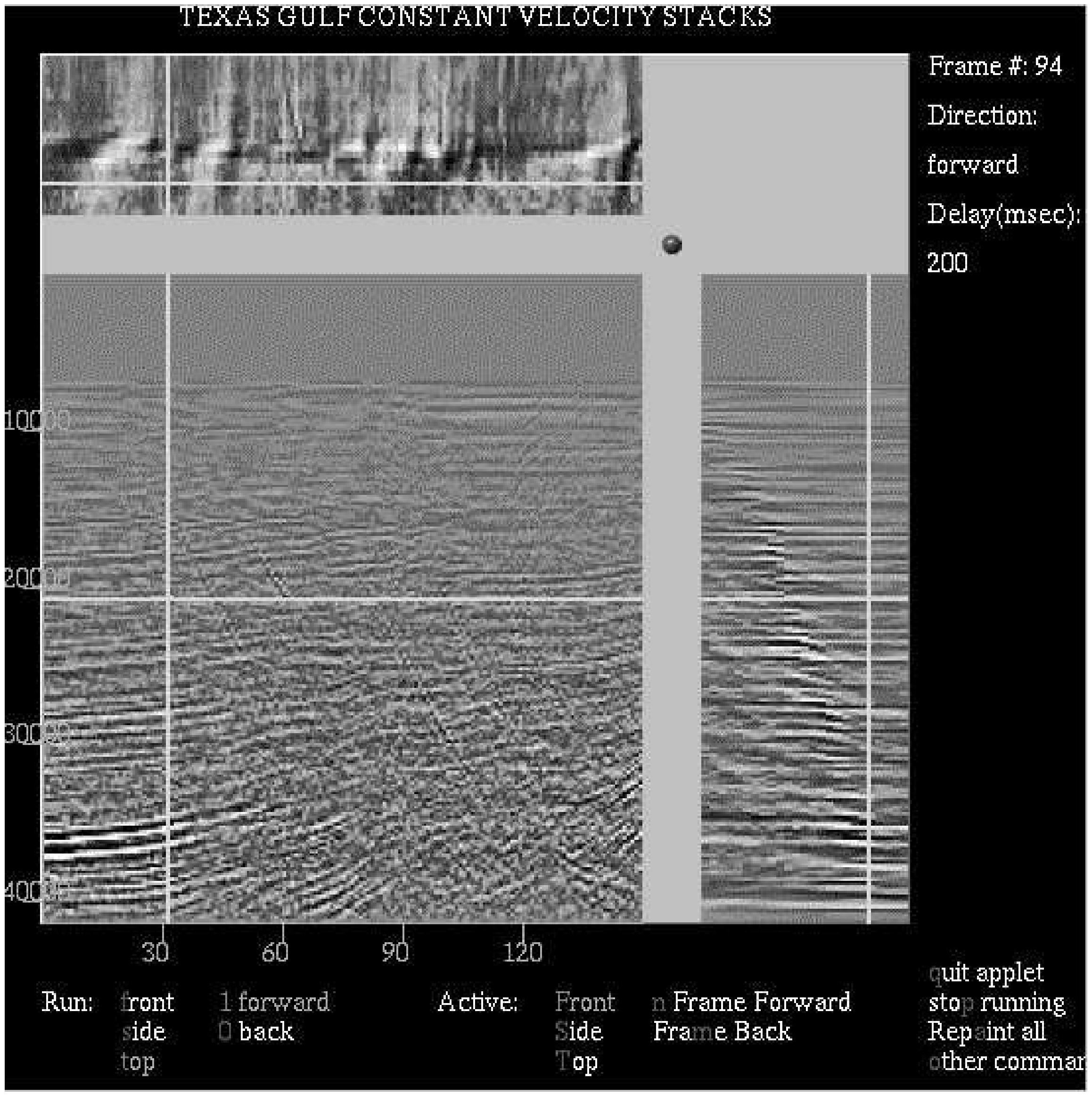

Figure 4: The `Cube Viewer` showing a portion of wesvel. The cube shows the change in a section as a function of velocity. If you have Java capabilities, click on the figure interact button to go to the `Cube Viewer`. bob2-cubejava [NR]

- can not be run on the web without putting considerable strain on the host machine.
- suffer from
  - a lack of interactivity in the case of `Cubeplot`
  - difficult menu controls in the case of X11movie

The `Cube Viewer` solves these problems. The `Cube Viewer` works along the same principle as the `Animator`. It loads in a SEPJava file one frame of the cube at a time, creating the front image during loading. Once the entire cube has been loaded, the side and top images are created. The display (Figure 4) of the `Cube Viewer` follows closely `Cubeplot`. The main screen is broken into three portions:

- Bottom left, the front view, constant i3 (slowest direction of the array).
- Bottom right, the side view, constant i2.
- Top left, the front view, constant i1 (fastest direction of the array).

On each displayed frame two yellow lines show the relative location of the other two displayed planes. The user can then use a series of key and mouse controls to navigate through the cube. Appendix D provides a user's guide to the `Cube Viewer`.

CONCLUSIONS

Our initial tests with Java were successful. It has proven easier to program in than C++. In addition, the interactive web abilities are unmatched by any other tool or language currently available. The major drawback is speed. The 5 to 10:1 ratio when compared to Fortran and C, needs to be improved. Hopefully, in future additions this problem can be somewhat remedied.

FUTURE WORK

Our ultimate goal is to be able to put our semi-annual SEP reports on the web. With the `Animator` and the `Cube Viewer` we have proven that at least at the graphics end this is possible. In the next year we hope to begin integration of the new graphical tools into our everyday working environment. In the future we will test the applicability of writing geophysical code in Java. Several question still remain:

- Is Java speed sufficient?
- Is the programming language clear enough to attract new users?
- Will Java's object oriented approach open up new opportunities in programming?

REFERENCES

Claerbout, J., and Schwab, M., 1995, CD-ROM versus The Web: SEP–**84**, 193–**??**.

Nichols, D., Urdaneta, H., Oh, H. I., erbout, J. C., Laane, L., Karrenbach, M., and Schwab, M., 1993, Programming geophysics in C++: SEP–**79**, 313–471.

APPENDIX A

FROG POND CODE

Fortran Version

```
#       Throw pebbles into pond.  Watch waves.     n=pond_size
#%
from par: integer n1:n=100, n2:n=n, n3:nframes=150
subroutine splash( n,nframes)
integer  i,j,k,    n,nframes,   iseed
real    velocity, viscosity, rand01
temporary real    now(n,n), old(n,n), new(n,n), laplacian(n,n), sum(n,n)
velocity= .20
iseed = 1995                                    # Random numbers for stone toss.
viscosity = .15
do i=1,n {  do j=1,n {  now(i,j)=0;  old(i,j)=0;  new(i,j)=0 }}
do k=1, nframes {
        i = 1 + rand01(iseed)*n
        j = 1 + rand01(iseed)*n                          # To a random location
        if( mod( k-1,10)==0)   now(i,j) = now(i,j)+1.    #       throw a pebble.
        do i=1,n
        do j=1,n
                   sum(i,j) = now(i,j) - viscosity * old(i,j)
        do i=2,n-1
        do j=2,n-1
                laplacian(i,j) = sum(i-1,j  ) - 2*sum(i,j) + sum(i+1,j  ) +
                                 sum(i  ,j-1) - 2*sum(i,j) + sum(i  ,j+1)
        do i=2,n-1
        do j=2,n-1
                new(i,j) = -old(i,j) + 2*now(i,j) + velocity * laplacian(i,j)
        call srite('out',new,n*n*4)
        do i=1,n{  do j=1,n {  old(i,j) = now(i,j) }}
        do i=1,n{  do j=1,n {  now(i,j) = new(i,j) }}       # clock tick
        do i=1,n{  do j=1,n {  new(i,j) = 0.       }}
        }
return; end
```

Java Version

```
import browser.*;
import awt.*;
import java.lang.*;
import net.*;
class FrogPond extends Applet {

 /* This is the version running on the Frog Pond Benchmark page */
        float now [][]; /* arrays for calculating raster values */
        float old [][];
        float neww [][];
        float laplacian[][];
        float sum [][];
```

```
int numButtons=10;
int selectedButton;
int drawStatus=0;
int newheight=200;      /* display size */
int newwidth=200;
final int height=50;     /* calculation matrix size */
final int width=50;
float velocity;
float viscosity;
int pclip;              /*the clip percentage: any value in the
                          calculation matrix that is at or above this
                          percentile or at or below 100-this
                          percentile will be assigned the maximum
                          or minimum value, respectively*/
int status;
int click;
boolean rmode=true;     /* is random mode on? */
boolean painted=false;  /* is pond finished painting? */
boolean up=false;       /* has user just clicked on the  applet? */
boolean newFrog=false;  /* will the frog be moving this frame? */
boolean firstFrog=true;
boolean firstClick=true;
int curX;
int curY;
int nextX;
int nextY;
int frogX;
int frogY;
int fpd=1;              /* frames of calculation per pond update */
int fnum=0;
boolean clicked=false;
Image frogImg;
ImageMaker imgmkr;
RButton[] buttons=new RButton[numButtons];
long startTime;
long pauseTime;
long pauseStartTime;
long paintTime;
long paintStartTime;
BarGraph graph;
float fPerS=0.0;

public void init() {
        int i;
        int j;
        now=new float[width][height];
        neww=new float[width][height];
        old=new float[width][height];
        laplacian=new float[width][height];
        sum=new float[width][height];
        resize(550,480);
        fnum=0;
        pauseTime=0;
        rmode=true;
```

```
                velocity=.20;
                viscosity=.15;
                pclip=100;
                click=0;
                status=1;
                curX=-1;
                curY=-1;
                frogImg=getImage("images/frog.gif");
                for(i=0;i<width; i++) {
                for(j=0;j<height;j++) {
                                now[i][j]=0;
                                old[i][j]=0;
                                neww[i][j]=0;
                }
                }
                imgmkr=new ImageMaker(width,height,newwidth,newheight,neww,
                                      pclip);
                imgmkr.makeImage();
                makeButtons();
                makeGraph();
                nextX=(int)(width*Math.random());
                nextY=(int)(height*Math.random());
                startTime=System.currentTimeMillis();
        }

        public void paint (Graphics g) {
                g.drawImage(imgmkr.img, 0, 0);
                for(int i=0;i<numButtons;i++) buttons[i].drawUp(g);
                updateStats(g);
                graph.drawLabels(g);
                painted=true;
        }

/* The update method is used so the applet display will be updated
 * throughout the execution of the applet, not just at its termination
 */
                public void update (Graphics g) {
                switch(drawStatus) {
                        case 0: updatePond(g);painted=true;
                                newFrog=false; break;
                        case 1: painted=true;
                                buttons[selectedButton].drawDown(g);
                                painted=true; break;
                        case 2: buttons[selectedButton].drawUp(g);
                                painted=true; break;
                        case 3: updateStats(g);painted=true; break;
                        case 4: updateGraph(g);break;
                }
                painted=true;
        }
        public final void updatePond(Graphics g){
                paintStartTime=System.currentTimeMillis();
                imgmkr=new ImageMaker(width,height,newwidth,                                    newheight,neww,pclip);
                imgmkr.oldras=neww;  /* create the image */
```

```
        imgmkr.pclip=pclip;
        imgmkr.makeImage();
        if(newFrog){                          //find new frog location
                int fx=(int)(frogX*width/newwidth);
                int fy=(int)(frogY*height/newheight);
                if(neww[fx][fy]<=0)g.setForeground(Color.blue);
                if(neww[fx][fy]>0)g.setForeground(Color.white);

                g.fillRect(frogX,frogY,37,28);
                g.setForeground(Color.black);
                if(firstFrog){pclip=99; firstFrog=false;}
        }
        g.drawImage(imgmkr.img,0,0);
        frogX=(int)(nextX*newwidth /width -19);
        frogY=(int)(nextY*newheight/height-14);

        /* if frog overlaps edge of pond, reposition him so he
           doesn't     */

        if(frogX>newwidth -37)frogX=newwidth -37;
        if(frogY>newheight-28)frogY=newheight-28;
        if(frogX<0)frogX=0;
        if(frogY<0)frogY=0;
        if(curX!=-1)g.drawImage(frogImg,frogX,frogY);
        g.clearRect(405,270,150,115);
        int len=405+g.drawStringWidth("Frames/sec:  ",405,290);
        long msecs=System.currentTimeMillis()-startTime-pauseTime;
        fPerS=(float)(1000*fnum)/msecs;
        g.drawStringWidth(Float.toString(fPerS),      len,290);
        len=    405+g.drawStringWidth("Frame #: ",    405,315);
        g.drawStringWidth(Integer.toString(fnum),     len,315);
}
public final void updateStats(Graphics g){
        int totalLen;
        g.clearRect(0,400,550,40);
        totalLen=g.drawStringWidth("XSize: ",   0,         410);
        totalLen+=g.drawStringWidth(Integer.toString(newwidth),
                                  totalLen,410);
        totalLen+=g.drawStringWidth("   YSize: " ,
                                  totalLen,410);
        totalLen+=g.drawStringWidth(Integer.toString(newheight),
                                  totalLen,410);
        totalLen+=g.drawStringWidth("   Status: ",totalLen,410);
        switch(status){
                case 1:totalLen+= g.drawStringWidth("running",                                                                   totalLen,410);break;
                case 2: totalLen+=g.drawStringWidth("finished",
                                                  totalLen,410);break;
                case 3:totalLen+= g.drawStringWidth("paused",
                                                  totalLen,410);break;
        }
        totalLen+=g.drawStringWidth("   Random: "      ,totalLen, 410);
        if(rmode) totalLen+=g.drawStringWidth("on",     totalLen, 410);
                else totalLen+=g.drawStringWidth("off", totalLen, 410);
        totalLen+=g.drawStringWidth("   Frame Skip: ",  totalLen, 410);
```

```
        totalLen+=g.drawStringWidth(Integer.toString(fpd-1),
                                                totalLen, 410);
        totalLen+=g.drawStringWidth("      Clip %: ",   totalLen, 410);
        totalLen+=g.drawStringWidth(Float.toString(pclip),
                                                totalLen,410);
        painted=true;
}

public final void updateGraph(Graphics g){
        graph.draw(g,fPerS);
}
public final void makeButtons(){

        buttons[0]=new RButton(405,0  ,30,20,"pause"   ,Color.orange,
        null,Color.black,null,true);
        buttons[1]=new RButton(405,55 ,30,20,"stop"    ,Color.red,
        null,Color.black,null,true);
        buttons[2]=new RButton(405,110,30,20,"clip +1%",Color.cyan,
        null,Color.black,null,true);
        buttons[3]=new RButton(465,55 ,30,20,"random"  ,Color.red,
        null,Color.black,null,true);
        buttons[4]=new RButton(465,0  ,30,20,"reset"   ,Color.orange,
        null,Color.black,null,true);
        buttons[5]=new RButton(465,110,30,20,"clip -1%",Color.cyan,
        null,Color.black,null,true);
        buttons[6]=new RButton(405,165,30,20,"fskip+"  ,Color.green,
        null,Color.black,null,true);
        buttons[7]=new RButton(465,165,30,20,"fskip-"  ,Color.green,
        null,Color.black,null,true);
        buttons[8]=new RButton(405,210,30,20,"enlarge"  ,Color.magenta,
        null,Color.black,null,true);
        buttons[9]=new RButton(465,210,30,20,"reduce"  ,Color.magenta,
        null,Color.black,null,true);
}

public final void makeGraph(){
        String labels[]=new String[3];
        labels[0]="Macintosh";
        labels[1]="Sun Solaris";
        labels[2]="Cray";
        graph=new BarGraph(30,440,450,20,0.0,2.0,3,labels);
}

public void mouseUp(int x, int y){

        int temp=drawStatus;

        if(x<newheight && y<newwidth){  /* user has clicked on pond*/
                if(!clicked)firstClick=true;
                clicked=true; click=1; rmode=false; newFrog=true;
                curX=nextX;
                curY=nextY;
                nextX=(int)(x*width /newwidth);
                nextY=(int)(y*height/newheight);
```

```
        } else {     /* see if user has hit a button */
                for(int i=0;i<numButtons;i++){
                        if(buttons[i].selected(x,y)) selectedButton=i;
                        up=true;
                }
        }
}
public final void handleMouseUp() {

        up=false;
        drawStatus=1; /* draw button down */
        repaint();
        drawStatus=2; /* draw button up */
        repaint();
        doButtonEvent();
}

public final void doButtonEvent() {
        drawStatus=3;
        switch(selectedButton) {
                case 0: if(status==3){status=1;break;}status=3; break;
                case 1: status=2;              break;
                case 2: if(pclip<100)pclip+=1; break;
                case 3: rmode= !rmode;         break;
                case 4: init();                break;
                case 5: if(pclip>6)  pclip-=1; break;
                case 6: if(fpd<39)   fpd++;    break;
                case 7: if(fpd>1 )   fpd--;    break;
                case 8: if(newheight<390){ newheight+=10;
                                newwidth+=10;} break;
                case 9: if(newheight>10){ newheight-=10;
                                newwidth-=10;} break;
        }
        repaint();
}

public final void calcValues() {

       int i,j;

        for(i=0;i<width ;i++){
        for(j=0;j<height;j++){
                sum[i][j]=now[i][j]-viscosity * old[i][j];
        }
        }
        for(i=1;i<width-1 ;i++){
        for(j=1;j<height-1;j++){
                laplacian[i][j]=  sum[i-1][j]-2*sum[i][j  ] +
                                  sum[i+1][j]+  sum[i][j-1] -
                                2*sum[i  ][j]+  sum[i][j+1];
        }
        }
        for(i=1;i<width-1 ;i++){
        for(j=1;j<height-1;j++){
```

```
                neww[i][j]=-old[i][j]+2*now[i][j]+velocity*
                                  laplacian[i][j];
        }
        }
}


public final void resetArrays(){

        int i,j;

        for(i=0;i<width;i++){
        for(j=0;j<height;j++){
                old[i][j]=now[i][j];
        }
        }
        for(i=0;i<width;i++){
        for(j=0;j<height;j++){
                now[i][j]=neww[i][j];
        }
        }
        for(i=0;i<width;i++){
        for(j=0;j<height;j++){
                neww[i][j]=0;
        }
        }
}


public final void randomToss(){
        curX=nextX;
        curY=nextY;
        nextX=(int)(width*Math.random());
        nextY=(int)(height*Math.random());
        now[curX][curY]+=1; /* frog has jumped, start wave */
        newFrog=true;
}
public void start (){

        int p=0;

        while(status==1||status==3){
                if(status==3)while(status==3){
                        checkMouse();
                        if(p==0)pauseStartTime=
                                System.currentTimeMillis();
                        p=1;
                }
                if(p==1)pauseTime=(System.currentTimeMillis()
                                        -pauseStartTime);
                p=0;
                checkMouse();
                if(click==1){
                        now[curX][curY]+=1; // drop stone where
```

```
                click=0;           // user clicked
            }
            if((rmode)&&(fnum)%10==0)randomToss();
            calcValues();// calculate new values
            fnum++;
            if(firstClick){
                drawStatus=3;
                repaint();         //so display will say
                firstClick=false;  //random is off
            }
            if((fnum%10)==0){
                drawStatus=4;
                repaint();
            }
            drawStatus=0;
            if((fnum%fpd)==0){
                painted=false;
                repaint();
                while(!painted){} /*make sure its finished */
            }
            resetArrays();
        }
        stop(); /*so Hotjava doesn't crash*/
    }
    final public void checkMouse(){
        if(up)handleMouseUp();
    }
}
```

APPENDIX B

CREATING A SEPJAVA FILE

SEPJava files follow the general form of SEPLIB files. Each dataset contains an ASCII file (.J by convention) and a binary data file. The ASCII file follows the standard SEPLIB conventions (in, esize, n1 n2, d1, d2, o1, o2, etc.) The only additional requirement is this cube description is given in a specific form (seen below) at the beginning of the ASCII file. The .J file for the data seen in Figure 3 is seen below:

```
in=circle.T@
esize=1
n1=250
n2=90
n3=5
d1=0.000002000
d2=50.000000
d3=1.000000
o1=0.002000
o2=50.000000
o3=0.000000
title=Phase Shift Circle
label1=time
label2=position
label3=phase
```

In addition to the .J file and the binary data file, an html file is required. The html file requires the applet, in this case the Animator, you wish to call; the location of the code for that applet; the path of the data; and the location of the header file.

```
<html>
<head>
<title> circle.J </title>
</head>
<APP CLASS="Animator"
     src="http://sepwww.stanford.edu/sep/krl/Animator/"
     PATH="http://sepwww.stanford.edu/pub/sep/bob/"
     HEADER="http://sepwww.stanford.edu/pub/sep/bob/circle.J">

</html>
```

At SEP, a SEPLIB history file can be created by using `Bytecube2www`:

```
Byte2java <input.H pars

Required:
        applet=a   Animator
               c   Cube Viewer
Optional:
```

```
        in_J=Name of the binary header file
        htmlout=name of html file [input.html]
        fileout=Java file name
        anipath=http path of the applet
                [http://sepwww.stanford.edu/sep/krl/[applet]/]

        data_http=http address of the java file
        data_out=unix directory
        htmlpath=html directory
                all defaulted to (/sepwww/pub/sep/username/
                   and http://sepwww.stanford.edu/sep/username/
                username can be over written by using user variable
                  to put files in subdirectories


Purpose: Converts a standard SEPLIB header file to our accepted form for a
SEPJava header file and puts the header file, the data file, and a
created html file into the user's sepwww directory.
```

APPENDIX C

ANIMATOR USER'S GUIDE

The Animator's default (and maximum) display size is 400 X 400 pixels. To change the display size you can click on the green rectangle in the upper left corner of the display and drag the mouse arrow to the desired bottom left corner while holding the mouse button down. At which point you will be asked to confirm the resizing of the frames. If you answer yes, each new image will be created and displayed, much like the initial stages of the applet. If you answer no, no action will be taken.

To zoom on a portion of the display, click on the image and drag the mouse arrow to select the desired area. A window will appear to the right of the display area, giving coordinates of the selected area. Again, upon releasing the mouse button, you will be asked if you want to zoom all frames. If yes, each image will be created and displayed. If no, only the current frame will be zoomed. To un-zoom the frame, press the Un-zoom button.

Various display and movie parameters can be manipulated by selecting the buttons to the right of the movie display, either by clicking or by pressing the key displayed in blue on the button.

Button Descriptions, shortcut indicated in (**key**).

- RB(**b**) - Runs the movie backwards, displaying all marked frames.
- RF(**f**)- Runs the movie forward.
- Stop (**s**) - Puts applet back in edit mode, at frame #1. Only functional when the movie is running.
- Pause (**h**) - Pauses the movie. Only functional when the movie is running.
- FB (**p**) - In edit mode, displays the previous frame. If the movie is running, puts the applet back in edit mode.
- FF (**n**)- In edit mode, displays the next frame. If the movie is running, puts the applet back in edit mode.
- Mark (**m**)- Marks the frame to be displayed in the movie, or un-marks it if already marked.
- Un-zoom (**z**) - Creates the original image for the current frame.
- Un-mark All (**u**) - Un-marks all frames.
- Mark All (**a**) - Marks all frames.
- Un-zoom All (**U**) - Un-zooms all frames.

- Add Frame (**A**) - Creates a new frame using the data for the current frame and inserts it after the current frame. This is useful if you want to highlight different portions of one frame in a movie.
- Quit Applet (**q**) - Stops the applet- It is now safe to leave the page.
- Delay + (**+**) and Delay - (**-**) - Increases or decreases the delay between frame in 50 ms increments.

APPENDIX D

CUBE VIEWER USER'S GUIDE

Before using the `Cube Viewer` set the size of your browser window. The default size will work, but a larger window size is also supported. Once you have gone to an html sight that uses the `Cube Viewer` you are prompted to set the size of the viewer and then the relative size of the three views. The size of the viewer is set by manipulating the green box and the relative size of the three views is set by moving the red ball in the middle of the display. At anytime after loading you can reset these parameters, but it will result in a long delay while the images are recreated.

Once the size is set, the data is loaded in and the frames are created. At this point the `Cube Viewer` has two main operating modes:

- run mode which allows you to play a movie backwards or forwards along any of three principle directions with a single key command
- frame mode which allows actions such as advancing by a single frame.

The run mode has several button options:

- **f**ront- movie through the i3 direction of the cube
- **s**ide- movie through the i2 direction of the cube
- **t**op- movie through the i1 direction of the cube
- **1**- to run the selected movie forward
- **0**- to run the selected movie backwards
- **+**- increase the delay between frames by 50 ms
- **-**- decrease the delay between frames by 50 ms
- sto**p**-to stop the current movie
- **h**- to pause the movie

The frame mode can be selected at any time by either clicking on one of the views with the mouse or using the appropriate key command:

- **F**ront- set the front as the active frame
- **S**ide- set the side as the active frame

- **T**op- set the top as the active frame
- **m** - to go backward one frame along active frame direction
- **n** - to go forward one frame along active frame direction

The `Cube Viewer` also recognizes:

- **q**uit - to quit the applet
- **a** - to repaint the applet, necessary when you resize the window, or have temporarily covered the display.