```
/******************************************************************/
/*                        illiSkel                                */
/******************************************************************/
/* (C) 1996 George Francis, Chris Hartman, Glenn Chappell         */
/* TeX version, 7 September 1996, 12 January 1997                 */

#include <gl.h>        /* graphics library    */
#include <device.h>    /* mouse and keyboard  */
#include <math.h>      /* for the atof        */
#include <sys/time.h>  /* for the speedometer */
#include <stdio.h>     /* standard io stuff   */

#define  ABS(x)           (((x)<0)?-(x):(x))
#define  DOT(p,q)         ((p)[0]*(q)[0]+(p)[1]*(q)[1]+(p)[2]*(q)[2])
#define  NRM(p)           fsqrt(DOT(p,p))
#define  FOR(i,a,b)       for(i=a;i<b;i++)
#define  FLYMODE          modus == 0
#define  TURNMODE         modus == 1
#define  IF(K)          if(getbutton(K))
#define  SOAK(K)        while(getbutton(K))
#define  TOGGLE(K,f)    IF(K){f = !f;SOAK(K);}
#define  IFCLICK(i,K,a) {static flg=i; TOGGLE(K,flg); if(flg){a};}

/*************** illiSkel's  "private" variables ****************************/
Matrix id={{1.,0.,0.,0.},{0.,1.,0.,0.},{0.,0.,1.,0.},{0.,0.,0.,1.}},aff,starmat;
int ii,jj,kk, modus,binoc,win,msg;
float lux[3]={1.,2.,3.},lu[3],gnd,torq,nose,mysiz,focal,speed,far,star[1000][3];
char phrase[256];
long mx,my,xorig,yorig,xcen,ycen;

/*************** your global variables*****************************************/
int cube,thick;
long znear,zfar;

/***************** Parameter management functions ***************************/

void arguments(int argc,char **argv){ /* from Pat Hanrahan, 1989 */
     while(--argc){
         ++argv; if(argv[0][0]=='-')switch( argv[0][1]){
         case 'w': win    = atoi(argv[1]);    argv++;argc--;break;
         case 's': speed  = atof(argv[1]);    argv++;argc--;break;
         case 't': torq   = atof(argv[1]);    argv++;argc--;break;
         case 'L': lux[0] = atof(argv[1]);
                   lux[1] = atof(argv[2]);
                   lux[2] = atof(argv[3]); argv+=3;argc-=3;break;
       } /*end switch */
     }/*end while */
}/* to add/subtract commandline arguments insert/delete like code */
```

```
void deFault(){  /* reusable initialization in the key of Z */
   win   = 2;   msg   = 1;   torq  = 0.02; speed = 0.5; nose= 0.06;
   mysiz = 0.1; focal = 1.5; far   = 20;   modus = 1; binoc = 0; gnd = 0;
   {float tmp=NRM(lux); FOR(ii,0,3) lux[ii] /= tmp;};/* light direction */
   FOR(ii,0,4)FOR(jj,0,4)starmat[ii][jj] = aff[ii][jj] = id[ii][jj];
   aff[3][0]= 0.; aff[3][1]= 0.; aff[3][2]= -4.2; /* move it away */
   thick=4; cube=1;
} /* end deFault */

void keyboard(){ /* control keys */
#define  IFSHIFT        if(getbutton(LEFTSHIFTKEY)||getbutton(RIGHTSHIFTKEY))
#define  PRESS(K,A,b)   IF(K){IFSHIFT{A;}else{b;}}
#define  PRES_S(K,A,b)  IF(K){IFSHIFT{A;}else{b;};SOAK(K);}
#define  CYCLER(K,f,m)  PRES_S((K), (f)=(((f)+(m)-1)%(m)), (f)=(++(f)%(m)) )

  TOGGLE(PRINTSCREENKEY, msg);                          /* messages */
  CYCLER(VKEY,binoc,2);                                 /* cross-eyed */
  PRESS(IKEY, mysiz /= 1.1, mysiz *= 1.1)               /* rescale world */
  PRESS(OKEY, focal *= 1.1 , focal /= 1.1)              /* telephoto */
  PRESS(PKEY, far *= 1.01 , far   /= 1.01)              /* beware of this */
  PRESS(NKEY, nose -= .001 , nose += .001 );            /* for binoculars */
  TOGGLE(QKEY,cube)                                     /* stick cube     */
  TOGGLE(SPACEKEY,modus);                               /* FLY/TURN mode   */
  PRES_S(PADASTERKEY,gnd=0 , gnd=0xffffffff);           /* white background */
  PRESS(SKEY,speed /= 1.02, speed *= 1.02);             /* flying speed */
  PRESS(TKEY,torq /= 1.02, torq *= 1.02);               /* turning speed */
  PRESS(ZKEY, deFault(), deFault());                    /* zap changes   */
  if (getbutton(HOMEKEY)) while(!getbutton(ENDKEY));    /* po' man's freezer*/
}/* end keyboard */

float speedometer(){ /* returns average of last 8 frames per second/ */
     double dbl; static double rate; static int ii=0;
     static struct timezone notused; static struct timeval now, then;
     if( ++ii % 8 == 0){  /* 8 times around measure time */
          gettimeofday (&now, &notused); /* elapsed time */
          dbl =  (double)(now.tv_sec - then.tv_sec)
                +(double)(now.tv_usec - then.tv_usec)/1000000 ;
          then = now;  rate = 8/dbl; }
     return((float)rate);
}
```

```
void messages(long xt, long yt){ /* text information as heads-up display */
float horz, vert;
#define  LF         (vert-=0.035)  /*as in line-feed */
#define  LAB_L(s,u) sprintf(phrase,s,u); cmov2(horz,vert); charstr(phrase);

  reshapeviewport(); /*fit viewport to window */
  zclear(); /* so writing is always in front */
  ortho2(-(float)xt/yt,(float)xt/yt,-1.,1.);  /* new projector */
  cpack(TURNMODE?0xffdd8822:0xff00cccc); if(!binoc)circ(0.,0.,.01);
  horz= -1.2; vert = -.95; LAB_L("%0.1f ",speedometer());
/*so that speedometer and bullseye is visible also in small windows  */
  viewport(0,getgdesc(GD_XPMAX),0,getgdesc(GD_YPMAX));
  horz=-.6; vert=.90;
  LAB_L("| (ESC)ape rtica  | (Z)ap changes |(BAR)fly/turn | stereo(V)iew |",0);LF;
  LAB_L("| (PRINT) messages| (PAD*) ground |(LOCK)keyboard| (HOME)/(END) |",0);LF;
  horz=-1.2; vert=.90;   /* in world coordinates for the viewport */
  LAB_L ("(S)peed        %0.4f",speed); LF;
  LAB_L ("(T)orque       %0.4f",torq); LF;
  LAB_L ("(N)ose         %0.3f",nose); LF;
  LAB_L ("f(O)cal factor %g",focal); LF;
  LAB_L ("my s(I)ze      %g",mysiz); LF;
  LAB_L ("near clipper   %g", mysiz*focal); LF;
  LAB_L ("far clip(P)er  %g",far); LF;LF;
  horz=0.0; vert=-.95;
  LAB_L ("illiSkeleton: George Francis, Chris Hartman, U. Illinois, 1996",0);
}/* end messages */

/******************** Scene production functions *****************************/

initstars(){ /* Glenn Chappell 1991, CMH 1994 */
 FOR(ii,0,1000)FOR(jj,0,3)star[ii][jj] = random()/(float)0x40000000-1.;
 }

drawstars(){
  pushmatrix();
  multmatrix(starmat);
  cpack(0xffffffff);
  bgnpoint(); FOR(ii,0,1000)v3f(star[ii]); endpoint();
  popmatrix();
  zclear();
}

drawcube(){ /* Steve Kommrusch, 1984, uses Gray code */
    /* Self-initializing function draws a cube.*/
    static int first=1; static float vq[8][3];
    static int fr[]= {0,1,5,1,3,7,3,2,6,2,0,4,5,7,6,4};
    if(first){FOR(ii,0,8)FOR(jj,0,3)vq[ii][jj]=(ii&(1<<jj))?1.:-1.; first=0;}
    linewidth(thick);
    bgnline(); FOR(ii,0,16)v3f(vq[fr[ii]]); endline();
} /* end drawcube */
```

```
skeldraw(long xt, long yt){  /* drawing factored out of main */
   if(binoc) viewport(0,xt/2,yt/4,3*yt/4);
   window(-mysiz*xt/yt,mysiz*xt/yt,-mysiz,mysiz,mysiz*focal,far);
   drawstars(); /* no parallax on the stars */
   translate(-binoc*nose,0,0);
   multmatrix(aff); drawcube();
   if(binoc)
       {
       viewport(xt/2,xt,yt/4,3*yt/4);  /* left is right */
       window(-mysiz*xt/yt,mysiz*xt/yt,-mysiz,mysiz,mysiz*focal,far);
       drawstars();
       translate(binoc*nose,0,0);
       multmatrix(aff); drawcube();
       }
} /* end skeldraw */


/***************************Navigation functions***************************/

calculite(){ /* transpose = inverse for orthogonal matrices */
  FOR(ii,0,3)for(lu[ii]=jj=0;jj<3;jj++) lu[ii] += aff[ii][jj] * lux[jj];
}

chaptrack(int mx,int my){  /* Chappell's flyor */
   int dx = getvaluator(MOUSEX) - mx;
   int dy = getvaluator(MOUSEY) - my;
   dx = ABS(dx)>5?dx:0; dy = ABS(dy)>5?dy:0; /* dead zone */
   loadmatrix(id);
   rot(dx*torq,'y'); rot(-dy*torq, 'x');
   PRESS(RIGHTMOUSE, rot(-10,'z'), rot (-1,'z'));
   PRESS(LEFTMOUSE,  rot( 10,'z'), rot ( 1,'z')); /* rotation is now complete */
   if(FLYMODE){pushmatrix();multmatrix(starmat);getmatrix(starmat);popmatrix();}
   PRESS(MIDDLEMOUSE, translate(0.,0.,-speed), translate(0.,0., speed) );
   if(TURNMODE){ Matrix temp;
       getmatrix(temp);
       loadmatrix(id);
       translate(aff[3][0],aff[3][1],aff[3][2]);   /*conjugate to rot center*/
       multmatrix(temp);
       translate(-aff[3][0],-aff[3][1],-aff[3][2]);
   }/* end TURNMODE */
       multmatrix(aff); getmatrix(aff); calculite();
}/* end chaptrack*/
```

```
/**********************main main main **************************************/

void main(int argc, char **argv){ long xt,yt,xo,yo;
  deFault();
  arguments(argc,argv);
  initstars();

  switch(win){
     case 0: break; /* mouse window */
     case 1: noborder(); prefposition(0,640,0,480); break;
     case 2: prefposition(0,getgdesc(GD_XPMAX),0,getgdesc(GD_YPMAX)); break;
   }
  foreground(); winopen("name me");
  zbuffer(1); doublebuffer(); RGBmode(); gconfig();

while(!getbutton(ESCKEY)){  /* eternal loop */
   IFCLICK(1,KKEY,keyboard(););
   reshapeviewport();
   czclear(gnd,getgdesc(GD_ZMAX));
   getsize(&xt,&yt);getorigin(&xo,&yo);
   IFCLICK(1,SCROLLLOCKKEY, chaptrack(xo+xt/2,yo+yt/2); )
   skeldraw(xt,yt);
   if(msg) messages(xt,yt);
   swapbuffers();
   } /* end while loop */
} /* end it all */
```

illiSkeleton Short Summary

George Francis

September 24, 2009

The real-time interactive computer animation **illiSkeleton.c**[1] is an abstraction of the classic 1994 **illiShell.c**. It maintains the structure, but is restricted to the console and, initially, omits lighting. It is in C/gl and illustrates a number of useful features, some in more than one way, to encourage experimentation. Here is the structure of this program.

Parameter management functions	
`arguments()`	//takes command line input
`deFault()`	//re-usable initialization
`keyboard()`	//non-queued keypress input
`speedometer()`	//reports average frame rate
`messages()`	//optional heads-up display
Scenery production functions:	
`initstars(), drawstars()`	// factored star drawers
`drawcube()`	// one-piece cube drawer with first flag
`skeldraw()`	// all graphics is done here
Navigation functions:	
`chaptrack()`	//mouse operated navigator
`main()`	// setup+loop

Dependency chart:
All functions are factors of `main()` with the following exceptions. The `speedometer()` is a factor of `messages()` . The `deFault()` is also called by certain resetting keypresses in `keyboard()` . The `arguments()` and `initstart()` functions are called only in the set-up part of `main()` . Both `drawstars()` and `drawcube()` are factors of `skeldraw()`. The factor `calculite()` of `chaptrack()`, and the normalization of the light direction in `deFault()` are not used in this initial version of the **illiSkeleton.c**, but are placed in the appropriate locations for future use.

[1]Long names being a nuisance, this family of RTICAare also called **isk.c** and **skel.c** on occasion.

In this summary description, it is assumed that the student is moving directly from **oc1.c** to **iSkel.c**, without necessarily working throught the intermediate lessons and tutorials. Thus comparisons are with **oc1.c** directly.

Additional `#include` header files are needed here. The `speedometer()` requires the `sys/time.h` header file for the `gettimeofday()` function. `math.h` is used for the `atof()` function in `arguments()`. The `stdio.h` is always useful and a nuisance to forget.

A number of `#define` (read "pound-define") macros are used here. These are expanded into the corresponding code by the compiler *before* compilation. Thus errors are not caught until the compiler reads the expanded code. Do not expect to see the macros in the error message. It is customary to use capital letters for pound-defines. For example, the macro `TOGGLE(K,f)` changes the truth-value of the bit-flag "f" every time the "K"-key is pressed *and* released.[2] The purpose of the `IFCLICK(i,K,a)` gadget is to enable you to make a small detour anywhere in your code without any rewriting at another location.[3] You can switch back and forth without exiting the running RTICA.[4]

A typical factor of `main(int argc, char **argv)` is a function that manages the command line `arguments()`. It allows the user to assign flags and parameters from the command line before the RTICAis started. This `arguments()` function was designed by Pat Hanrahan so that it is easy for the programmer to add and subtract socalled *command line switches*, namely items marked with a single "dash-letter" followed by none or more integers or floats.

Entering a command line that looks like this,

```
iris %    skel.x -w 1 -L 1.  2.  3.
```

sets the global flag `win` to 1. This specifies a particular size and location of the drawing window in `skeldraw()` . The function `atoi()` converts alphameric strings to integers, while `atof()` does the same thing for floats. The `-L`

[2]We call these objects *illiGadgets* since they are not, strictly speaking, "widgets", though they serve a similar purpose.

[3]For example, in `main()` we have isolated the keyboard with an `IFCLICK` so that we can type in one window while looking at the RTICAwithout messing it up. This is much nicer than having to keep pushing the mouse into windows to re-establish the "focus."

[4]Some C++ evangelists say that one should not use such things because they may lead to mistakes. Too many `IFCLICK`s scattered throughout your code will make the operation of your RTICAamusing, if not aggravating. Use it for hacking, debugging and experimenting. With some care, however, these tricks make C-code much easier to read, write and experiment with.

switch specifies a direction for the light, which requires three floats to follow the "dash-ell"[5].

All parameters should be assigned in the `deFault()` function.[6] It is also possible to assign "constant" parameters at the time of their definition. This was done for the identity matrix `id` , and the light direction vector, `lux` . It is very common for programmers to neglect to update the `deFault()` code when hacking an RTICA, or to neglect to document the reasons for placing something among the default assignments and calculations.

The `keyboard()` function converts the user's wishes into the state of the RTICA. It is heavily aliased with compiler `#defines` for readability and programming. Note that the scope of a compiler `#define` is from its first occurrence to the end of the program. Thus it is harmless to delay the definition of these aliases until they become useful in the `keyboard()` function. It might be safer but less elegant to define them at the beginning of the RTICA.

Three *illiGadgets* are defined here, and have been factored for easier reading. Remember that the compiler simply substitutes, so that is how you can unpuzzle these aliases. For example, the soaked toggle negates an integer flag, but does not take effect until you release the key. This avoids an odd number of toggles which would leave the RTICAin its original state. The `PRESS(K,A,b)` will execute the code abbreviated by `b` each time `keyboard()` is called, as long as you hold down the key named by `K`. If you also hold down either Shift-key, then code phrase `A` is executed. We might compare this gadget to a slider. If you do not want the pseudo-continuous action, then soak the gadget as in `PRE_S`. The `CYCLER(K,f,m)` advances the integer flag `f` modulo `m` each time `K`-key is pressed and released. Thus `TOGGLE(K,f)` could also be aliased to `CYCLER(K,f,2)`.

Exercise 1. Alternative Slider Gadget.
Design a slider with two different stepsizes, a fine and a coarse one, using the `ALTKEY` in addition to the `SHIFTKEY`s. Design a slider with a variable upper and lower limit of the range using some other control key. Be sure to display your actions on the message board.

The `IFCLICK(f,K,a)` macro has a very special function and should be used only temporarily and for experimental purposes. You can put a "siding" into the code stream which is toggled with the key named by `K`. It uses a static flag whose initial value is `f`, and whose scope is limited by a set of braces. If the `flg` is false, then `a` is not executed, if it is true, then it is executed.

[5]This version of `arguments()` does not tolerate errors and its abuse will dump core easily. A better protected but less versatile version will be introduced later.

[6]The capital F distinguishes this function from one with a similar name in some Iris header files.

Whenever the program flow passes by this point while the `K`-key is depressed, `flg` is toggled to its opposite value.

The function of the various keys is discussed at the place where their parameters and flags are implemented. The last two merit some discussion. The `ZKEY` here simply calls the `deFault()` function. There is room here for a second set of defaults on the shifted-`ZKEY`. However, it is easily converted to a cycler that permits an even larger choice alternative initial conditions.[7]

The `messages()` function takes the dimensions of the current window, `xt, yt` , as arguments and does the following. It holds the current position, `horz` and `vert`, in world coordinates that assign (0,0) to the center of the window, and $+1, -1$ to its top and bottom, as specified by the `ortho2()` command further on. The macro `LF` is, in effect, a line-feed. The macro `LAB_L(s,u)` writes the string `s` in which the value of `u` is embedded. It does this with the `sprintf()` function, whose syntax can be inferred from its usage here, or from its `man` pages. We are using 2-dimensional graphics so that the writing does not rotate with the mouse. Thus we must use `cmov2()` instead of the 3-dimensional `cmov()` function .

We clear the z-buffer so that our writing is visible no matter what the scene is in the window. The `ortho2(xmin,xmax,ymin,ymax)` establishes world coordinates that maintain the aspect ratio of the window, but not its parameters. If we are not in binocular mode, we place a bullseye in the center whose color echoes whether we are in `FLYMODE` or `TURNMODE`.

Since there is no good way of laying out text into a window whose size and position is voluntary, the remainder is usually skipped except for the wide open window `if(win==2)`.[8] Otherwise, we place messages into the window according to any one of several conventions. It is useful to have the most commonly used keys in **skel.c** to continue to have the same function in the descendants of **illiShell.c**. The resizable window mode, `if(win==0)`, is often used in while debugging and testing, so the frame rate as given by the speedometer is always printed.

The RTICA**skel.c** has two objects, the stars and the stick cube, with distinct affine matrices, call them M and A, . The background stars are created by `initstars()` and rendered by `drawstars()`.[9]

[7]We encourage this and similar programming devices to reduce the waste of time in recompiling or the waste of space in duplicating nearly identical versions of the same program.

[8]What is done here instead is a good illustration on the difference between the graphics window, the viewport and the projection matrix. When a window is opened maximally, the messages will also appear.

[9]You may wish to personalize your star chart so you can recognize an RTICAthat de-

The `drawstars()` function can be explained as follows. Duplicate the current matrix on top of the geometry pipeline stack. The top copy is composed with star matrix, M=`starmat`, and a thousand white stars are drawn, X=`star[ii]` . Since `drawstars()` is called in `skeldraw()` immediately after the off-axis perspective projection matrix, P = `window()`, is placed on the stack, the stars are, in fact, drawn thus: PMX. However, M is updated for each frame by `chaptrack()` when it is in the `FLYMODE`[10] Therefore, it is more correct to say that at time n we draw the stars thus: $PM_1M_2...M_nX$.

The cube is created and drawn in a somewhat different way. A collection of static quantities are used. The flag, `first`, is set, and shunts the `drawcube()` function through the initialization the first time it is called. The vertices of a 3D cube are loaded into a static float array, `vq[8][3]` . The bit pattern in the binary numeral, `ii` , for the current vertex, `vq[ii]`, determines the coordinate `vq[ii][jj]` to be $+1$ or -1, according to whether the jj-th bit (starting at 0) if `ii` is 1 or 0. The left-shift `1<<jj` moves the single bit in 1 left `jj` places. The bitwise and-operator, `&`, determines the parity of the jj-th bit in `ii`. The static integer array `fr[]` describes a polygonal path over the edges of the cube. Since the 3D cube does not have an Eulerian path, some edges are drawn twice. This is done in a way that hints how an Eulerian path may be decribed over a 4D cube.

Exercise 2. Hypercube
Draw a 4D cube, the *tesseract,* this way. Also, devise a control mechanism using key presses, that rotates the tesseract in 4-space before projecting it to 3-space. Use several different projection techniques for visualizing the tesseract. Cf the classic geometrical movie, "The Hypercube", by Tom Banchoff.

Exercise 3. illiTorus
Create the modification **illiTorus** of **iSkel** by inserting the torus from **tr1.c** thus. Rename `drawit() to drawtor()` in **tr1.c** and insert the functions `paint(), hair(), drawtor()`, near `drawcube()`. Of course, you will have to expand the pound-includes, pound-defines, and list of global variables to make `drawtor()` work. Then, insert it into `skeldraw()` instead of, or in addition to `drawcube()`. You might put those under an `IFCLICK()` control for added variety, and you might put four `PRESS` gadgets in which move the range-endpoints of the two angle parameters of the torus. This would yields a visible homotopy from a rectangular patch to a torus.

Since we are constructing graphics programs, the management of the geometry pipeline for each frame is truly the main activity. Generally, the corresponding code is written into `main()` . It has been factored out here for clarity. The `skeldraw()` function works like this. If binocular images are to be drawn, then the viewport is set to be the left quarter of the current window, but centered vertically. Subsequent drawing is done into this rectangle, and

scended from your own.

[10]In `TURNMODE` the sky is stationary.

clipped at its edges. First, an general (off-axis) projection matrix P, is placed on the stack with the function: `window(xmin, xmax, ymin, ymax, near, far)`.[11] A useful way to relate the viewport to the window is to imagine looking at the scene with the camera at the origin, and looking through a rectangular window with corners as specified, and a distance `near` from the camera, all in world coordinates. However, the image in the window is magnified or shrunk to fit into the viewport. Only the portion of the world located in the frustum of a rectangular cone between `near` and `far` is visible. Thus the `near` parameter serves two purposes: it establishes the perspective proportion and the near clipping plane. We decouple these functions by extracting a common factor, called `mysiz`. Note that changing `mysiz` with the `IKEY` slides the window back and forth in the rectangular viewing cone. Thus the scene does not change size, shape or location in the viewport. The front clipping plane can thus be moved back and forth at will. To change the angle of the viewing cone, it is necessary to change the effective focal distance with the `OKEY`. The two keys are so calibrated that pressing both at the same time, with or without the `SHIFT-KEY,` has the desirable effect of keeping the near clipping plane in the same place.

The stars are drawn next. Being infinitely far away, they are seen the same way by both eyes. But, to see the finite scene in binocular vision, the object needs to be shifted right or left half the distance between the eyes, as given by nose parameter on the `NKEY`. Since the left viewport is to be seen by the right eye in the crossed-eye[12] viewing mode, we shift the first image to the left, E^R. Thus PE^RAX is the order of matrix multiplication on a vertex X. In the monocular viewing mode, $E^R = I$, is the identity and we are wasting a matrix multiplication, but skip the left eye projection to the right viewport, PE^LAX. It is useful to associate the matrix products $P^R = PE^R$ and $P^L = PE^L$. The multiple viewport may be used to other purposes than for binocular pairs. For example, the front and rear view of the same object. Or, using many more viewports, a succession of small multiples in a homotopy.

Exercise 4. Small Multiples

Modify `skeldraw()` to show the front and rear view of the same object. Use four viewports, placing orthographic plan and elevation in three of them, with a perspective in the fourth. Use four viewports to show the four orthographics projections of a 4D object into 3D.

[11]It is unfortunate that this aptly named function had to be renamed in the **OpenGL** vocabulary. The contemporary identification of "window" with a resizeable viewport is too strong to allow for a double meaning. Thus it acquired the hardly euphonic name "frustum", after a section of any solid between parallel planes, often misspelled "frustrum."

[12]Some people call it the "cross-eyed" mode, since it is (erroneously) thought to induce this malady.

Devise a system of many viewports that display a homotopy some time steps apart in a system of temporal small multiples in the sense of Tufte.[13]

The principal navigation is effected in `chaptrack()` whose arguments are the coordinates of the center of the window, `mx, my` where the bullseye is drawn. The displacement, `dx, dy` , of the mouse from the center is clamped to zero if the mouse is within 5 pixels. This deadzone makes the mouse less responsive. The displacement is interpreted as a small rotation about the x-axis, which points East, and the y-axis, which points North. The left and right mouse buttons are used to effect a large or small rotation about the z-axis, which points into the screen. The fraction, `torq`, adjusts the responsiveness of the mouse. Since rotations are non-destructive, one starts by loading the identity matrix onto the stack. In the `FLYMODE` the incremental rotation, U, on the stack is relative to the camera, which assumed to be at he origin of the world coordinate system.[14] Therefore, the stars must be rotated. A duplicate of U is left on the stack by the `pushmatrix()` . It updates the star matrix M by multiplication. Thus, if the mouse is parked slightly off the bullseye, a succession of small rotations are applied to M.

Pressing the middle mouse button effects a translation in the Z-direction by a vector increment, dZ. The fraction `speed` adjusts the apparent speed of forward motion. The shifted middle mouse flies backwards. We may regard the composition $dA = dU\,dZ$ as an incremental change of the affine matrix in the Euclidean group. After n frames, the affine matrix, $A = I\,dA_n\,dA_{n-1}\,...\,dA_2\,dA_1$, is just the integral of all the little differential affine motions `chaptrack()` has implemented.

It should be noted here, that just like a Euclidean translation matrix, $T(m)$ depends on a single vector m of motion, a Euclidean rotation matrix, $U(a)$ depends on a vector a whose direction $\hat{a}$ and magnitude α is the axis and angle of rotation. We shall consider the arithmetic of the group of rotations later, and suppress the rotation vector in the notation. For now it is useful to know that the affine matrix of a Euclidean motion may be factored, $A = T(m)U$, with composition, commutation and inversion rules given by:

$$T(m_1)U_1T(m_2)U_2 = T(m_1 + U_1m_2)U_1U_2$$

$$T(m)U = UT(U^Tm)$$

$$(T(m)U)^{-1} = T(-U^Tm)U^T.$$

[13]Edward Tufte, *The Visual Display of Quantitative Information,* Graphics Press, 1983.

[14]The letter R is unsuitable for the name of a rotation matrix. We use U instead.

The superscript T emphasizes the fact that the inverse of a rotation matrix is just its transpose.[15]

After n-frames, the affine matrix, $A = I\ dA_n \ldots dA_2\ dA_1$, is just the integral of all the little differential affine motions calculated by `chaptrack()`.

However, in the `TURNMODE` the object is to be rotated about its own center. In that case, we use the translation vector,m, of $A = T(m)\,U$ to translate a point X back to the origin, apply dA, and translate back.[16] Thus once through `chaptrack()` in `TURNMODE` with $dA = dTdU$, has the effect of replacing A by

$$A_1 = T\ dA\ T^{-1}\ A = T\ dT\ dU\ T^{-1}\ T\ U = dT\ T\ dU\ U.$$

Consequently, after n iterations, the net change is,

$$A_n = dT_n\ dT_{n-1} \ldots dT_1\ T\ dU_n\ dU_{n-1} \ldots dU_1\ U.$$

and the *illiRotor* has smoothly accumulated the translation and rotation intended by the mouse displacements.

Exercise 5. Six-way Navigator
Build a navigator which has all six translations. For example, let a click of the middle mouse button represent a switch into translation mode, with the mouse deviation from the center representing translation. Note that this way it is not possible to fly because the mouse manages two modes consecutively instead of simultaneously. To recover the simultaneity, use buttons to manage translations.

All that remains is to assemble all of these components into `main()`. In the *overture* of `main()` we set up the state of the RTICA. These routines are done only once, and the order in which they occur does matter somewhat. We read the `arguments()` from the command line after the `deFault()` in case the user wishes some changes. Since `deFault()` is called by the `(Z)`ap-key, and at present, `arguments()` does not change the default values, zapping reverts to the hard coded defaults. Note that a useful modification would

[15]Care must be taken to distinguish between Euclidean transformations in 3-space, and their representation as 4-dimensional matrices as they apply to the so-called homogeneous coordinates of points in 3-space.

[16]In the column vector mode, the affine matrix $T(m)U$ is represented by

$$\texttt{aff[col][row]} = \begin{bmatrix} U_{00} & U_{10} & U_{20} & m_0 \\ U_{01} & U_{11} & U_{21} & m_1 \\ U_{02} & U_{12} & U_{22} & m_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In the traditional computer graphics mode, the row vector for a vertex is multiplied into the transpose of this matrix. However, the code does not have to be changed, because `aff[][]` is the same array of arrays in either case.

be for (Z)ap to become a cycler which moves through various default sets, including the one from the command line (exercise!)

Since the star-object was factored into `initstars()` and `drawstars()`, the former is called here, the latter in the eternal loop. Recall that the other object, `drawcube()`, initializes 'itself' the first time it is called.

The switch on the window flag defaults to a voluntary window for `win=0`, to a borderless window in the correct position for shipping to an NTSC[17] projector for `win=1`, and a full screen for`win=2`. The `foreground()` reverses the default action by the Iris, which runs graphics processes in the background. Whether or not you wish to run in foreground is a matter for the application. Experiment! Similarly, you might see what happens if you don't use `zbuffer()`, `doublebuffer()`, or `RGBmode()`. However, except for the second one, nothing visible will occur with just the wire-frame cube. You should put a painted surface, for example the octahedron, into **skel.** If you `IFCLICK` some of these options inside the eternal loop, don't forget that sometimes `gconfig()` needs to be called to implement the changes.

The second part is the *eternal loop*, which can be (ESC)aped. Sometimes, it is good to use a shifted (ESC)ape key for quitting, so that one can edit in another window. Originally, the same sequence of functions are called each time through the loop. However, for certain purposes, such as editing text in another window, is may be useful to suppress the `keyboard()`, or to freeze the mouse activated `chaptrack()`. Here IFCLICKs are used for these options. Note that inserting and deleting such "interrupts" is editorially easy. Only one place in the code needs changing.

The `reshapeviewport()` matches the current viewport to the window, which might have been resized by hand since the last frame was constructed. The `czclear()` function is a compound that clears both frame- and z-buffers. The dimensions `xt, yt` and the coordinates of the lower left hand corner of the current window are read by `getsize( &xt, &yt)`, and `getorigin(&xo,&yo)` respectively. This way, the computed center of the viewport can be fed to `chaptrack()`.[18] Since `chaptrack()` sets up all of the geometrical transformations, we can call `skeldraw()` next, followed by the `messages()`. You might put an `IFCLICK` around the `swapbuffers()` to see what happens if you don't.

Exercise 6. illiOctahedron
Put the octahedron from **oc1.c**l into **skel.c** to obtain **illiOctahedron**.

[17]To fit into a standard NTSC video screen we use 540 x 480 pixels

[18]This function is named after Glenn Chappell, who whose 1990 Math 428 project, *illiFlyer*, greatly influenced our navigation algorithms ever since.