illiSkel in C/OpenGL/GLUT

George Francis

3 January 2002

1 History of the program.

The real-time interactive computer animation (RTICA) **illiSkeleton.c** is a practical derivative of the classic 1994 **illiShell.c**. It maintains the general structure of its parent but is restricted to animating the monitor of the console. The **illiShell.c** is a fully functional prototype application for the CAVE which also compiles and runs on the console without proprietary CAVE libraries.

Originally written in C and IrisGL, the illiShell and its evolutes have gone through many linguistic permutations, including C++ and OpenGL, with and without the GLUT extensions. Its C/OpenGL/GLUT derivative **skel.c**has been the basis of almost all RTICAs written since 1997 by students in illiMath courses and workshops. As such, the illiSkel lends some kind of minimal structural unity to illiMath, without resorting to proprietary libraries and packages which impede portability and hamper creativity. The 2002 version of illiSkel is introduced here. A subsequent chapter treats the extension to a CAVE and a CUBE application.

2 Programming with illiSkel.

To minimize the time it takes to master the illiSkel, and maximize the freedom to modify, extend or rewrite it, the program consists of a single source file of fewer than 365 lines of code. We assume only the most rudimentary programming skills, and have kept things brief and simple. The `main()` function appears at the end of the program, in keeping with the (now unpopular) philosophy of not working with a concept before knowing exaclty what it means. In particular, factors of a function precede it in the linear order of the code. Like listing the ingredients of a recipe before the instructions on how to cook the soup, you have to be able to read the program backwards as well as forward. Thus keystroke editors with nimble searching (vi or emacs) are better for studying **skel.c**than menu-driven scrolling editors. The reader displeased with this design is encouraged to rewrite the skel in a more conventional style, but with an agreement that every feature will work at least as well in the improved version.

To work with the illiSkel means to subtract and add functions, factor functions that are getting too verbose, and divide the functions into affinity groups

by what they do and how they relate to each other. Such "arithmetic" also leads to a better understanding of the illiMath system of writing real-time interactive computer animation.

By subtracting functions you produce a derivative RTICA which is more easily understood and built up again into a different one. You add functions to produce an extension of the original, integrating new features, better graphics, subtler navigation, etc. When a concept has grown too big to keep in mind all at once, we factor it into smaller concepts. Similarly, a function that has grown so big that it no longer fits on a page or a screen should be factored into more manageable bites. But also when you plan to make many experimental modifications to the same piece of a longer function, it wise to factor that section out. Finally, functions fit into groups by how they share variables and purposes.

3 Function dependency tree.

To get an idea of how how the RTICA works when it is running (rather than compiling) we use an easily edited function dependency tree. For the illiSkel this might look like this.

```
main()
  arguments()
  deFault()
  glut-initializions
  glut-mainloop
  with callbacks:
    drawcons()
      open-gl functions
      glFrustum()
      drawstars()
        random()
      drawall()
        drawtor()
          drawvert()
        drawcube()
      messages()
        char2wall()
        speedometer()
      glutSwapBuffers()
    keyboard()
      deFault()
        autotymer()
      bump()
        hasnumber()
        getnumber()
    specialkeybo()
    mousepushed()
    mousemoved()
    reshaped()
    idle()
      autotymer()
      glutPostRedisplay()
      chaptrack()
```

Here is how to read this dependency tree. The `main()` function executes `arguments(), deFault()` and registers the names of the callback functions `drawcons(), keyboard(), ... , idle()`. Then the RTICA enters an endless main-loop. The display function here is called `drawcons()` and its alternative in the illiShell, `drawcave()`, is a callback function for the CAVE library. On a single processor computer the display function is executed (almost) as frequently as the speed of the graphics card allows. This loop is interrupted when

a key is pressed on the keyboard, a button pushed on the mouse, the mouse is moved, or the window is reshaped, and the corresponding callback function is executed. The global parameters and flags changed by the suite of input callbacks are processed by the `idle()` function, which may be thought to be called at least once per display cycle.[1]

We distinguish three groups of functions in the illiSkel:

Parameter management functions	`steering`
`arguments()`	//takes command line input
`deFault()`	//re-usable initialization
`keyboard()`	//non-queued keypress input
`messages()`	//optional heads-up display
`speedometer()`	//reports average frame rate
`autotymer()`	//a simple clockwork animator
`idle()`	//done between graphics frames
Scenery production functions:	`scenery`
`initstars(), drawstars()`	// factored star drawer
`drawall()`	// draw everything else like
`drawtor()`	// a painted torus
`drawcube()`	// a hypercube
`drawcons()`	// graphics on the console
Navigation functions:	`navigation`
`chaptrack()`	//mouse operated navigator

This document normally fits into a sequence of lessons, starting with the illiOctahedron, illiTorus, illiQub, and the exercises associated with them. However, it is self-contained and can be profitably mastered with, perhaps, one or two micro-derivatives (ca 100 lines of code) such as **oc1.c, qub.c**.

4 Defines, includes and global variables.

In the interest of compactness, items usually found in a private header file appears at the top of **skel.c**. After including the public header files associated with the standard libraries, we list a sequence of useful if debatable compiler directives, also known a *pound-defines*. They are a species of *macros*. The precompiler rewrites the code you hand it[2] by substituting one string of symbols for another. Thus, we shorten Brahma-Gupta's constant (355/113) approximating π to `M_PI`.[3] The parenthesis are advisable, because the precompiler just substitutes strings, not values. Many a bug develops by ignoring this fact of life.[4]

The precompiler is, however, more obliging than that: it also substitutes functional argument names. Thus the string `MAX(foo, 5)` in your code is

[1] On a multiprocessor computer, or in a distributed graphics cluster, the idle function may be executed asynchronously with the graphics display functions. Thus, graphics calls should should be made only by the display function, and data should be updated in the idle function and its factors. That the navigation function `chaptrack` appears to violate this rule is discussed below.

[2] Originally, before C++ specific compilers were created, the C++ precompiler rewrote the code into C before submitting to a standard C-compiler.

[3] In Unix this actually redefines a more accurately defined constant by the same name. The windows compiler is happy to be told a value of *pi*.

[4] The less permissive syntax of C++ methodically uses *inline functions* instead of pound-defined macros.

rewritten to `(((foo)<(5)?(5):(foo))`. The extra parentheses prevent possible misunderstandings in more complicated contexts.[5]

Macros make subsequent code shorter and more readable. It is cutomary to use all caps for their names to make them easier to spot in the source code. Do not expect to find them in compiler error messages, because the compiler sees only only their expansions. Though replacing macros with function calls is an alternative, it makes the program run more slowly and leads to verbose, tortured code. Take the macro `IFCLICK`, which uses the function `getbutton()` to set a switch bypassing a segement of code. The argument of `getbutton()` is the name of a key, and the function returns 1 if the key had been pressed.[6] Inserting an IFCLICK macro anywhere in a function creates a block with a static flag whose value persists from one call to the next. A keypress toggles the flag. If the flag is down, the code segment is bypassed. In the `idle()` function, for example, pressing the (=)-key will freeze navigation because the function `chaptrack()`, which updates navigation, is *not* called again until you press (=). Try the (–)-key and see what the OpenGL depth test does for a living.[7]

`getbutton()`

Exercise. Since all interrupts from the keyboard or the mouse are mediated by the GLUT mainloop, you cannot bypass a callback function as easily. But you can prevent the effect of a function by setting a flag right at its beginning which makes it return empty-handed, as it were. For instance, write `IFCLICK('+', return;)` as the first line of `mousemoved()` and mouse motion will be ignored when you toggle the (+)-key. This won't work right for the keyboard callback. Why? Sometimes you'll really want to write text into one window while watching the RTICA in another, and wish that the RTICA were deaf. Modify the code of `keyboard()` which solves this problem.

Exercise. Some macros are temporary constants, which you wish to change between recompilations. For instance, the default **skel.c**has 32 meridians and latitudes on its torus. After you have understood how illiGadget work, write one which makes it possible to change how many strings make up the mesh of the torus while it is running.

5 Parameter management functions.

We next discuss the those functions which are involved with the management of control parameters. They are concerned with *how* something is rendered

[5] And, for the reader meeting the expression `B?T:F` for the first time, know that the compiler evaluates it to the expression `F` if the value of `B` is 0, and otherwise to `T`.

[6] In IrisGL, `getbutton()` was a system function which really interrupted the program flow and told the truth. In OpenGL, such interrupts are not allowed for reasons of inter-system portability. So we must simulate this function as explained further on.

[7] Using IFCLICK transgresses the very tidy habit of processing all keypresses in the same (two) input funtions `keyboard(), specialkeyo()`. Moreover, some C++ evangelists say that one should not use such tricks at all. Too many `IFCLICKs` scattered throughout your code will make the operation of your RTICA amusing, if not aggravating. Use it sparingly during development or your code and for analysing someone elses code. Use it for hacking, debugging and experimenting, especially when it is not clear yet what features should become permanent options.

by the RTICA , not with *what* is rendered. The clockwork animation function `autotymer()` appears first in the source code only for convenience of editing. The first factor of of `main()` is `arguments`, which picks up the input you wrote on the command line. It is not possible to settle on a best linear ordering of the functions in the source code, so we keep the historical order for easier comparison with RTICAs based on older versions. Use your editor to find the functions in the code, and to trace its variables forward and backward.

A typical factor of `main(int argc, char **argv)` is a function that manages the command line `arguments()`. The system reads the entire command line and provides `main()` with a counter, `argc`, and a pointer of pointers, `argv`. Initially, `argc` equals the number of words on your command line, `argv[0][0]` is the the first character in the first word. The `arguments()` function we use in **skel.c**was designed by Pat Hanrahan to make it easy for you to add and subtract *command line switches*. These are items marked with a single "dash-letter" followed by one or more integers or floats.[8] arguments()

Entering a command line that looks like this,

iris % `skel.x -w 0 -L 0. 0. 10. -g .8`

sets the global flag `win` to 0. This specifies a particular size and location of the drawing window, as specified by the calls to the GLUT library initialization in `main()`. The function `atoi()` converts alphameric strings to integers, while `atof()` does the same thing for floats. Thus, the final `-g .8` sets the default `gap0` size to .8.[9]

The light direction vector also comes twice, `lux[], luxx[]`, but for different reasons. The default direction is (1,2,3), but it can be changed on the command line with the `-L` option. We have capitalized the flag to remind you that is a vector, not a number. In order for the bright spot (signifying the light source) to move across a rotating object correctly, our lighting method, *illiLight* rotates the light direction appropriately with `luxx[]`.

All parameters should be assigned their default values in the `deFault()` function.[10] It is also possible to assign constant parameters at the time of their definition. This was done for the the light direction vector, `lux[]`. Note that we make sure that `lux[]` is really a unit vector. In `deFault()` we also reset the two navigation matrices, `aff[], starmat[]`, to the identity.[11] The clockwork `autotymer()` is also reset by `deFault()`. As you hack an RTICA you should resist the temptation of neglecting to update `deFault()`.

---

[8]Hanrahan's ingenious code is not for the fainthearted. It also does not suffere fools gladly. It expects to be used by people who know what they're doing. Therefore, it had been superceded in illiMath RTICAs by the much more robust but different `getopt()` command line argument handlers. Hanrahan's code was revived when we found just how awkward it was to make the "getoptery" work in Windows. You are welcome to replace replace Hanrahan's arguments with something more foolproof as long as the functionality remains.

[9]The reason we need two gap variable is this. The `deFault()` functions resets all variables to their current default values. Most of these are numbers. But some can be managed from the command line.

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

[11]Note the clever way someone in the past discovered how to get a 1 exactly for `ii`={0,5,10,15}. Integer division yields the quotient, the mod function yields the remainder.

The `keyboard()` function converts the user's wishes into the current state of the RTICA by means of *illiGadgets*.[12] The code here is heavily aliased with `keyboard()` macros to make it easier for you to add and subtract gadgets. Since the precompiler handles these, their effective scope is global, but you can wait to define them to just before their first use.

The arguments of `keyboard()` are the key name and location of the mouse (which we don't use yet). The first thing we do is raise a flag in the key-vector `clefs[]` corresponding to the pressed key.[13] This way functions other than the current one can know which keys were pressed. We can even register that several keys were pressed. It is the duty of the `clefs[]` readers, such as `getbutton()` to unset the flags.

Five classes of illiGadgets are defined here by macros, though we use only the first three here. If you press the key described by the string `K`, then `PRESS(K,A,b)` will execute code fragment `A` if the key is capitalized, and otherwise `b`. The `TOGGLE, CYCLE` macros do as they are named. There is an integer and a float slider wannabe, which you can use advisedly. Be sure to experiment on a running illiSkel before reading further. Note, in particular, that repeater keys also repeat the action. This can cause trouble on particularly fast computers and other measures have to be introduced into the macros to unpredictable values.

This particular `keyboard()` has Stuart Levy's advanced gadgetry partially installed. With it, you do not need to hunt for a particular value, say of the `speed, torq` parameters[14] by pressing keys. You type a decimal number, such as 0.0, or .9, before pressing the (Q)-key, and that is the value of the parameter. To preserve the ability of changing a value quickly by a keypress, the `bump()` function multiplies the current value by $(1 + \epsilon)$. Dividing by that sum is close to multiplying by $1 - \epsilon$).

Exercise. The ways of wizards are not lightly to be tampered with. You could bump the `nose, focal, mysiz, wfar, gap, amb, and pwr` gadgets. But the multiplicative nature of bump suggests that it is inappropriate for some of these. Which? To demonstrate your understanding of SLevy's gadgets, write an additive bump, which is useful in integral gadgets, like cyclers.

The function of the keys is discussed later, at the place where their parameters and flags are implemented. But the last two gadgets 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.[15]

[12] We call these objects "gadgets" to distinguish them from "widgets", which serve the same purpose. Unlike widgeteers, we discourage pull-down menus, scroll-bars, simulated dashboards and the like, all of which distract you from using your eyes to look at the RTICA , and your hands from controlling its performance. Imagine puppeteers or airline pilots applying their hands and eyes to pull-down menus.

[13] We also guard against overrunning the array by making sure that key-numbers higher than 127 are registered in `clef[0]`.

[14] These influence the power behind your mouse. Slow computers need higher values of `speed, torq` and vice-versa.

[15] We encourage this and similar programming devices to reduce the waste of time in re-

A discussion of two more input functions belongs here but are deferred. The `specialkeybo()` is needed because the GLUT library recognizes that the keycode returned by the non-ascii keys are system dependent. The second is the function that interprets mouse gestures. Since we use the mouse exclusively for navigation its discussion is also deferred.

The `messages()` function manages the (optional) heads-up display of data on the graphics screen. Such *graffiti* grabs your attention and invites immediate experimentation independent of other documentation.[16] The indispensible companion feature to keyboard and mouse input is a verbal echo on the screen. While most keypresses also have a visible echo in the effect on the animation they produce, these tend to be qualitative. At screen location `(x,y)` the macro `LABEL(x,y,W,u)` write text `W`, formatted as in the printf family of C-functions, and uses one variable `u`. The `char2wall()` factor uses a handy character generator which will prove useful for placing text into a 3D scene as well. We use a 2-dimensional coordinate system for the graphics window, calibrated from lower-left at (0,0) to upper right at (3000,3000). Thus the bullseye is reliably in the dead center of whatever window we choose to look through.

messages()

There is a mnemonic embedded in the graffiti in that keys that can be pressed are enclosed in parentheses. The square-bracket indicates SLevy gadgets. If you want to add graffiti, note that the current spacing for letters is 70 units vertically.

The `autotymer()` is a poor-man's clockwork animator. It is designed to make it easy to simulate keypresses which manage parameters. When you toggle the (H)-key, the torus shrinks nearly into a rectangle, and then expands again to almost close up. Here is how the `autotymer()` does it. The first time it is called, a static flag called `first` is set. It is also set when the autotymer is re-called as part of the (Z)ap gadget. It is, once again set at the end of the animation so it can start over.

autotymer()

Each instance of the `TYME` macro has an internal counter which we can give a mnemonic name to, or you can just call each one `foo`, if you like. The first time through, each counter is set to its maximum value, given by the second argument of `TYME`. Once all clocks are wound up, `first=0`, and the animation begins. The first 150 times the `autotymer()` is called is decrements the `shrink` counter, and executes the code written into the third argument of `TYME`. Namely, the minimum angles `th0, ta0` are incrmented by one degree, and the maximum values are decremented. When this clock is exhausted (`shrink==0`, the second clock is addressed. It does nothing for 20 cycles, and then the third clock grows the rectangle back into the torus. You can insert and delete as many `TYME`s as you please, but the last one must be the `finish` clock, which resets `first`.

You should experimentally design an different animation before checking out just how the `TYME` macro does its job. It is a block with a static integer `cnt` which the `first` time is set to its `max` value. Thereafter, it is decremented. If `cnt==0`, it skips its act and passes the ball to the next clock. Otherwise,it

compiling or the waste of space in duplicating nearly identical versions of the same program.

[16] We retain the name, "messages", for back compatibility. The recommended new name, "graffiti", is more descriptive and less likely to be confused with other program function. In the older illiShell's heads-up writing was done differently on the console and the CAVE.

decrements `cnt`, does its `act`, and jumps to the `:Break` label. Yes, Virginia, contrary to what your high-school computer science teachers told you, C does have a *goto* command, which on rare occasions such as here, is quite useful.

6 Scenery generating functions.

The illiSkel has a two objects equipped with their own place matrices. They are the stars and the torus. The stars do not move across the screen in `TURNMODE`, but rotate with the object in `FLYMODE`. Being fixed to the virtual firmament, they do not translate. The torus, on the other hand, can be made to do all of these motions. Let us examine each of these objects before we look at their interaction.

The function `drawstars()` is self-initializing. The first time it is called it loads up `MANYSTARS` into the `star[]` array. First, it puts them into a unit cube, then it projects them to the unit sphere.[17] Then, and thereafter, it multiplies a duplicate of the current matrix on the the `GL_MODELVIEW` stack by its proper matrix `starmat[]`. It chooses a bluish-white color, and plots the stars by handing the OpenGL graphics library the 3-vector of floats marking the star's position on the unit sphere. You might prefer the wire-fram teapot to the stars. Try it!

Take a peek at `drawcons()` where `drawstars()` is called before the function `drawall()` which draws the rest. We do not want the stars to appear in front of stuff beyond the unit sphere. So `drawstars()` clears `GL_DEPTH_BUFFER_BIT`, effectively putting them at "infinity".

The `drawtor()` comes in two pieces. One vertex is created and lighted in `drawvert()`, given the θ, τ angular coordinates of the torus. This function is usually factored again into functions that calculate the position of the vertex, the color of the vertex, and the color modified by the angle between the light source and the normal of the surface. For compactness, all three are shoehorned into one funciton, making it easy to survey in one screen, but a chore to puzzle out the details. drawtor

Recall that a torus is a circle of circles. Indeed, so is a sphere, except that it is more properly a circle's worth (the equator) of semicircles (the meridians.) To make a torus, you merely push each meridian great circle a distance away from the polar axis. That's done in the very end of `drawvert()`. So, using polar coordinates, `nn[]` is both a position on the sphere and a normal to it there. The Lambert cosine `lmb` is a fraction that measures how close the point is to noon.[18] The *Lambert* or it diffuse-lighting model applies only this fraction to each componenet color. It is appropriate to an ideal surface which scatters light energy received equally in all directions. A *blackbody* absorbs all light, and a *mirror* reflects light in a purely specular way.

Since objects tend to look surreal without some backlighting, we re-value the back-facing `lmb` to 20 percent. Even for front facing patches, there is always

[17] Lucky that the random generator is likely to miss the dead center of the cube, eh?

[18] The sun is directly overhead where your normal points to it.

some ambient light, and we take this into account by not letting `lmb` drop below the `amb` parameter. You can interactively play with `amb` and the pseudo-specular `pwr`.

The specular point of a light-source on a surface is where the normal bisects the light and the viewing directions. We save ourselves extra computation by using the caustic point, which depends only on `lmb`. Furthermore, Phong lighting uses a power of the specular cosine. We use only the tangent of that power function at 1, since that's all that really matters. You should draw the graph of `spec` as a function of `lmb, pwr` to see what I mean. After deciding on the color of the vertex with Chris Hartman's illiPaint, we take the larger of the diffuse and the specular components.[19]

The illiPaint method is a poor-man's texture map. Each vertex is given a color whose RGB components are linear function of two variables `dog, cat`. These, in turn, are fractions of the way across the source patch $[\tau_0, \tau_1] \times [\theta_0, \theta_1]$. Thus, the red component varies linearly with `dog`, and the blue component varies contra-linearly with `cat`. Can you explain what the green component does?

Exercise. Knowing this little about `drawvert()` you should be able to create remarkable surfaces and painting effects. How about a toroidal surface both of whose radii vary with the parameter angles. For now only the `autotymer()` can change the source patch. Make this interactive with new gadgets. Make the surface fine and coarse meshed on demand.

All the `drawtor()` function has left to do is create the mesh of triangles which constitutes the numerical torus. Depending on how fine we want this mesh (good colors) and how nimbly we want it to move (few vertices) we will choose the number of `MERIDIANS, LATITUDES`. The torus is actually drawn as a series of parallel strips. And that's why we can have such a cheap `gap` making gadget.

Exercise. Install a second object in the same world as the torus. For instance, a cube or a hypercube in `drawcube()`.

The collection of all objects in the world (not the stars, though) are packed into `drawall()`. Once you have several objects you can practice your `IFCLICK` skills to place, size, and color them properly.

The final function is the display callback of the GLUT-libary, here called `drawcons()`. This is where we integrate our own functions with those of OpenGL and GLUT. Line by line, this is what happens each time through the glut-mainloop. The color buffer bit and the depth buffer bit are cleared. By default, the color is black. But you can put a nicer background in here with `glClearColor(0.1,0.2,0.3,0)`. We set one or two viewports, depending on whether the binocular flag is up or not. The viewport is usually coextensive with the drawing window. Here is an illustration why there is a difference: the same window has two viewports.

`drawcons()`

[19] Standard lighting models, such as the one proper to OpenGL, uses a polynomial of fractions rather than the maximal envelope of ambient, diffuse and specular components. This difference makes illiLight simpler to understand, easier to handle, and more robust, but all at the expense of sublety and versatility.

First we call for a perspective[20] projection with `glFrustum(xmin, xmax, ymin, ymax, near, far)`. In the raw, the six arguments of this funtion describe the frustum of a cone. The cone is produced by looking from the origin through a rectangular window[21] located a distance `near` down the *negative* z-axis, and clipped in `-far < z < -near < 0`. This eternally confusing convention guaranteess a right-handed coordinate system with the x-axis pointing to the right, and the y-axis pointing upward. You may notice how we have rewired the conical viewing box to be controlled by more geometrical parameters. First we make sure that the aspect ratio remains constant so that when you re-mouse the window, the object doesn't become oblongated. Second, we have a fraction `mysiz`, which couples the focal distance `near=mysize*focal` with the size of the rectangular canvas the view is painted on at `near`. The name derives from the task of making yourself so small that you can fly into an object without having it clipped by a large value of `near`. Try it! Play with the f(O)cal and the my s(I)ze gadgets.

We next modify affine (or `GL_MODEL_VIEW`) matrix, which will be multiplied into the projection matrix eventually. Starting with the identity we draw the stars. (Recall, the stars manage their own place with the `starmat[]`.) We then shift a little off center to account for the left-eye right-eye displacement, equal to twice the `nose` to eye distance. Now multiply by `aff[]`, the matrix that places the world relative to the origin in the way it would look if you moved about a stable world.[22]

We next draw the entire scene into the other eye, widen the viewport fully for the graffiti, and swap graphics buffers. This last call to GLUT draws all that is currently in the frame buffer to the screen, repeatedly, until a new frame complete and ready to replace the old one.

7 Navigation in Euclidean space.

Since we are construcing a graphics program, the management of the geometry pipeline for each frame is truly the main activity. The principle of OpenGL is very simple but its ramifications are extensive and can be mysterious and confusing without a clear mathematical idea to guide us. A geometrical figure is constructed out of points, line segments and planar patches. Even is the lines and patches appear curved (and they are intended to so appear), they are nevertheless made up of subliminally small straight lines and flat patches. A polylateral[23] curve is completely described the succession of vertices along it.

[20] Photographers and computer graphics systems that are less versatile than OpenGL favor on-axis or centered perspective with parameters based on the aperture angle, aspect ration, near and far. It is simple to implement this in the Frustum command.

[21] Indeed, this command is more aptly named "window" in IrisGL. It is unfortunate that this aptly named function had to be renamed in the OpenGL vocabulary. The term "window" had become too strongly tied to a resizeable viewport. So, avoid misunderstanding, OpenGL adopted the hardly euphonic word "fructrum", meaning a section of any solid between two parallel planes, often mispelled as "frutrum".

[22] The Copernican versus the Ptolemaic weltanschauung. OpenGL favors the latter.

[23] There is some difference between common math speak, and common computer graphics lingo. In graphics, a "polygon" means a patch enclosed by a polylateral. In geometry, a

Moreover, these vertices can be placed anywhere in space, so the polytaerals aren't planar, or convex, and we won't call them polygons.

The first problem to be solved in drawing surfaces is how to reduce their description to vertex streams. A surface made up of planar, polylateral patches is a *polyhedral surface*. Only a triangle (or "trilateral") is unambiguously planar. Consider four non-coplanar vertices in space and tetrahedron so formed. A particular succesion of the vertices separates the skin of the tetrahedron into two *dihedrals* corresponding to the two ways of triangulating a quadrilateral. Thus the triangle is the geometrical primitive, and all surfaces in computer graphics are triangulated.

There are two useful ways a succession of vertices $V_0, V_1, ..., V_n$ describes a succession of triangles. The `GL_TRIANGLE_STRIP` produces a ribbon with with two rails, one traced out by the even vertices, $V_0, V_2, V_4, ...$, and the other by the odd vertices, $V_1, V_3, V_5, ...$ The polylateral following the original vertex stream zig-zags back and forth between the rails, like the rungs of a bridge tressle. Of course, vertices may be repeated in the list. For example, if $V_4 = V_0, V_5 = V_1$ then the triangle strip covers a tetrahedron.

Exercise. An intriguing geometrical question is which polyhedral surfaces can be drawn by a single triangle strips, without covering a facet twice. Maybe you can so "bandage" every polyhedral surface. For instance, how would you "bandage" a cube, an octahedron, dodecahedron, and icosahedron? What about the graph $z = f(x, y)$ over a rectangular grid in the xy-plane? Hint: There is a "Columbus Egg" solution to this problem.

So, being vertex based, the geometry pipeline[24] feeds on a stream of vertices, bracketed by the appropriate `glBegin()` and `glEnd()`, performing a series of geometric transformations on each vertex, and producing the desired graphic on the screen. Regardless how these transformations are implemented (hardware, high or low level library functions, etc) they are represented by a 4x4 matrix of real numbers. The composition of operations is represented by matrix multiplication.

Each vertex stream, X, encounters three matrices at the top of their respective pushdown stacks: the current modelling (a.k.a. affine) matrix A, followed by the current projection matrix Π, followed by the viewport matrix, Ψ, about which we say very little. Thus, column mode[25] we see the effect of $\Psi\Pi AX$.

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

There is some expository redundancy here. Maybe saying it again isn't so bad for now.

"polylateral" is synonymous with a polygon. We will deviate from common speech only to avoid misunderstanding.

[24] Strictly speaking, a pipeline refers to a process which accepts new input before having finished its job on the previous input. Think of Ford's assembly line. But whether the pipeline is real or virtual, the conceptual structure is the same in OpenGL.

[25] After ten years of writing vectors as rows on the left of their matrix transformations in IrigGL, OpenGL reverted to the scientific notation which writes vectors as columns on the right of their matrix multipliers. This way, linear transformations compose like functions, from right-to left. You'll have to learn how to handle this source of confusion and errors effectively. Algebra is the right language for this.

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[26] viewing mode, we shift the first image to the left, E^R. Thus $\Pi E^R AX$ is the order of matrix multiplication on a vertex X for the right eye. 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, $\Pi E^L AX$. It is useful to associate the matrix products $\Pi^R = \Pi E^R$ and $\Pi^L = \Pi E^L$. 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. Small Multiples. Modify `drawcons()` 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. 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.[27]

The principal navigation is effected in `chaptrack()`, whose arguments are the input from the mouse (or the wand in the CAVE) as reported by the callback function `mousepushed()`. [28] The output is an update of the two affine matrices, `aff[], starmat[]`. `chaptrack(), mousepushed()`

The displacements, `dx, dy` of the mouse from the center of the window `(xwide/2, yhigh/2)` are clamped to have a 100 square-pixel dead zone, whihc makes the mouse less responsive. These displacements are interpreted as a small rotation about the x-axis, which points East, and the y-axis, which points

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

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

[28] For illustrative purposes, this callback includs some arcane code which translates which of the three buttons were pressed into a 3-bit flag `PAW`, a 1-bit flag, `SHIF` for the shift-button on the keyboard, and two integral coordinates of the mouse, `XX, YY`. It is not essential that `mousepushed()` be written this way, but you will find such constructions elsewhere and you may want to use them yourself for special purposes. In particular, in many cased bit-field flags are more efficient for passing and storing control messages than streams of entire integers.

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,[29] one starts by loading the identity matrix onto the stack. In the `FLYMODE`, which is easier to understand, the incremental rotation, U, on the stack is relative to the camera, which assumed to be at he origin of the world coordinate system.[30] Since the stars are only rotated, we make a duplicate of U on the stack with a `glPushMatrix(), glPopMatrix()` bracket. Inside this bracket, we update `starmat[]` on the left. It updates the star matrix by multiplication. Thus, if the mouse is parked slightly off the bullseye, a succession of small rotations are applied, so that after n-cycles later, a star is placed at $U_nU_{n-1}...U_2U_1X$.

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 cycles, the affine matrix, $A = I\, dA_n\, dA_{n-1} \ldots dA_2\, dA_1$, 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:

$$T(m_1)U_1T(m_2)U_2 = T(m_1 + U_1m_2)U_1U_2$$
$$T(m)U = UT(U^Tm)$$
$$(T(m)U)^{-1} = T(-U^Tm)U^T.$$

The superscript T emphasizes the fact that the inverse of a rotation matrix is just its transpose.[31]

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. In the column vector mode, the affine matrix $A = T(m)U$ is

$$\texttt{aff[]} = \begin{bmatrix} A[0] & A[4] & A[8] & A[12] \\ A[1] & A[5] & A[9] & A[13] \\ A[2] & A[6] & A[10] & A[14] \\ A[3] & A[7] & A[11] & A[15] \end{bmatrix}. = \begin{bmatrix} U_{00} & U_{10} & U_{20} & m_0 \\ U_{01} & U_{11} & U_{21} & m_1 \\ U_{02} & U_{12} & U_{22} & m_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}..$$

[29] Most pipeline operations are cumulative, i.e. they act on what is already there by matrix multiplication.

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

[31] 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.

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,

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

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

The last thing `chaptrack()` does is to rotate the light source direction by the inverse of the rotation component of `aff[]`. The the inverse of an affine matrix is

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

. Since we don't translate the stars, we need only the matrix multiplication by the transpose of to top-left 3x3 minor of `aff[]`.

Exercise. 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 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[32] projector for `win=1`, and a full screen for`win=2`. Since we now use the GLUT library, the remainder of `main()` has already been described a the beginning of this chapter.

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

[32]To fit into a standard NTSC video screen we use 640×480 pixels

```
/*****************************2002************************************/
/****       skel.c = OpenSkelGlut.c = Noosh97 with CAVE removed    ****/
/****      (C) 1994--2002 Board of Trustees University of Illinois ****/
/****      A Model Real-Time Interactive  C/OpenGL/GLUT  Animator  ****/
/****    George Francis, Stuart Levy, Glenn Chappell, Chris Hartman****/
/****    e-mail  gfrancis@math.uiuc.edu : revised 2jan02 by gkf    ****/
/*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <gl\glut.h>       /* sgi  <glut.h> */
#include <sys/timeb.h>     /* sgi  <sys/time.h> */
#include <math.h>
#pragma warning (disable:4305)       /* constant double-to-float  */
#define  MAX(x,y)         (((x)<(y))?(y):(x))
#define  MIN(x,y)         (((x)<(y))?(x):(y))
#define  CLAMP(x,u,v)     (x<u? u : (x>v ? v: x))
#define  ABS(u)           ((u)<0 ? -(u): (u))
#define  FOR(a,b,c)       for((a)=(b);(a)<(c);(a)++)
#define  DOT(p,q)         ((p)[0]*(q)[0]+(p)[1]*(q)[1]+(p)[2]*(q)[2])
#define  NRM(p)           sqrt(DOT((p),(p)))
#define  M_PI             (355./113)   /* sgi already defins \pi   */
#define  DG            M_PI/180
#define  S(u)          sin(u*DG)
#define  C(u)          cos(u*DG)
#define  CLAMP(x,u,v) (x<u? u : (x>v ? v: x))
#define  random        rand        /* library dependent name  */
#define IFCLICK(K,a){static ff=1; if(getbutton(K))ff=1-ff; if(ff){a} }
#define MERIDIANS 32
#define LATITUDES 32
#define MANYSTARS 10000
/************************** global variables *************************/
int win=1;                 /* used once to choose window size         */
float gap, gap0=1.;        /* deFault() uses gap0 set by arguments()  */
float lux[3]={1.,2.,3.};   /* world light non unit  vector */
float luxx[3];             /* object space  direction vector          */
float amb, pwr;        /*ambient fraction, pseudo-specular power      */
float mysiz,speed, torq, focal, wfar; /* navigation control variables */
unsigned int PAW,XX,YY,SHIF;  /* used in chaptrack gluttery           */
int xwide,yhigh;               /* viewportery width and height       */
int mode,morph,msg,binoc;      /* viewing globals */
int th0, th1, dth, ta0, ta1, dta;  /* torus parameters                */
#define FLYMODE  (0)         /* yellow: turns around head as center   */
#define TURNMODE (1)         /* purple: turns around object center    */
int ii, jj, kk;  float tmp, temp;          /* saves gray hairs later  */
float aff[16], starmat[16], mat[16];   /* OpenGL placement matrices   */
int binoc;                       /* flag for binocular stereo        */
```

```
float nose;                         /* to eye distance in console       */
char clefs[128];                    /* which keys were pressed last     */
/********************** steering ************************************/
void autotymer(int reset){                          /* cheap animator  */
#define  TYME(cnt,max,act) {static cnt; if(first)cnt=max; else\
                           if(cnt?cnt--:0){ act ; goto Break;}}
  static first = 1;              /* the first time autymer is called  */
  if(reset)first=1;              /* or if it is reset to start over   */
  TYME( shrink , 150 , th0++;th1--;ta0++;ta1--)
  TYME( pause  , 20, 0  )
  TYME( grow   , 150,th0--;th1++;ta0--;ta1++)
  TYME( dwell  , 30, 0  )
  TYME(finish  , 1 , first = 1 )  /* this TYME must be the last one   */
  first = 0;
  Break:   ;                     /* yes Virginia, C has gotos         */
}
/***********************************************************************/
void deFault(void){         /* (Z)ap also restores these assignments  */
  th0=5; th1=355;  ta0=5; ta1=355; gap = gap0;    /* torus parameters */
  msg=1; binoc=0; nose=.06; mode=TURNMODE;       /* gadget parameters */
  speed=.02; torq=.02; focal = 2.; wfar=13; mysiz=.01; morph=0;
  FOR(ii,0,16) starmat[ii]=aff[ii] = (ii/4==ii%4); /* identity matrix */
  amb = .3; pwr = 10;                           /* lighting params   */
  tmp=NRM(lux); FOR(ii,0,3)lux[ii] /= tmp;  /* normalize light vector */
  aff[12]=0; aff[13]= 0; aff[14]= -4.2; /* place where we can see it  */
  autotymer(1);          /* reset autotymer to start at the beginning */
}
/************************** scenery *********************************/
void drawvert(int th, int ta){    /* make one properly lighted vertex */
  float  bb,gg,rr;
  float  lmb,spec,nn[3], dog, cat;
  /* radius of unit sphere is also unit normal to the torus           */
  nn[0] = C(th)*C(ta);
  nn[1] = S(th)*C(ta);
  nn[2] =       S(ta);
  /* illiLight by Ray Idaszak 1989 uses max{amb*lmb, rgb*lmb, spec}   */
  lmb = DOT(nn,luxx); lmb =(lmb<0 ? .2 : lmb); lmb = MAX(amb, lmb);
  spec = CLAMP((1.1 - pwr+pwr*lmb), 0., 1.);
  /* illiPaint by Chris Hartman 1993 maps R2(cat,dog)->R3(r,g,b)      */
  dog = (ta-ta0)/(float)(ta1-ta0); cat = (th-th0)/(float)(th1-th0);
  rr =  MAX(spec, lmb*dog);
  gg =  MAX(spec, lmb*(.25 + ABS(cat -.5)));
  bb =  MAX(spec, lmb*(1 - cat));
  glColor3f(rr,gg,bb);
  /* torus has unit small diameter and unit big radius                */
  glVertex3f( C(th) + .5*nn[0],
              S(th) + .5*nn[1],
                0   + .5*nn[2]);
} /* end drawvert */
/***********************************************************************/
void drawtor(void){ /* illiTorus with gaps */
  int th, ta;
  dth = (int)((th1-th0)/MERIDIANS);  /*   this many   meridian strips */
```

```
  dta = (int)((ta1-ta0)/LATITUDES);  /* and  triangle pairs per strip */
  for(th=th0; th < th1; th += dth){
   glBegin(GL_TRIANGLE_STRIP);
     for(ta = ta0 ; ta < ta1 ; ta += dta ){
         drawvert(th,ta); drawvert(th+gap*dth,ta);}
   glEnd();
  }/* end for.theta loop */
}/* end drawtor */
/*************************************************************************/
void drawcube(void){ /* transfer from skel.c as an exercise  */ }
/*************************************************************************/
void drawall(void){ drawtor(); drawcube();}
/*************************************************************************/
void drawstars(void){  /* replace with SLevy's much prettier stars    */
  static float star[MANYSTARS][3]; static int virgin=1;
  if(virgin){             /* first time through set up the stars      */
     FOR(ii,0,MANYSTARS){ /* in a unit cube then  on unit sphere      */
       FOR(jj,0,3)star[ii][jj]  =(float)random()/RAND_MAX - 0.5;
       tmp=NRM(star[ii]); FOR(jj,0,3)star[ii][jj]/=tmp;
     }
   virgin=0; /* never again */
  }
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();           /* optional insurance or superstition    */
    glMultMatrixf(starmat);
    glColor3f(0.8,0.9,1.0);
    glBegin(GL_POINTS);
      FOR(ii,0,MANYSTARS)glVertex3fv(star[ii]);
    glEnd();
  /* glutWireTeapot(1);  if you prefer one on the firmament instead  */
  glPopMatrix();            /* optional insurance or superstition    */
  glClear(GL_DEPTH_BUFFER_BIT); /* put the stars at infinity         */
}
/*********************** steering ***********************************/
void arguments(int argc,char **argv){           /* Pat Hanrahan 1989 */
      while(--argc){++argv; if(argv[0][0]=='-')switch(argv[0][1]){
      case 'w': win =atoi(argv[1]); argv++; argc--; break;
      case 'g': gap0 =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;
   }}}
/*************************************************************************/
int number, hasnumber, decimal, sign;  /* globals for SLevy's gadgets */
   /* these are assigned in keyboard() but used by these factor fcns  */
float getnumber(float dflt){    /*  from keyboard, factor of bump()   */
     float v = (sign ? -number : number);  /* positive or negative nr */
     if(!hasnumber) return dflt;       /* if no new nr use old on     */
     return decimal>0 ? v/(float)decimal : v;
}
void bump(float *val, float incr){          /* SLevy 98 */
  float by = fabs(incr);   /* wizard speak ... best not mess with it  */
  char fmt[8], code[32];
```

```
  int digits = 1;
  if(hasnumber) {
    *val = getnumber(0);
    return;
  }
  if(by <= .003) digits = 3;
  else if(by <= .03) digits = 2;
  sprintf(fmt, "%%.%de", digits);
  sprintf(code, fmt, *val * (1 + incr));
  sscanf(code, "%f", val);
}
/********************from SLevy 2jan02 ******************************/
int getbutton(char key) {
    int uu = clefs[key & 127]; clefs[key & 127]=0; return uu;
}
/*************************************************************************/
void keyboard(unsigned char key, int x, int y){
   clefs[key&127]=1;  /* globalize the keys that were pressed         */
#define  IF(K)            if(key==K)
#define  PRESS(K,A,b)     IF(K){b;} IF(K-32){A;}  /* catch upper case */
#define  TOGGLE(K,flg)    IF(K){(flg) = 1-(flg); }
#define  CYCLE(K,f,m)   PRESS((K),(f)=(((f)+(m)-1)%(m)),(f)=(++(f)%(m)))
#define  SLIDI(K,f,m,M)   PRESS(K,(--f<m?m:f), (++f>M?M:f))
#define  SLIDF(K,f,m,M,d) PRESS(K,((f -= d)<m?m:f), ((f += d)>M?M:f))
/* Only 127 ASCII chars are processed in this GLUT callback function  */
/* Use the specialkeybo function for the special keys                 */
   IF(27) { exit(0); }                          /* ESC exit          */
   TOGGLE('v',binoc);                           /* cross-eyed STEREO */
   TOGGLE(' ',mode);                            /* space key         */
   TOGGLE('h',morph);                           /* autotymer on/off  */
   CYCLE('w',msg,3);           /* writing on/off/speedometer+bullseye */
   PRESS('n', nose -= .001 , nose += .001 );    /* for binoculars    */
   PRESS('s', bump(&speed,.02), bump(&speed,-.02));/* flying speed    */
   PRESS('q', bump(&torq, .02), bump(&torq, -.02)); /* turning speed  */
   PRESS('o', focal *= 1.1 , focal /= 1.1)      /* telephoto         */
   PRESS('i', mysiz /= 1.1, mysiz *= 1.1)       /* rescale the world */
   PRESS('p', wfar *= 1.01 , wfar   /= 1.01)   /* rear clipping plane */
   PRESS('z', deFault(), deFault());            /* zap changes       */
   PRESS('g',gap /= .9, gap *= .9);            /* gap parameter      */
   PRESS('a',amb /= .9, amb *= .9);            /* ambient fraction   */
   PRESS('r',pwr /= .9, pwr *= .9);            /* pseudo-spec power  */
/********** SLevy's parser creates the input decimal ******************/
   if(key >= '0' && key <= '9'){         /* if key is a digit numeral */
          hasnumber = 1; number = number*10+key-'0'; decimal *= 10; }
   else if(key == '.') { decimal = 1; }  /* it's a decimal !          */
   else if(key == '-') { sign = -1; }    /* it's negative  !          */
   else { hasnumber = number = decimal = sign = 0;} /* erase mess     */
   glutPostRedisplay();                /* in case window was resized */
}
/*************************************************************************/
void specialkeybo(int key, int x, int y){
  clefs[0]= key ;
  switch(key){    /*  HOME END PAGE_DOWN RIGHT F1 etc  see glut.h    */
```

```
  case GLUT_KEY_F1:  th0++; th1--; break;
  case GLUT_KEY_F2:  th0--; th1++; break;
  /*default: fprintf(stderr,"nonASCII char [%d] was pressed.\n", key);*/
  }
}
/*************************************************************************/
float speedometer(void){                        /* this one is for win32*/
  double dbl; static double rate; static int ii=0;
  static struct _timeb lnow, lthen;
    if(++ii % 8 == 0){                 /* 8 times around measure time */
   _ftime(&lnow);
dbl =  (double)(lnow.time - lthen.time)
 +(double)(lnow.millitm - lthen.millitm)/1000;
lthen = lnow;  rate = 8/dbl;
      }
    return((float)rate);
}
/*************************************************************************/
void char2wall(float x,float y,float z, char buf[]){
     char *p; glRasterPos3f(x,y,z);
     for(p = buf;*p;p++) glutBitmapCharacter(GLUT_BITMAP_9_BY_15,*p);
}
/*************************************************************************/
void messages(void){char buf[256];
  /* console messages are done differently from cave */
#define  LABEL(x,y,W,u) {sprintf(buf,(W),(u));char2wall(x,y,0.,buf);}
  glMatrixMode(GL_PROJECTION); glPushMatrix();  /* new projection matrix */
  glLoadIdentity(); gluOrtho2D(0,3000,0,3000);  /* new 2D coordinates */
  glMatrixMode(GL_MODELVIEW); glPushMatrix(); glLoadIdentity();
      if(mode==TURNMODE) glColor3f(1.,0.,1.); else glColor3f(1.,1.,0.);
      LABEL(1500,1500,"%s","o"); /* place a bullseye dead center */
      LABEL(80,80,"%4.1f fps",speedometer());
      if(msg==2)return;  //try this
      LABEL(80,2840,\
      "(ESC)ape (V)Binoc (MAUS2)Fore (BAR)%s (H)omotopy (W)riting",
             mode?"TURNMODE":"FLYMODE");
      LABEL(10,10,"illiSkel-2002 by Francis, Levy, Bourd, Hartman,\
& Chappell, U Illinois, 1995..2002 %s","");
      LABEL(80,2770,"(N)ose   %0.3f",nose);
      LABEL(80,2700,"[S]peed  %0.4f",speed);
      LABEL(80,2630," tor[Q] %0.4f",torq);
      LABEL(80,2560,"near clipper %g", mysiz*focal);
      LABEL(80,2490,"f(O)cal factor %g",focal);
      LABEL(80,2420,"my s(I)ze %.2g",mysiz);
      LABEL(80,2350,"far cli(P)per= %.2g",wfar);
      LABEL(80,2280,"(Z)ap %s","");
      LABEL(80,2210,"(G)ap %.2g",gap);
      LABEL(80,2140,"(A)mb %.2g",amb);
      LABEL(80,2070,"pw(R) %.2g",pwr);
    glPopMatrix();
    glMatrixMode(GL_PROJECTION); glPopMatrix();
}
/************************ navigation ********************************/
```

```
void chaptrack(int paw,int xx,int yy,int shif){/* Glenn Chappell 1992 */
   long dx,dy;
   dx = xx -.5*xwide; dx = abs(dx)>5?dx:0;        /* 5 pixel latency  */
   dy = yy -.5*yhigh; dy = abs(dy)>5?dy:0;
   glMatrixMode(GL_MODELVIEW); glPushMatrix(); glLoadIdentity();
   if(mode==TURNMODE) glTranslatef(aff[12],aff[13],aff[14]);
   glRotatef(dx*torq,0.,1.,0.); glRotatef(dy*torq,1.,0.,0.);
   if(paw&(1<<GLUT_RIGHT_BUTTON ))glRotatef(shif?-10:-1,0.,0.,1.);
   if(paw&(1<<GLUT_LEFT_BUTTON  ))glRotatef(shif?10:1,0.,0.,1.);
   if(mode==FLYMODE){
      glPushMatrix();
      glMultMatrixf(starmat);
      glGetFloatv(GL_MODELVIEW_MATRIX,starmat);
      glPopMatrix(); }
   if(paw&(1<<GLUT_MIDDLE_BUTTON))glTranslatef(0.,0.,shif?-speed:speed);
   if(clefs[0]==GLUT_KEY_UP) glTranslatef(0.,0., speed);
   if(clefs[0]==GLUT_KEY_DOWN) glTranslatef(0.,0., -speed);
   if(clefs[0]==GLUT_KEY_LEFT) glTranslatef(-speed,0.,0.);
   if(clefs[0]==GLUT_KEY_RIGHT) glTranslatef(speed,0.,0.);
   if(clefs[0]==GLUT_KEY_PAGE_UP) glTranslatef(0., speed,0.);
   if(clefs[0]==GLUT_KEY_PAGE_DOWN) glTranslatef(0.,-speed,0.);
   if(mode==TURNMODE) glTranslatef(-aff[12],-aff[13],-aff[14]);
   glMultMatrixf(aff);
   glGetFloatv(GL_MODELVIEW_MATRIX,aff);
   FOR(ii,0,3){luxx[ii]=0; FOR(jj,0,3)luxx[ii] +=aff[ii*4+jj]*lux[jj];}
   glPopMatrix();
}
/************************ scenery ***********************************/
void reshaped(int xx, int yy){xwide=xx ; yhigh=yy;} /*win width,height*/
/********************************************************************/
void drawcons(void){ float asp =(float)xwide/yhigh; /* aspect ratio   */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glClearColor(0,0,0,0);   /* base color, try (.1,.2,.3,0.)           */
  if(binoc) glViewport(0,yhigh/4,xwide/2,yhigh/2);
  glMatrixMode(GL_PROJECTION); glLoadIdentity();
  glFrustum(-mysiz*asp,mysiz*asp,-mysiz,mysiz,mysiz*focal,wfar);
  glMatrixMode(GL_MODELVIEW); glLoadIdentity();
  drawstars();
  glTranslatef(-binoc*nose,0.0,0.0);
  glMultMatrixf(aff);
  drawall();
  if(binoc){
      glViewport(xwide/2,yhigh/4,xwide/2,yhigh/2);
      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      drawstars();
      glTranslatef(binoc*nose,0.0,0.0);
      glMultMatrixf(aff);
      drawall();
    }
  glViewport(0,0,xwide,yhigh);
  if(msg) messages();
  glutSwapBuffers();
```

```
}
/************************* steering ********************************/
void idle(void){            /* do this when nothing else is happening */
   if(morph) autotymer(0);                        /* advance autotymer */
   glutPostRedisplay();                           /* redraw the window */
   IFCLICK('=',chaptrack(PAW,XX,YY,SHIF);)        /* bypass navigation */
   glDisable(GL_DEPTH_TEST);                    /* bypass depth buffer */
   IFCLICK('-',glEnable(GL_DEPTH_TEST); )       /* bypass depth buffer */
}
/*******************************************************************/
void mousepushed(int but,int stat,int x,int y){
  if(stat==GLUT_DOWN) PAW |= (1<<but); /*key came down and called back*/
  else PAW &= (-1 ^ (1<<but));              /* on the wayup erase flag*/
  XX=x; YY=y;        /* position in window coordinates (pos integers) */
  SHIF=(glutGetModifiers()==GLUT_ACTIVE_SHIFT)?1:0;  /* shift down too*/
}
/*******************************************************************/
void mousemoved(int x,int y){ XX=x; YY=y; }
/***************** one ring to rule the all ************************/
int main(int argc, char **argv){
   arguments(argc,argv);                        /* from the commandline */
   deFault();                          /* values of control parameters */
       glutInit(&argc, argv);
       glutInitDisplayMode(GLUT_DOUBLE|GLUT_DEPTH);
       switch(win){
           case 0: break;                    /* manage your own window */
           case 1: glutInitWindowSize(640, 480);
                   glutInitWindowPosition(100,100); break;
           case 2: glutInitWindowPosition(0,0); break;
         }
       glutCreateWindow("<* illiSkel 2002 in C/OpenGL/GLUT *>");
       if(win==2) glutFullScreen();
       glEnable(GL_DEPTH_TEST);                     /* enable z-buffer */
       glutDisplayFunc(drawcons);
              /* the following are optional for interactive control  */
       glutKeyboardFunc(keyboard);
       glutSpecialFunc(specialkeybo);
       glutMouseFunc(mousepushed);
       glutMotionFunc(mousemoved);
       glutPassiveMotionFunc(mousemoved);
                                    /*  beyond here all are needed  */
       glutReshapeFunc(reshaped);
       glutIdleFunc(idle);
       glutMainLoop();
}
/*******************************************************************/
```