```
/*****************************************************/
/****************** oc1.c ****************************/
/*****************************************************/
/* modified from 1988 SGI GUIDE pg 2-21 on 9apr89     */
/* this version (C)1996, G Francis, 9sep96            */
#include <device.h>
#include <gl.h>
float vv[6][3] = { { 1.0, 0.0, 0.0 },
                   { 0.0, 1.0, 0.0 },
                   { 0.0, 0.0, 1.0 },
                   {-1.0, 0.0, 0.0 },
                   { 0.0,-1.0, 0.0 },
                   { 0.0, 0.0,-1.0 } };
drawit(){bgntmesh();
         cpack(0x0000ff);               v3f(vv[0]);
         cpack(0x00ff00);               v3f(vv[1]);
         cpack(0xff0000);  swaptmesh(); v3f(vv[2]);
         cpack(0xff00ff);  swaptmesh(); v3f(vv[4]);
         cpack(0x00ffff);  swaptmesh(); v3f(vv[5]);
         cpack(0x00ff00);               v3f(vv[1]);
         cpack(0xffff00);               v3f(vv[3]);
         cpack(0xff0000);               v3f(vv[2]);
         cpack(0xff00ff);  swaptmesh(); v3f(vv[4]);
         cpack(0x00ffff);               v3f(vv[5]);
         endtmesh();
}
main(){ Matrix id, aff;
  {int ii,jj; for(ii=0;ii<4;ii++)for(jj=0;jj<4;jj++)
       id[ii][jj] = aff[ii][jj] = (ii==jj)?1.0:0.0;}
     winopen("octahedron");
     zbuffer(1); doublebuffer(); RGBmode();  gconfig();
     while(getbutton(ESCKEY)==0){
          int dx = (getvaluator(MOUSEX)-640)/16;
          int dy = (getvaluator(MOUSEY)-512)/16;
          loadmatrix(id);
          rotate(dx ,'y'); rotate(-dy,'x');
          multmatrix(aff); getmatrix(aff);
          reshapeviewport();
          ortho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
          multmatrix(aff);
          cpack(0); clear(); zclear();
          drawit(); swapbuffers();
     } /*end while*/
} /* end it all */
```

```
    /*****************************************************/
    /*****       oc1.c with GLUT and OpenGL      ********/
    /*****       (C)1997 George Francis          ********/
    /*****       Rewritten oc1.c, Alex Bourd     ********/
    /*****       and Matthew Stiak 9 Oct 1996     *******/
    /*****       TeX revision 30 December 1996    *******/
    /* Mark J. Kilgard, OpenGL:Programming for the   ****/
    /*****  X Window System,  Addison-Wesley, 1996.  ****/
    /*****************************************************/
    /* cc $1 -o $2 -g -I/<path>/glut -L/<path>/glut \ ***/
    /* -lglut -lGL -lGLU -lXmu -lXi -lXext -lX11 -lm  ***/
    /*****************************************************/

#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <GL/glut.h>
#define  ESC        27  /* ASCII */
#define  FOR(i,a,b) for(i=a;i<b;i++)

GLfloat vv[6][3] = { { 1.0, 0.0, 0.0 },   /* vertices for  */
                     { 0.0, 1.0, 0.0 },   /* the octahedron*/
                     { 0.0, 0.0, 1.0 },
                     {-1.0, 0.0, 0.0 },
                     { 0.0,-1.0, 0.0 },
                     { 0.0, 0.0,-1.0 } };
GLfloat aff[16];     /* the affine matrix as a linear array  */
GLint MouseX,MouseY; /* globalize mouse in keyboard callbacke*/
/*************************************************************/
void display(void){  /* reset drawing buffer each time around*/
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);  /* multibuffer mode in OpenGL */
  glLoadMatrixf(aff);          /* a.k.a. ModelViewMatrix     */
  glBegin(GL_TRIANGLE_FAN);    /* compare with tmesh         */
     glColor3f(0.,1.,0.);  glVertex3fv(vv[1]);
     glColor3f(0.,0.,1.);  glVertex3fv(vv[0]);
     glColor3f(0.,1.,1.);  glVertex3fv(vv[5]);
     glColor3f(1.,1.,0.);  glVertex3fv(vv[3]);
     glColor3f(1.,0.,0.);  glVertex3fv(vv[2]);
     glColor3f(0.,0.,1.);  glVertex3fv(vv[0]);
  glEnd();                        /* zonohedra need two fans  */
  glBegin(GL_TRIANGLE_FAN);
     glColor3f(1.,0.,1.);  glVertex3fv(vv[4]);
     glColor3f(0.,0.,1.);  glVertex3fv(vv[0]);
     glColor3f(0.,1.,1.);  glVertex3fv(vv[5]);
     glColor3f(1.,1.,0.);  glVertex3fv(vv[3]);
     glColor3f(1.,0.,0.);  glVertex3fv(vv[2]);
     glColor3f(0.,0.,1.);  glVertex3fv(vv[0]);
  glEnd();
  glutSwapBuffers();
}
```

```
/*************************************************************/
void keyboard(unsigned char key, int x, int y){
  switch(key){ /* this callback is only for common ascii keys */
     case ESC: fprintf(stderr," Thanks for using GLUT ! \n");
               exit(0); break;}
/*   case ' ': the space-bar was pressed                     */
/*   case 'A': a capital A was pressed                       */
}
/*************************************************************/
void chaptrack(void){     /* idle callback factor acts on aff */
  GLfloat dx=(MouseX-200.)/32.;   /* middle of initial window */
  GLfloat dy=(MouseY-200.)/32.;
  glMatrixMode(GL_TEXTURE);   /* handy stack for matrix math  */
     glLoadIdentity();
     glRotatef(dx, 0.0,1.0,0.0);  /* (angle,axis) of rotation */
     glRotatef(dy,-1.0,0.0,0.0);
     glMultMatrixf(aff);
     glGetFloatv(GL_TEXTURE_MATRIX,aff);    /* no glGetMatrix */
}
/*************************************************************/
void mousemotion(int x,int y){ MouseX=x; MouseY=y;
}                                           /* an OpenKludge  */
/*************************************************************/
void idle(void){ /* what glutMainLoop does when no interrupts */
     glutPostRedisplay(); /*  redraw window each time through */
     chaptrack();
}
/*************************************************************/
int main(void){
      /* make your own identity matrix with Kronecker's delta */
  int ii,jj; FOR(ii,0,4)FOR(jj,0,4)aff[ii*4+jj]=(ii==jj);

  glutInitWindowSize(400, 400);
  MouseX=MouseY=200;
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("<< oc1.c in OpenGL with GLUT >>");
      /* following callbacks are executed by the glutMainLoop */
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutPassiveMotionFunc(mousemotion);   /*not done when still */
  glutIdleFunc(idle);
     /* enables the already initialized graphics states       */
  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_SMOOTH);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
  glutMainLoop();
  return 0; /* ANSI C requires main to return an integer */
}
```

illiLesson 1.

George Francis

September 24, 2009

1 Contents

Here is a fast version of the first and second lesson. It consists of a summary explanation of the **oc1.c** RTICA[1] which draws a colorful octahedron that responds to the mouse. This RTICA[2] might be considered the "Hello, World" program for geometrical graphics. We suggest some variants and include a generalization, **tr1.c**, which displays a lighted torus.

We suggest you have a separated copy of the code handy as you study this lesson for the first time.

2 illiOctahedron

The end of a line,called `newline` and written `\n`, indentation, tabs and multiple spaces, all of which influence the typographical positioning are called *white space* in C and are ignored by the compiler. Commentary between /* and */ is also ignored. On the other hand, the pound symbol, `#`, indicating a compiler directive, must be preceeded by a newline, i.e. it must appear at the beginning of a line. The `#include` instructs the compiler to insert an entire segment of code. When the name of this code is between angle brackets, <>, it refers to some standard library header file, marked by the `.h` suffix. Thus `gl.h` is the name of the file that contains the definitions of all the graphics related commands. The mouse is not part of graphics, its commands are defined in `device.h` header file.

This program has only one global variable, `vv`, and that is an array of floating point numbers arranged into a 6x3 matrix. It represents the six points at

[1] Real-time, interactive computer animation.

[2] It was motivated by a curious error in a similar program on page 2-21 of 1988 *Graphics Library Programming Guide* for the Iris-4D Series. That one got the t-mesh algorithm all wrong, although the result looked perfectly alright. An object lesson in graphics programming! Besides, the SGI program was not interactive.

the corners of an octahedron. The name[3] of the array is `vv`, the name of the second vertex is `vv[1]`, and its third coordinate is `vv[1][2]`. Observe that in C we start counting at 0. In C it is possible to assign an entire array at the time of its definition.

Every C-program has a subroutine called `main()`. It is the first piece of the program to be executed, and it could be the only one. But it is customary to "factor" computational recipes into subroutines which is referred to as **functions** in C, and are recognizable as such by the parentheses. The code that defines a function is written between braces. If there is only one command in the definition, you may omit the braces, but not the semi-colon. The semi-colon may be omitted *after* a closing brace, but not *before*. The code between braces is called a `block`.

Although `OpenGL` is a direct descendant of the Iris `gl` graphics library we use here, it is no more just a renaming of the commands than Chaucer's works are just misspelled modern English. Unfortunately, `OpenGL` favors function names that just won't fit into this margin. We shall include a listing of `OpenGL` version, oc1.C and tr1.C, at the end of the chapter.

This program has only one function other than `main()`. The `drawit()` function draws the eight faces of the octahedron in the manner prescribed by the t-mesh syntax. The first command, `bgntmesh()`, begins the t-mesh. This is followed an alternation of a color command, such as `cpack(0xaabbggrr)`, and the name of a vertex, occasionally interrupted by the `swaptmesh()` command. The argument for `cpack()` is a long integer,[4] is usually written as a hexadecimal numeral, with the `ox` prefix. The highest byte is a transparency index for the alpha channels, a feature we do not discuss here. The next is the blue byte, here `bb`[5]. Then comes the green byte `gg`, then the red byte `rr`. Note that `drawit()` assigns red to the (1,0,0) vertex, green to the (0,1,0) vertex etc.

The `v3f(vv[0])` command tells the `tmesh` algorithm to process a 3D vertex who coordinates are given as floats. Nothing is drawn at this time. The second vertex is `vv[1]`, and still nothing is drawn even though we now know the *base edge* of the first triangular facet of the octahedron. As soon as the next vertex is chosen in space, a facet can be drawn. But the iteration is incomplete until one of the two newly formed edges is chosen to be the base edge of the next facet, as specified by the fourth vertex chosen. That is why the third vertex is preceded by a `swaptmesh()` command in this case. With or without this modifier, a triangle with a red, a green and a blue vertex

[3]In order to preserve, even emphasize the difference between orthodox mathematical notation and the custom of computer science, we avoid the use of single letters for names of variables. Moreover, some Unix editors do not find single letter names efficiently. On the other hand, we discourage the vogue of naming an object of a category by the name of the category. Thus, calling a vertex `vertex` is as uncouth as christening ones daughter "Daughter".

[4]The use of the variable types, `int` and `float`, is usually adequate for graphics, and considerably more mnemonic than `long, short, double`, etc.

[5]Which happens to be the numeral 187 in hexadecimal.

is drawn, and the colors are linearly averaged over the triangle. This is the default Gouraud shading in gl.

The t-mesh algorithm describes a triangulated surface in an unambiguous way.[6] If a sequence of more than three vertices is t-meshed, already the fourth vertex could form the second triangle *either* with the edge joining the 3rd vertex to the second, or the edge joining the 3rd vertex to the first. Since our triangles are oriented, we shall call the former edge (the one joining the 3rd vertex to the 2nd) the *right* edge, the one joining the 3rd vertex to the first vertex is the *left* edge, and the *base* edge joins the first two vertices of the triangle. To optimize the drawing of long, rectangular strips, a stream of vertices without the `swaptmesh()` modifier alternates in chosing the right-left-right-left... edges of the current triangle to be the base edge of the next triangle. If this scheme is to be violated, the `swaptmesh()` command says to use the other edge to be the next base edge, and, in effect to restart the meshing process. Thus, the three successive `swaptmesh()` commands means that the succession of triangles passes to the left of `vv[2]`, then to the left of `vv[4]` and once more to the left of `vv[5]`, but then to the right of `vv[1]`, left of `vv[3]` etc.

In `OpenGL` there is no `swaptmesh`. Instead a second primitive, the *triangle fan,* was included. A zonohedron, like all of the Platonic solids, and many more irregular polyhedra, can be constructed by means of two triangle fans.

The structure of the `main()` function of a graphics program has two essential parts, the *overture* and the *eternal loop.* In the former the state is initialized in which the latter draws frame after frame until it is interrupted or terminated.

In 'gl-speak' the word *matrix* invariably means a 4x4 array of floats, and there is a convenient way of creating such objects with the type function `Matrix.` We shall use two 4x4 matrices: the constant identity matrix, `id`, and the variable affine matrix, `aff`. The affine matrix represents an element in the group of Euclidean motions (rotations and translations) which is applied to the coordinates of the octahedron in each frame.

The next five commands set up the graphics environment. A window in which the figure will be drawn, and which has the label "octahedron", is created by `winopen("octahedron")`. The `zbuffer(1)` turns on z-buffering. Changing the argument turns it off, `zbuffer(0)`. As each new pixel is about to be drawn into the frame buffer, the z-buffer algorithm checks its z-coordinate (the one away from you) against that of the pixel already there. This way, only the nearer one is visible, and the further ones are hidden. In the `doublebuffer()` mode the iris maintains two distinct frames, the one

[6]But it is non-trivial and requires care to get it right. Indeed, the error in the octahedron example which inspired the present lesson was in the t-meshing. Unhappily, the difficulty of explaining t-meshes was solved in `OpenGL` by omitting this useful feature. While one can live without it, the resulting code is less elegant.

that is currently seen on the screen, and another invisible one, into which the next picture is currently being drawn. The doublebuffer mode may be disabled with the the `singlebuffer()` command. This is not prudent except for special artistic or diagnostic effects. In the `RGBmode()`, each vertex is assigned the three current color values, as specified by the most recent `cpack()` command, or its equivalent. The graphics initializations are 'confirmed' by the `gconfig()` command, which starts the graphics machinery which is called the *geometry pipeline* on an Iris. A common programming error is to call for a function which resides in the graphics library prior to enabling the geometry pipeline.

We use a double loop to initialize both `id` and `aff` to be identity matrices of the Kronecker delta function. It gives us the opportunity to introduce two very useful features of C: *local variables* and *conditionally valued functions.*

A variable defined at the beginning of a block is recognized only until the end of that block. If it is defined to be `static`, then it keeps its last assigned value for the next time the block is visited by the program flow. Otherwise, it is reinitialized each time. By placing a phrase of code inside a set of braces, we can define the indices `ii` and `jj` for the duration of the double loop. Alternatively, we could have defined `ii` and `jj` at the beginning of the `main()` block, or as `global variables` at the top of the program. Since we enjoy good editors and fast compilers, it is customary to modify code frequently and experimentally. As we shall see, these devices such the one we use here are useful for "hacking" an RTICA.[7]

The value of the expression `(Q)?(A):(B)` is the value of the expression `A` if the expression `Q` has a non-zero value (is 'true'), and it has the value of the expression `B` if `Q` has a zero value (is 'false').[8] Be careful to remember that in C, branches are taken when the opportunity occurs, and the expressions on the untaken branch is not evaluated. Thus if `Q` is false, and `A` contains an assignment, it will not executed even though it occurs to the left of `B`.

It is an illiView custom to use the ESCape key to exit the RTICA, or sometimes, the shifted ESCape key. For this reason, the `while()` loop should depend on the ESCape button even though there are other ways of terminating a running program.[9] Now, each time through the drawing loop the

[7]They are, however, hardly encouraged by computer scientific orthodoxy. Easily abused, they rapidly lead to undecipherable C-code.

[8]Coding minimalists may note that we would not really need all of this feature of C. The Boolean `(ii==jj)` already fucntions like the Kronecker delta.

[9]The succession ever more drastic ways of terminating a run-away program requires a separate discussion. This redundancy is necessary, because if your drawing loop hangs, the testing of the ESCape key at the end of the `while()` block will never be reached.

following events happen. First, a small deviation from the identity matrix in the Euclidean motion group is constructed. It is based on the deviation of the mouse from the nominal center of the screen at (640,512).[10] It is applied to the affine matrix, which, in turn, is applied to an orthographic projection matrix. The hidden frame buffer and the z-buffer is then cleared, the octahedron is drawn into it. Then the roles of the visible and hidden buffers are swapped. Now, let's see this in greater detail.

The identity matrix is loaded into the geometry pipeline, by `loadmatrix(id)`. This is a *destructive* command which replaces whatever matrix is on the geometry pipeline stack by its argument. The next two commands are *incremental*. The rotation about the y-axis followed by a rotation about the x-axis are multiplied into the matrix which is currently on the stack. Matrix multiplication being non-commutative, it is essential to agree on the order of multiplication. This, in turn, depends on whether we think of triples of numbers representing points in space as row or as column vectors. Historically, the graphics community prefers the *row-vector* convention, while mathematicians prefer the *column-vector* convention. To please both, Jim Clark, the founder of Silicon Graphics and the first author of its graphics library, **gl**, switched from column to row mode. But `OpenGL` has switched back to column mode. Since this convention influences only the documentation, not the implementation of the code, and both have their uses, we shall use both. But it is advisable not to mix them up.

There are no wholly satisfactory notational conventions for indicating the dependence of a matrix on a parameter. Rotations in 3-space are *about an axis*, and thus `gl` provides functions which multiply the current matrix on the stack by a rotation about the x,y or z axis. That is all we mean here. Later, we shall have to be much more precise about what we mean by a rotation.

In the column-mode, we write functional composition from right to left, with the column vectors of the object written last in the right-most place. In that case, we have modified the identity I by two small rotations, U^x, U^y, and applied it to the extant affine matrix A. Since `ortho()` is again a destructive matrix operation, `getmatrix(aff)` saves the new value of A. Thus, so far we have executed the pseudo-code line: $A \leftarrow IU^xU^yA$.

In case the size of the window has been enlarged with a mouse move, the `reshapeviewport()` command makes sure that the viewport matches the new window size.

The orthographic projection matrix P requires six arguments, corresponding to the range limits for `xmin, xmax, ymin, ymax` and `znear, zfar`. The first four are clear enough. They give the dimensions of the viewing window in world-coordinates. Thus, regardless of the shape and position on the screen you open the window, the left edge is at `xmin`, the right edge at `xmax`, the bottom edge at `ymin`, and the top edge at `ymax`. It is more com-

[10] Later, we shall want more sophisticated ways of using the deviation of the mouse from the center of the current drawing window.

plicated for the z-direction, and fortunately, it is largely irrelevant for the orthographic projection. For perspective projections, understanding these two last coordinates is important. We have written these coordinates as if they meant `zmin, zmax`, even though this is incorrect, see Exercise 1.

If X represents a vertex produced by `drawit()`, then the second half of the drawing loop just does PAX. The color black is specified by `cpack(0)`, and the frame buffered is cleared to black with `clear()`.

3 Exercises

Exercise 1.

Devise an experiment to see just what matrix the command `ortho(-1,1,-1,1,-1,1)` places on the projection stack. Consider using a `showmat()` function in the correct place, where:

```
showmat(){int ii,jj; Matrix mat;
   getmatrix(mat); printf("\n"); for(ii=0; ii<4; ii++)
   printf(" %f %f %f %f \n",mat[ii][0],mat[ii][1],mat[ii][2],mat[ii][3]};
}
```

Eventually, you should put the output of `shomat()` on the screen, and observe several matrices as they are used by an RTICAwhile you manipulate its inputs.

Exercise 2.

In `oc1.c` a small fraction of the displacement of the mouse from the center of the full screen steers the continuing rotation incrementally. You are solving a differential equation in a Lie group. In **oc2.c** implement a *trackball*-style rotor, for which the most recent displacement of the mouse determines an absolute rotation about an axis in the plane of the screen. One possible solution is given by this replacement of the `while()`-loop.

```
while(getbutton(ESCKEY)==0){
                static ox = 640, oy = 512;         /* screen center  */
                int dx = (getvaluator(MOUSEX)-ox); /* x-displacement */
                int dy = (getvaluator(MOUSEY)-oy); /* y-displacement */
                if( dx*dx + dy*dy ){               /* only if moved  */
                  ox += dx; oy += dy;
                  loadmatrix(id);
                  rotate(dx ,'y'); rotate(-dy,'x');
                  multmatrix(aff); getmatrix(aff);} /* end if*/
                resizeviewport();
```

```
                ortho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
                multmatrix(aff);
                cpack(0); clear(); zclear();
                drawit(); swapbuffers();}          /* end while*/
```

Exercise 3. Modify **oc1.c** to **oc0.c**, which demonstrates the t-mesh error made by the authors of the 1988 GL Programming Guide. Here is one possible way of doing this. Replace the `drawit()` function by `drawfaces()` which uses the incorrect t-mesh. It also draws only a portion of the octahedron, depending on the `NKEY` cycler. `drawedges()` does just that. The code segment uses some useful macros:

```
#define  IF(K)          if(getbutton(K))     /* illiGadgets */
#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};}

drawfaces(){
 static int nn=8; IF(NKEY){nn = ++nn % 9; SOAK(NKEY);}
           bgntmesh();
           cpack(0x0000ff);              v3f(vv[0]);
           cpack(0x00ff00);              v3f(vv[1]);
           cpack(0xff0000); swaptmesh(); v3f(vv[2]);
if(nn > 0){cpack(0xff00ff); swaptmesh(); v3f(vv[4]);}
if(nn > 1){cpack(0x00ffff); swaptmesh(); v3f(vv[5]);}
if(nn > 2){cpack(0x00ff00); swaptmesh(); v3f(vv[1]);}
if(nn > 3){cpack(0xffff00);              v3f(vv[3]);}
if(nn > 4){cpack(0x0000ff);              v3f(vv[2]);}
if(nn > 5){cpack(0xff00ff); swaptmesh(); v3f(vv[4]);}
if(nn > 6){cpack(0x00ffff); swaptmesh(); v3f(vv[5]);}
if(nn > 7){cpack(0x00ff00); swaptmesh(); v3f(vv[1]);}
           endtmesh(); }

drawedges(){ /* octahedra have Eulerian paths */
   static int ee[13] = {0,1,2,3,1,5,0,4,5,3,4,2,0}; int ii;
   cpack(0xffffff); linewidth(5);
   bgnline(); for(ii=0;ii<13;ii++)v3f(vv[ee[ii]]); endline();
}
```

To implement these new function write this segment into `main()` just before you `swapbuffers()`

```
  {static bf=0; TOGGLE(BKEY,bf); backface(bf);}
                IFCLICK(0,EKEY, drawedges();)
                IFCLICK(1,FKEY, drawfaces();)
```

Describe what the new gl-function `backface()`, and the macro `IFCLICK()` does by experimenting with **oc0.c**.