```
            /***********************************/
            /* tr1.c = Torus with illiLight     */
            /* RTICA by G. Francis, U. Illinois */
            /* This version (C)1996 August 31   */
            /***********************************/
#include <gl.h>
#include <device.h>
#include <math.h>
#define  MAX(u,v)      ( u<v ? v : u)
#define  MIN(u,v)      ( u<v ? u : v)
#define  CLAMP(x,u,v)  (x<u? u : (x>v ? v: x))
#define  FOR(i,a,b)    for(i=a;i<b;i++)
#define  DG            M_PI/180
#define  S(u)          fsin(u*DG)
#define  C(u)          fcos(u*DG)

float lu[3], vv[3],lux[]={0.3,0.5,0.8}, amb=.3, pwr=10., lmb ; int ii,jj;

int paint(float lmb, int cat, int dog){
     int rr,gg,bb; float spec;         /* illiPaint, Chris Hartman 1993    */
     rr = dog;                         /* map R^2(dog,cat) -> R^3(RGBspace */
     gg = 64 + abs(cat-128);
     bb = 255 - cat;                   /* illiLight by Ray Idaszak 1989    */
     lmb= MAX(lmb,amb);                /* clamp lmb to ambient below */
     spec = MIN(255, 255*(1.-pwr +pwr*lmb)); /* clamp spec below maximum */
     rr = MAX(lmb*rr,spec); gg = MAX(lmb*gg,spec); bb = MAX(lmb*bb,spec);
     return( (bb<<16) + (gg<<8) + rr);               /* hex for cpack */
}
hair(int th,int ta){ /* of the dog that bit you */
      float nn[3];   /* torus = circle of circles */
      nn[0] = C(th)*C(ta);  vv[0] = C(th) + .5*nn[0];
      nn[1] = S(th)*C(ta);  vv[1] = S(th) + .5*nn[1];
      nn[2] =       S(ta);  vv[2] =         .5*nn[2];
        lmb = nn[0]*lu[0] + nn[1]*lu[1] + nn[2]*lu[2]; /* Lambert cosine */
        lmb = CLAMP(lmb,0.,1.);
}
drawit() {int th, ta;
      for(th=10; th < 346; th += 15){
          bgntmesh();
          for(ta=5; ta< 346; ta +=15){
              hair(th,ta);
              cpack(paint(lmb,255*th/360,255*ta/360));
              v3f(vv);
              hair(th+15,ta);
              cpack(paint(lmb,255*(th+15)/360,255*ta/360));
              v3f(vv);
          }endtmesh();
       } /* end theta */
}
```

```
calculite(Matrix aff){ /* fix light source relative to rotation */
  FOR(ii,0,3){lu[ii]=0; FOR(jj,0,3)lu[ii]+= aff[ii][jj]*lux[jj];}
}
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 'a': amb    = atof(argv[1]);    argv++;argc--;break;
         case 'p': pwr    = 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 */

main(int argc, char **argv){ Matrix id, aff;
        FOR(ii,0,4)FOR(jj,0,4)id[ii][jj]=aff[ii][jj]=(ii==jj);
        arguments(argc,argv);
winopen("torus with illiLight ");
zbuffer(1); doublebuffer(); RGBmode(); gconfig();

while(!getbutton(ESCKEY)){
                int dx = (getvaluator(MOUSEX)-640)/4;
                int dy = (getvaluator(MOUSEY)-512)/4;
                loadmatrix(id);
                rotate(dx ,'y'); rotate(-dy,'x');
                multmatrix(aff); getmatrix(aff);
                calculite(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 loop */
} /* end it all */
```

illiLesson 2.

George Francis

September 24, 2009

1 Introduction

The RTICA **tr1.c** introduces a painted[1] surface, the torus, which is generated by a simple, trigonometric parametrization. Indeed, the normal to this surface is unusually simple and thus need not be computed with cross-products. The normal is need for every kind of lighting. Here we use a lighting model which is considerably simpler, more robust, and surprisingly versatile. Its motivation is discussed below.

2 illiTorus

You should think of the `main()` function of the program as the locomotive which pulls along a train of factor functions. In **oc1.c** there are two functions, the array of octahedral vertices `vv[6][3]`, and the t-mesh, `drawit()`, which drew the painted faces of the surface. In `tr1.c` we separate the drawing function `drawit()` into two *factors*[2] `paint()` and `hair()`. The latter generates *what* is to be drawn, the former says *how* it is to be colored and lighted. The `main()` function has two new factors, `arguments()`, which handles input

[1] To use color in a mathematically useful manner, it is necessary that the RTICA can assign a color to each computed point on the surface. One can do better than that with texture maps, which we introduce later on. We would not need texture maps for painting surfaces if we could draw a sufficiently fine mesh and still animate the surface interactively.

[2] Factoring functions has an aesthetic, conceptual and practical purpose. A phrase of computer code that does not fit into a space that can be read in its entirety is difficult to scan, either for content or for misprints. As a rule, a C-block should fit into a paragraph, or into a computer screen window. Poems are easier to understand in their entirety than a novel. Finally, smaller pieces of code are easier to edit and rearrange.

from the command line in an extensible manner,[3] and `calculite()`, which, together with `paint()`, subsitutes for the enormously complicated business of light and shadow in the standard `gl`-lighting model.

In `oc1.c` the `drawit()` function displayed all vertices and their colors in a single list. Since the torus is a surface which is parametrized by two angles, we can draw this surface with a pair of nested loops. The inner one draws a triangulated ribbon by generating a stream of vertex pairs. Their position, `vv[3]` is calculated by the function `hair()`[4] The same function uses the surface normal, `nn[3]`, to compute the Lambert cosine, `lmb`. This, in turn, is used by `paint()` to set the color for the vertex as the argument for the `cpack()`, and the vertex is sent to the t-mesh algorithm, `vf3(vv)`.

The Lambert cosine is that of the angle between the light-source and the unit-normal. This simplified Lambert-Phong lighting model includes an ambient fraction and a pseudo-specular bright spot.[5] To keep the bright-spot (more or less) stationary relative to the observer while the torus rotates, the light direction is recalculated by the function `calculite()`.

Here then are the new features:

- Generating a torus as an *ovalesque*, which is a successsion of circular[6] ribbons. A surprisingly large number of interesting surfaces are ovalesques.[7]
- Defining position and normal for points on a parametric surface.[8]
- A *paint function*[9] which interprets a two dimensional surface attribute

[3]Command line arguments are data you place after the name of the program, usually with an indicator in the form of a dash followed by a letter, followed by one or more numbers. While Hanrahan's code presents a instructive puzzle in C-syntax, it is easy to modify. You can remove or add a case, as we have done by commenting out code between `case` and `break;`, or writing in a new one.

[4]Which also computes the unit normal to the surface at each vertex, hence its silly name.

[5]The prefix 'pseudo' signifies that the spot is not quite at the correct location on the surface.

[6]More generally, ellipses.

[7]All of the rounded graphical primitives: cylinders, cones, spheres, ellipsoids, tori and knot-tubes. But also more artistic shapes, such as the *Etruscan Venus* are generated by a moving ellipse.

[8]For general parametric surfaces it is always a question whether it is more efficient to compute their normals analytically or approximating them by taking cross-products. For ovalesques, the radius of the circles, while not always a normal, is at least transverse to the surface, and can serve as a pseudo-normal for lighting purposes.

[9]Color wheel would be the more traditional term. But *color* already has too many different meanings. So we use the term *paint* for the pigment assigned to a vertex as distinct to the rgb-point in 3-dimensional color space.

as a 3-dimensional surface color.

- How to rotate an object under stationary illumination using a rudimentary rotor.

3 Ovalesques

Just as a *ruled surface* is one generated by a moving line, an *ovalesque* is a surface generated by an oval curve. A plain torus is swept out by a smaller circle centered and perpendicular to a larger circle[10]. The generating circle itself can be created by a *trigonometric interpolation* from the equatorial (horizontal) vector which rotates with theta, and a fixed vertical vector. The radius of this circle is $\frac{1}{2}$. Each such circle is displaced a distance 1 along the equatorial vector. This may be expressed thus

$$\begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} + \frac{1}{2} \left\{ \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} \cos(\tau) + \sin(\tau) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}.$$

Other surfaces are constructed differently, and we wish to separate the *definition* of a surface logically from its *drawing* on the Iris. Here, `drawit()` uses a nested loop, the θ parameter making not quite a circle in the outer loop, and the τ parameter making not quite a circle in the inner loop, which is t-meshed.

The t-mesh function on the Iris is so designed as to make it easy to code a triangulated ribbon. Note that this mesh makes no calls to `swaptmesh()`. To draw the ribbon think of a ladder. If you mention the vertices, beginning with the lower left, and proceed rung by rung, then the t-mesh requires no swapping[11].

Note that `drawit()` calls the `hair()` not only to supply the coordinates of the current vertex and its normal, `vv[], nn[]`, but also update `lmb`, the *Lambert Cosine* of the angle between the light and normal directions. We use a global variables here for convenience[12] and rely on the reader to remember

[10] What surface do you get if 'smaller' and 'larger' are reversed?.

[11] You might use the `backface()` function to rig a version of `tr1` to show how the orientation is affected by the order of mention. How many swaps do you need to draw the backside of the ribbon so that the triangles are the same? An important application of this secret is the t-meshing of ribbons that are painted differently on one side than on the other.

[12] Redesign this RTICA so that `hair()` returns pointers to these items. While you're at it, make `drawit(int(*hair)())` able to draw different surfaces under button control.

their names. Globals need to be justified because they are a source of error in very large programs, especially when more than one cook stirs the broth.[13]

To retrieve the argument for `cpack()` from `paint()` we send this function the Lambert cosine and the position of the parameter pair as fractions of the the rectangular parameter patch. More elaborate ovalesques will be discussed elsewhere. Actually, `drawit()` doesn't care what it is asked to draw, so long as it is parametrized by a rectangular patch.

4 Parametric Surfaces

The torus is, of course, doubly periodic. So it is natural to use trigonometric functions to define it. Also, the unit normal, `nn[]`, for the torus is particularly simple. It points to the position on the generating (meridian) circle after it has been translated to the origin and adjusted to unit radius. This shows that the *Gauss map* of the torus (which assigns to each point on the surface its unit normal) covers the sphere twice.[14]

Since the Gauss map is needed to estimate the lighting of an illuminated surface, and few surfacs other than planes, spheres, cones, cylinders and tori have simple Gauss maps, this item computationally expensive[15]. All things[16] being equal, the choice between the simplicity of approximating the normal by crossing two independent positional displacements[17] and deriving parametric expressions for it, is a trade-off between efficient coding and mathematical elegance.

5 Paint Functions

The C-phrase prefix `int` in the definition of

```
int paint(float lmb, int cat, int dog){...}
```

[13] Of course, when hacking a program, globals variables, along with static local variables, become invaluable.

[14] See pp 97, 103 of G. Francis, *A Topological Picturebook*, Springer Verlag, 1987, or any good geometry book for more on the Gauss map.

[15] Often more so than the position, since it involves partial derivatives and cross products.

[16] Like speed.

[17] Preferably ones that have to be computed anyway.

in **tr1.c** reminds you that it returns one integer.[18] Its inputs consist of the floating point Lambert cosine, and two integers[19] which one should think of as two color bytes. The scheme here is to let red follow the dog, which is correlated here to the θ parameter. Blue follows cat, but with a negative correlation;[20] and green peaks when the cat is close to the edge. The idea of this paint scheme is to let one axis linearly interpolate (*lirp*) two primary colors, while the second axis pushes this towards a linear interpolation between the complementary secondary colors. This way one can infer, at least qualitatively, the parameters preimage of a position on the surface.[21]

Theories for how to simulate the light on drawings of a surface, whether by hand or computer, all begin with Lambert's cosine.[22] A perfectly matte reflector radiates visible energy in all directions proportional to the amount received from the light-source. This in turn, is proportional to the cosine, $\lambda =$ `lmb`, of the angle between the normal and the light direction. `hair()` also clamps λ to 0 when the normal faces away from the light.[23] Thus the backside of a surface facing the light, and the front face of a surface facing away from the light would both be invisible.[24] So `paint()` next clamps λ above an agreed upon *ambient* fraction.

To simulate the bright spot produced by the reflection of a single, white light source off any colored surface, we locate the region when λ exceeds a certain threshold, and ramp all three colors steeply to the maximimum. This threshold is controlled by a parameter, called `pwr`.[25] The higher this number, the smaller is the diameter of the bright spot.[26]

[18] Actually, C-functions return integers by default, and other types by declaration. But they can only return one item, hence more elaborate outputs of functions, such as vectors and arrays, have to be transferred by their pointers. C-functions have other peculiarities the novice would never suspect. In these notes we shall experiment with various C-ish ways of working around these minefields but not always with much fanfare.

[19] We retain Chris Hartman's apt names, since the dog often chases after the cat, or the other way around.

[20] For didactic reasons, only.

[21] Another favorite paint scheme is to map a single scalar associated with a position on the surface to a rainbow color spectrum. This is considerably more difficult to program and we leave it as an exercise.

[22] See pp60-64 of the *Topological Picturebook* for the story of the geometrical theory of light and shade began with Lambert and Bouguer in the early 18th century.

[23] The cosine is negative.

[24] Generally, artists alleviate this by using partial *backlighting* by replacing a negative λ by a fraction of its absolute value.

[25] In conventional lighting models actual powers of fractinons are used. However, the power of a fraction is essentially zero until it rises steeply to 1. Only the steep ramp is of interest and we replace the power by a linear function.

[26] A surprising artifact of this model occurs when the power drops below 1. The continuous change to a monochrome lighting is not possible with conventional lighting models.

6 Stop the Sun

To simulate a stationary sun[27] shining on a rotating object, we consider our object stationary, and move the sun about it. We do this because the color of a vertex needs to be specified the moment its coordinates are started down the graphics pipeline. And these are in the coordinate system of the object. We rotate the object by multiplying its coordinates by a rotation matrix after the color has already been computed. Therefore, we compute where the sun should have been in the object's native world so as to appear stationary to the rotating object. Fortunately, the inverse of a rotation matrix in Euclidean geometry is its transpose. That is why `calculite()` is so simple.[28]

The advantage of our lighting model, *illiLight*,[29] conventional ones[30] is its simplicity and robustness. The object cannot simply dissolve into colorful nonsense or disappear altogther when its control parameters, `amb, pwr`, accidentally are allowed to go out of range[31]. Instead of polynomial functions of fractional numbers[32], illiLight uses maxima and minima to select the dominant lighting effect. It is not hard to add additional, monochromatic lights, and to localize them, though for such complicated features it is better to master[33] the Iris lighting model, which comes with library routines for all of this.

7 Command Line Arguments

Here is how `arguments(int arg c, char**argv)` works. The C-shell interpreter[34] arranges the information on the command line into an array `argv` of words

[27]We take the light source to be infinitely distant so that the light-direction does not have to be recalculated for each vertex.

[28]More generally, we shall need the inverse of a Euclidean motion, which combines a rotation with a translation. This is not much more complicated, but will be deferred until the next chapter.

[29]This lighting model was first implemented by Ray Idaszak in 1988, when illiView was started. It was improved over the years, notably by Glenn Chappell's `calculite()`, and Chris Hartman's (`paint()`.)

[30]Like the one provided by the Iris and described in the **Guide**.

[31]However, mysterious color artifacts develop if floating point values are passed to integers functions such as `cpack()`

[32]Lambert cosines, for example.

[33]We do not recommend it to people who are in a hurry or easily frustrated by inscrutable 'black boxes'.

[34]The program listening for you to press the **Enter**-key.

`argv[0], argv[1], ... argv[argc - 1]`, each being a character string.[35]

When `--` preceeds the variable name, it means that its value is decremented *before* it is used. Since `argc` is always greater than 0, the `while()` loop is never entered unless there is something else besides the name of the executable on the input line. If there is, we advance the word pointer, `argv`, so it points to the next word. That is why `argv[0][0]` now is the first character of the *second* word on the command line.[36] In effect, the action `++argv` drops a word off the front of the list. We next use the letter following the dash, namely `argv[0][1]`, to decide what to do next. Whatever it is, we pick up the subsequent words, which are numerals, of course. These numerals have to be converted to numbers, and for that there are the functions `atoi(), atof()`, which are abbreviations for "alphameric data converted to integer, resp. float values."

Each time we use a numeral, we have to eliminate it from the command line, by incrementing the word pointer, `argv++`, and decrementing the word count, `argc--`. The `break` takes us out of the cascade of cases.

As it appears now, the only items you can change from the command line are the ambient fraction, the power integer, and the coordinates of the light source.[37]

8 Modifications and Improvements

There are, of course, a great many more features and improvements that come to mind immediately, that should be added to `illiTorus` to make it much nicer and versatile RTICA . In fact, we do this in the next chapter. A more profitable outlet of your creative inclinations at this point would be to use the present structure of `oc1.c` and `tr1.c` to explore some other polyhedra, some other parametric surfaces, and perhaps to combine the two kinds of geometrical objects in the the same RTICA .

[35] Thus `argv[0]` is, in fact, the name of the program you are executing.

[36] And it has to be a dash or we quit.

[37] The command line is the oldest interactive means of influencing a C-program. The mouse, keyboard, and latlely *widgets*, like pull-down menues, sliders etc., have largely supplanted the command line for data input. One good use remaining is to set a few switches for alternate operations of your program.