Projects for Mathematics Students

Curves and surfaces

Prerequisites:

Programming skills in C, ability to use a standard function library, and mathematics skills to understand simple functions and their graphs

Graphics to be learned from these projects:

Defining an image using viewing projections and an eye point; use of graphics primitives and transformations for modeling; rendering features of a graphics API such as lighting and transparency; various callbacks and interactions that control changes in both modeling and rendering; the use of capabilities such as clipping and alpha blending in rendering images.

The projects we have developed for mathematics students in the first computer graphics course focus around curves and surfaces based on various formulas. These give students a chance to become more familiar with some standard kinds of shapes and to build their intuition about how functions work.

One of the interesting opportunities for mathematical explorations is to consider surfaces that have singularities or discontinuities in the domain being examined. The display of these functions will not itself have any of these, but it should be possible to find places where they would occur.

Surfaces may be graphs of a real-valued function of two variables, or they may arise from other processes that work on a two-dimensional domain. Real functions of two variables are common in multivariate calculus and in actual applications; you will find examples in the set of physics projects, for example. Parametric surfaces are created by functions from real two-space into real three-space. They can define surfaces with more complex behaviors, such as multiple sheets, as shown in Figure 3, as well as surfaces such as the torus. This is the source of another kind of graphics project in which the student is asked to display a surface defined by parametric functions of two variables: one for each of the three coordinates in the 3-space that will contain the surface. It may even be possible for some classes to examine higher-dimensional surfaces through functions from a two-dimensional domain into higher-dimensional space, with the displayed surface being projected down into three-space for viewing.

Surfaces are plotted as illustrated in Figure 1 by creating a grid on a rectangular domain in two-space and applying a function or functions to the points in the grid to determine points in three-space. This figure illustrates the fundamental principal for surfaces that are given by a function of two variables; it shows a coarse grid on the domain (a 6x6 grid instead of 125x125) for the actual surface in Figure 2 so you may see the underlying grid and the relationship of the function's value to the surface more clearly. Those points are used to determine rectangles in three-space that can be displayed with standard OpenGL functions. The gridded surfaces we create in this way are only approximations of the real surfaces, of course, and it will be instructive to the students to consider what happens when they try to work with functions whose surfaces includes discontinuities of various kinds.

Parametric surfaces are plotted in much this same way, but they are slightly more difficult for the student to grasp because they do not have the close relationship between the domain grid and the surface components shown in the figure. Some motivating examples may help students understand how the function surface process extends to parametric surfaces.

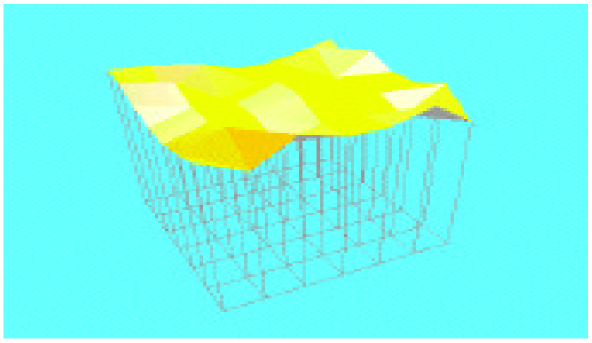


Figure 1: mapping a domain rectangle to a surface rectangle

Using these surface and curve examples as course projects

We can use these examples to give students in the graphics course an opportunity to use the functionality of OpenGL while examining mathematical questions.

- The first project could be simply to display a surface defined by a function. This would require proper initialization of the OpenGL system, definition of the viewing environment for the visualization, use of geometric primitives to display the triangles or quads derived from the domain grid, use of hidden-surface display, and use of color to display the surface.
- The second project could add shading and lights to the visualization, illustrating ambient, diffuse, and specular lighting and showing the surface with appropriate highlights. This project could also focus on parametric surfaces, allowing the student to generalize his or her experience with a function surface to a parametric surface.
- The third project could add keyboard-controlled rotations, introducing event-driven programming with callbacks and allowing the instructor to discuss the way the user experiences controls. These surfaces can be complex, so you may want to include clipping planes in this project to allow the student to slice through the surface and see the internal details, though this is not shown in the example because it is covered in the conic section example. Because of the time it might take to display a surface with high resolution, the third project should also add display lists to improve performance on the display.
- The fourth project could switch to the conic section display, because this adds a number of opportunities for menus and for clipping planes as well as re-using the keyboard rotation control from the previous project. These displays are not as complex as the surface displays and so can probably be seen effectively in smaller windows, so it might be interesting to have students create stereo pairs to see the conic sections in 3D.
- The fifth project could change direction to display spline surfaces with user-defined, user-selectable, and keyboard-controlled movable control points. Such projects would require students to create a set of control points that makes reasonable sense, to create selection buffers and name stacks, and to manage hit records. An example of such a project is included in these materials.

These projects do not cover all the areas of the course, however. They do not include alpha blending or textures, for example; the instructor might want to add these to other project (but be aware that adding alpha blending may not give the results you want because its results depend strongly on the sequence in which elements are displayed. You probably need to do other work such as Z-sorting polygons in order to get transparency to look right.)

Function surfaces with optional animation

If we consider a function of two variables, `z=f(x,y)`, acting on a contiguous region of two-space, then the set of points `(x,y,f(x,y))` forms a surface in three-space. This project explores such surfaces through processes that are described briefly in Figure 1 above.

The goal of the project is to allow a student to see the fundamental principles of surfaces by creating a rectangular domain in X-Y space, evaluating the function at a regular grid of points in that domain, and creating and displaying small rectangles on the surface that correspond to a small rectangle in the underlying grid. This is not quite so simple, however, because the rectangle in the surface may not be planar. We can solve that problem by dividing the surface rectangle into two triangles, each of which is planar.

The project, as it is defined first to the student, is to create a single view of a surface. The challenges are to create the view environment and to understand what is meant by the surface triangles and rectangles, and how they are generated. We suggest two-sided surfaces with different colors above and below, so the student can clearly see how the view goes from one side to the other when the surface is rotated. This will also allow the student to distinguish the two sides when some other phenomena, such as surfaces that show their underside at the edge of the domain. Figure 2 below shows such a surface with a yellow color and with three lights — red, green, and blue, evenly spaced around the space — that shows how this can look. Once they have done that, which makes a good first project, a later project can add the ability to rotate the surface in space. This can use keyboard rotation controls, as shown in the example code, and is a good introduction to event-driven programming with callbacks.

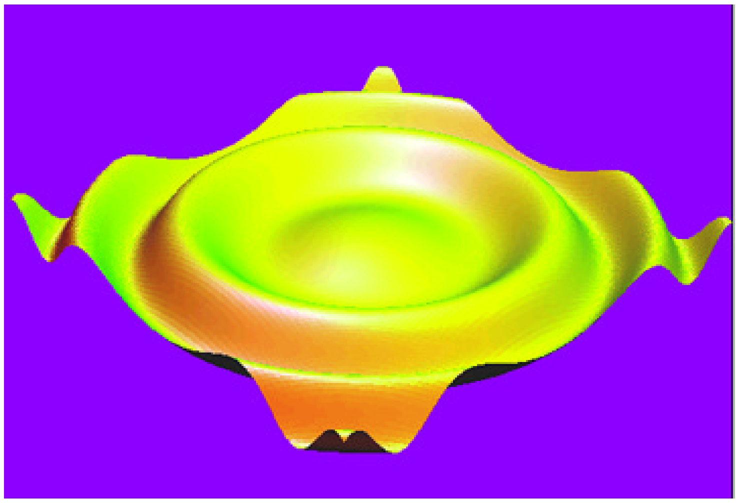


Figure 2: an example of a function surface display

In order to have a good set of functions to work with, we suggest that the instructor give the students a small number of functions whose shape is well understood. However, there are a number of sources of functions. We encourage you to have your students look at their courses, including courses in physics or chemistry, or in references such as the CRC tables of mathematical functions for curious and interesting formulas that they do not immediately understand, with a goal of having their images help them with this understanding.

To avoid having to change the code and recompile when the student wants to look at a different function, we suggest having the student create a menu of surfaces and defining the project to include menu selection for the function. Some interesting functions to consider for this are:

1. z = c * (x^3 - 3*x*y^2)
2. z = c * (x^4/a^4 - y^4/b^4)
3. z = (a*x^2 + b*y^2)/(x^2+y^2)
4. z = c*cos(2*π*a*x*y)
5. z = c*cos(2*π*a*x)*(2*π*a*y)
6. z = c*log(a*x^2 + b*y^2)

In this project it is fairly easy to pose some questions for students about the meaning of what they see, particularly if the instructor has chosen a good set of functions that have various kinds of discontinuities within a usual domain. Note that function 3 includes an essential singularity at the origin, so students will be faced with having to interpret this surface's inaccuracies.

Later development of this project, used either to introduce animation or after some animation has been covered, can consider not one single surface but a one-parameter family of surfaces. The parameter's values can be stepped along by the idle callback function to allow the student to work with a more complex set of surfaces and in particular to see how the value of the function parameter affects the shape of the surface. This animation can be combined with rotation and even clipping controls so that the student can move the surface around as it is animating, though desktop systems do not seem to have enough speed to do this easily. Animations such as this are very motivating to students, so you are encouraged to include them whenever possible. (Unfortunately, we cannot include an example of an animation in this note, but the sample source code, `animSurf.c`, does provide a surface animation that you can compile and demonstrate to your students.)

This version of the project not only allows students to use menu and keyboard controls for function selection and world rotation; it also allows them to see the difference between the speeds of different kinds of systems and graphics cards. This may or may not be a good thing, depending on how good your labs are and how competitive your students are! However, if it fits the overall goals of your program, then some speed comparisons can be a good thing, and this can be expanded to compare how quickly different students' programs execute.

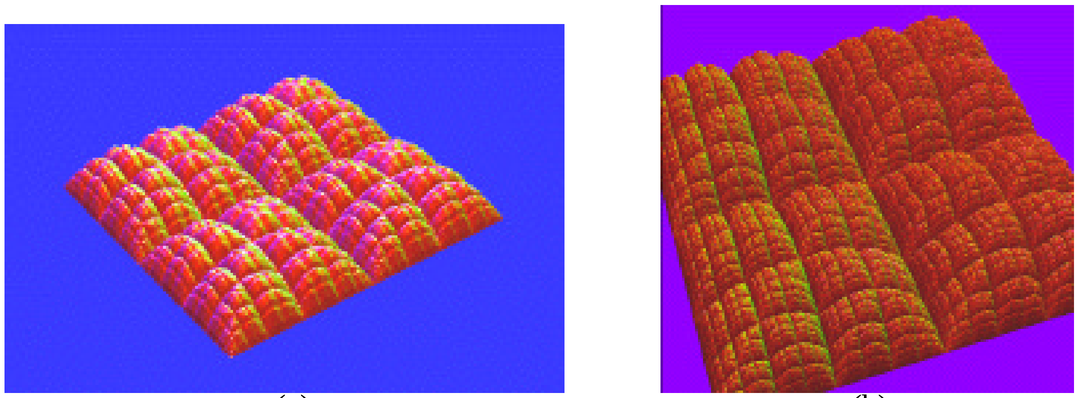
Surfaces for special functions

Functions that are given by formulas in closed form are almost always analytic at almost all points in the domain. Special cases such as zero denominators or undefined transcendental functions usually disturb that analytic nature only at discrete points. However, there are other kinds of functions that exhibit more unusual behavior. One such kind of function is everywhere continuous but nowhere differentiable. Computer graphics can allow students to experiment with such functions and get a better understanding of their behavior.

As an example of a function of a single variable that is everywhere continuous but nowhere differentiable, consider the Weierstrass function

$$f(x) = \Sigma_i \sin(x*2^i)/2^i$$

where the sum is over all positive integers i. This can easily be extended to a function of two variables with the same property, and the surface would be useful for the student to see. For computational purposes, however, it is probably better (for speed purposes) to use an algebraic function instead of the transcendental sine function, so we have developed an example that uses a "zig-zag" function. This kind of example has been called a blancmange function (after a traditional British holiday pudding whose surface is extremely wrinkled) and the surface for this example is shown in Figure 3 below, both at modest and high resolutions.



(a) (b)

Figure 3(a): the blancmange surface, (b) zoomed in with more iterations

Parametric surfaces

In the function surface projects above, the student was asked to start with a grid in the X-Y domain and compute a value of Z for each point in the grid. The resulting points (x,y,z) were then used to compute rectangles (or triangles) in real three-space that are graphed.

The grid need not be so directly involved in the surface, however. In the parametric surface projects, the students will start with a grid in parameter space (which we will call U-V space). For each point (u,v) in the grid, three functions will give three real values for each of these points; these values are the x-, y-, and z-coordinates of the points that are to be graphed. So the difference between function surfaces and parametric surfaces is relatively small from the programming point of view.

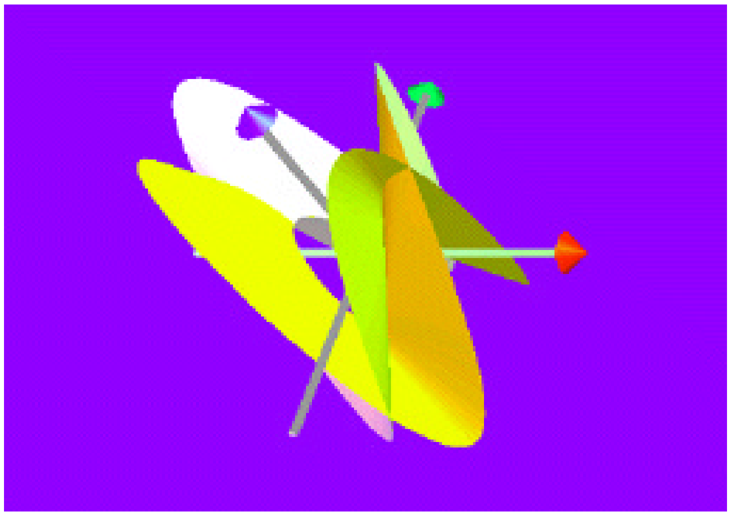


Figure 4: a parametric surface

From the point of view of the surfaces themselves and they way they look when displayed, however, the differences between these processes can be immense. Function surfaces are always single-sheet: they always look like a horizontal sheet that has been distorted upwards and downwards, but never wraps around on itself. Parametric surfaces can be much more complex. For example, a sphere can be seen as a parametric surface whose coordinates are computed from latitude and longitude, a two-dimensional region; a torus can be seen as a parametric surface whose coordinates are computed from the angular displacement around the torus and the angular displacement in the torus ring, another pair of dimensions. Figure 4 shows an example of a parametric surface from the Banchoff paper that consists of two intersecting sheets. This figure has the same property of two colors on its two sides and three colored lights so the surfaces are distinguishable. It also displays coordinate axes to help the observer see the space. The code for this surface display is given in the file `paramSurf.c` in the set of code examples. That code includes a few options for the parametric functions themselves, so you can easily try variations of the display.

There are many surfaces that can be created using parametric definitions. It is easy to define a torus in this way, for example; the functions are

x(u,v)=(A+B*cos(v))*cos(u);
y(u,v)=(A+B*cos(v))*sin(u); and
z(u,v)=B*sin(v) for real constants A and B

for values of u and v between 0 and 2*π, where A is the radius of the torus body and B is the radius of the torus ring. Other examples may be considered as well. A couple of these that you may wish to experiment with are:

Helicoid:

x(u,v)=a*u*cos(v),
y(u,v)=b*u*sin(v);
z(u,v)=c*v for real constants a, b, c

Conical spiral:

x(u,v)=a*[1-(v/2*π)]*cos(n*v)*[1+cos(u)]+c*cos(n*v);

y(u,v)=a*[1-(v/2*π)]*sin(n*v)*[1+cos(u)]+c*sin(n*v);

z(u,v)=(b*v/2*π)+a*[1-v/(2*π)]*sin*u) for real constants a, b, c and integer n

This area of surface exploration offers some very interesting opportunities for advanced studies in functions. For example, the parametric surface may not exist in 3-space but in higher-dimensional space, and then may be projected down into 3-space to be seen. For example, there may be four or more functions of u and v representing four or more dimensions. Familiar surfaces such as Klein bottles may be viewed this way, and it can be interesting to look at options in projections to 3-space as another way for the user to control the view.

Finally, there are surfaces that are composed piecewise of a number of patches, each of which is a parametric surface. These include spline surfaces, which are supported by OpenGL directly without the detailed use of basis functions or spline matrices that one would normally see in mathematical treatments of splines. It is thus possible to examine the possibilities in spline surfaces graphically and encourage the student to look forward to the detailed mathematical treatment that would be available in later mathematics courses.

Functional and parametric curves

In a similar but simpler direction, one can display curves by a function from real one-space into real three-space just as we did function surfaces, or one can display curves that are defined parametrically just as we did parametric surfaces. The results are similar; function curves look like standard graphs, while parametric curves can have loops or other complex behavior. It is also possible to have curves defined by other processes, such as differential equations. This area is not pursued in this writeup, but single-parameter curves could be a good set of examples to introduce a class to the general OpenGL environment, to the notion of viewing the behavior of functions, and to learning to see spatial information.

A simple example of curves defined by functions is given by a standard helix:

x=a*sin(t), y=a*cos(t), z=t.

Other curves can be more interesting and complex. Some can be derived by taking the parametric surfaces described above and making one of the u or v variables a constant; we will not write any of these explicitly. Others may come from different sources. A couple of interesting examples are the rotating sine wave:

x=sin(a*t)*cos(b*t), y=sin(a*t)*sin(b*t), z=c*t/(2*π)

or the toroidal spiral:

x=(a*sin(c*t)+b)*cos(t), y=(a*sin(c*t)+b)*sin(t), z=a*cos(c*t).

This latter example is shown in Figure 5 below. Curves such as these are sufficiently complex that students should feel some satisfaction in seeing the results of their programs.

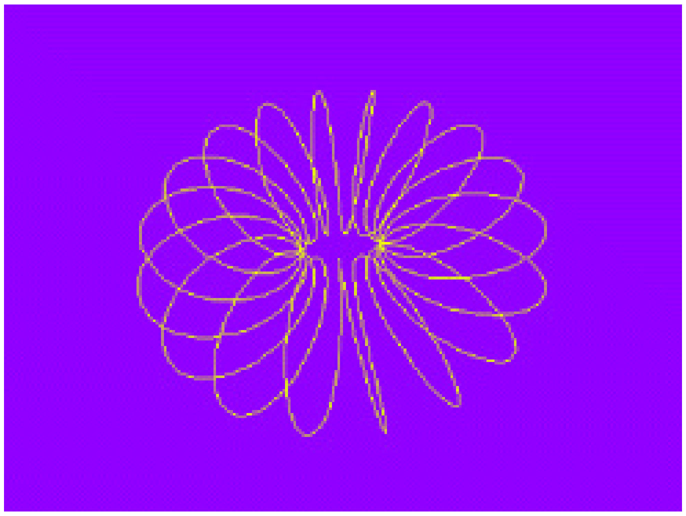


Figure 5: The toroidal spiral curve

Not all curves are given by equations, however. An excellent example of curves described by differential equations that describe a complex phenomenon having no closed-form solution is given by the curve defined by the Lorenz equations, shown in Figure 6.

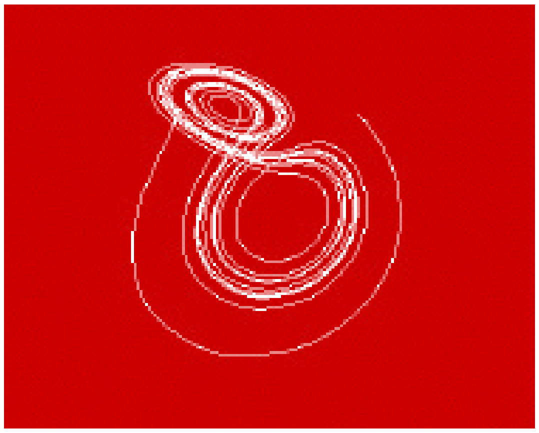


Figure 6: an example of the Lorenz curve

These differential equations:

dx/dt = s * (y - x)
dy/dt = r * x - y - x * z
dz/dt = x * y - b * z

for constants s, r, and b are often described in discussions of strange attractors. Under certain circumstances (in particular, s=10, r=28, and b=8/3, used in the example in the figure) chaotic behavior occurs, and the set of curves given by the differential equations (with parameter t) are very interesting, but to do it well you need high-quality numerical integration. This may require tools you do not have in your programming environment, and the example that is included here only uses difference equation approximations.

Conic sections

One of the common things students hear in algebra and calculus is that the graphs of quadratic expressions are all curves produced by conic sections, by intersections of a plane with a cone. This project allows students to generate the various conic sections by drawing a cone and clipping that cone with a plane, and to observe the shapes of the resulting curves. The sample code illustrates various common features of OpenGL — modeling and viewing, lighting, clipping, and keyboard and menu callbacks. The graphics programming is not in itself difficult, but when the callbacks and interface issues are added it should be a good program for *interactive* graphics. Figure 7 below shows two screen captures from the sample code that illustrate what the project can do, and the instructor is encouraged to experiment with the code and the project to try out the interaction.

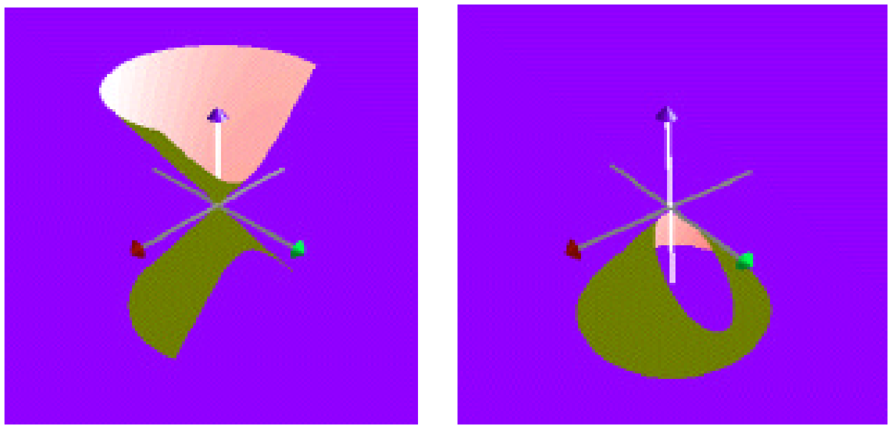


Figure 7: examples of two conic section displays (left: hyperbola; right: ellipse)

This project is a good place to have students work with rotations and menus, because the different kinds of conic sections are defined by the way the clipping plane strikes the cone, and a menu can be used to make that selection. Rotations allow the student to see how the section looks from many different angles. The example code also includes keyboard control that moves the clipping plane through three-space while maintaining its orientation, letting the student experiment with different positions of the clipping plane. With all these opportunities for student control of the image, this is a particularly good project for getting students to think about what interface options work for this kind of geometric process. The code for this example is in the file `conics.c` in the set of sample codes.

Adding 3D viewing:

If you create a window that is twice as wide as it is high, and if you divide it into left and right viewports, you can display two images in the window simultaneously. If these two images are created with the same model and same center of view, but with two eye points that simulate the location of two eyes, then the images simulate those seen by a person’s two eyes. Finally, if the window is relatively small and the distance between the centers of the two viewports is reasonably close to the distance between a person’s eyes, then the viewer can probably resolve the two images into a single image and see a genuine 3D view.

Such a view is shown in Figure 8: a pair of views of conic sections. None of these processes are difficult, so it would add some extra interest to at least one project to include 3D viewing in the project.

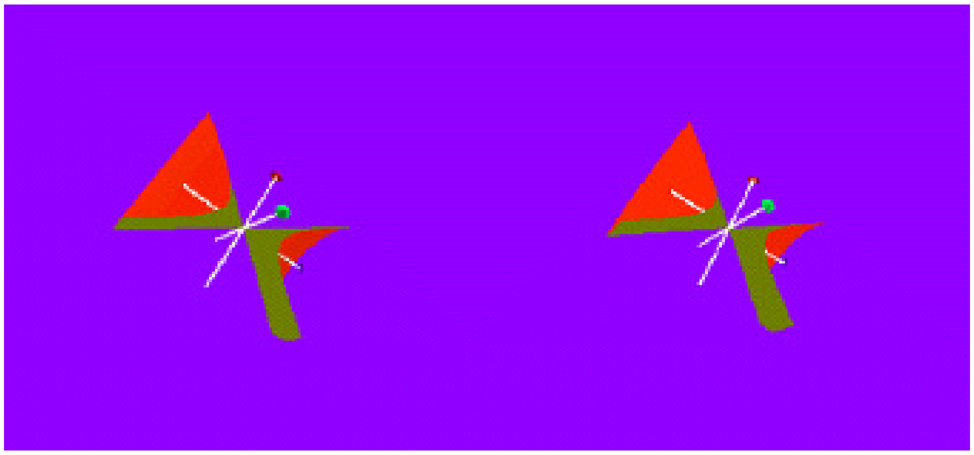


Figure 8: A stereo pair that the viewer should be able to resolve.

Another approach to 3D viewing is available using Chromadepth™(?) glasses. These have diffraction gratings in the lenses that change the angle of refraction of light coming into each lense depending on the wavelength of the light. Each pair of glasses has two lenses that are oriented oppositely, so that the refraction makes objects seem to have less of an angle between their images through the glasses than is actually the case. Longer wavelengths are refracted less than shorter wavelengths, so things that are colored red seem to move back into the image less than things that are blue. The effect is that red things seem closer than blue, and that effect can be used to apply differential color to images in order to provide depth cuing.

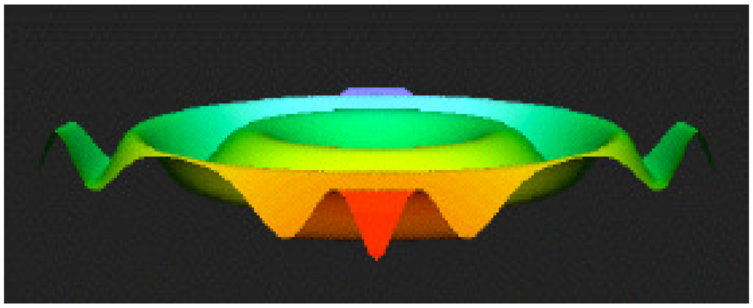


Figure 9: A surface with Chromadepth™ color encoding

In OpenGL, this differential color can be applied by using a one-dimensional texture map with a texture that is a ramp between red and blue, and applying that texture in a texture-modulation mode so the effect of lighting on the surface are visible. Values from the depth buffer in an image are used to select the color from the texture map. The result is an image like that of Figure 9, a Chromadepth image of the animated surface shown in Figure 2 created by the code in the `mathChroma.c` example. The disadvantage of this approach is that an image cannot use color to encode information; the advantage is that it makes depth viewing much simpler. Chromadepth glasses are inexpensive, and their source is noted in the references.

Credits:

This work was supported by National Science Foundation grant DUE-9950121. All opinions, findings, conclusions, and recommendations in this work are those of the author and do not necessarily reflect the views of the National Science Foundation.

The author would like to acknowledge the support of the San Diego Supercomputer Center and of the assistance of a number of colleagues, including xxxx, in reviewing and helping to improve these materials.

References:

- Zimmermann, Walter and Steve Cunningham, *Visualization in Teaching and Learning Mathematics*, MAA Notes Number 19, Mathematical Association of America, 1991
- Banchoff, Tom et al., “Student-Generated Software for Differential Geometry,” in Zimmermann and Cunningham, above, pp. 165-171
- von Seggern, David, *CRC Standard Curves and Surfaces*, CRC Press, 1993
- Textbook for multivariate calculus ...
- Textbook for differential geometry? ...
- Chromadepth glasses and other information are available from

  Chromatek Inc
  1246 Old Alpharetta Road
  Alpharetta, GA 30005
  888-669-8233
  http://www.chromatek.com/