Appendix C

Computer Scripts

Those readers who have access to computer systems running Maple©, Mathematica©, Derive©, or similar software can use these systems to facilitate gaining geometric intuition and imagination of the concepts of differential geometry. However, the current state-of-the-art technology for generally available computer graphing programs is not capable of producing what would be the most useful displays. For example, it is not currently possible, with widely useable programs, to view a curve on a surface and to use the mouse to dynamically move a point along the curve and see displayed the three curvature vectors—intrinsic (geodesic), extrinsic, and normal.

In this appendix we have included several computer exercises for Maple, and these and additional scripts are also available for downloading on-line at

`ftp://math.cornell.edu/pub/Henderson/diff_geom`.

The first section contains a file with the definitions of several functions used by the other scripts. The other scripts are labeled according to the problem in the text to which they are most connected.

Standard Functions

These are definitions of commonly used functions that can be called by other scripts, and referred in the other scripts as the file with name `diffgeo_defs`. If you save it as a file with a different name, then you will have to change the appropriate lines in the scripts.

```
with(plots):
with(linalg):
macro(medgreen = COLOR(RGB,0,0.8,0.5));
macro(medgrey = COLOR(RGB,0.6,0.6,0.6));
vecnorm:= (vec) -> sqrt(dotprod(vec,vec)):
vecnormalize:= (vec)->evalm((1/vecnorm(vec))*vec):
Tangent := proc(t,func)
   local Velocity, VelEval, VelLength;
      Velocity := map(diff, func(x), x);
      VelEval := map(evalf,(subs(x=t, Velocity)));
      VelLength := sqrt(dotprod(VelEval,VelEval));
   evalm((1/VelLength)*VelEval)
end:
Tanvec := proc(t,func)
   polygonplot3d([func(t),convert(evalm(func(t)+
      Tangent(t,func)), list)],color=medgreen)
end:
CNormal := proc(t,func)
   local TanPrime, TanPrimeEval,
      TanPrimeLength;
      TanPrime:= map(diff,Tangent(x,func),x);
      TanPrimeEval:=
         map(evalf,subs(x=t,eval(TanPrime)));
      TanPrimeLength:= sqrt(dotprod(TanPrimeEval,
      TanPrimeEval));
   evalm((1/TanPrimeLength)*TanPrimeEval)
end:
Normvec := proc(t,func)
   polygonplot3d([func(t),convert(evalm(func(t)+
      CNormal(t,func)),list)],color=red)
end:
Binormal := (t,func) ->
   crossprod(Tangent(t,func),CNormal(t,func)):
BiVec := proc(t,func)
   polygonplot3d([func(t),convert(evalm(func(t)+
      Binormal(t,func)), list)],color=blue)
end:
Velocity := proc(t, func)
   local fprime;
      fprime := map(diff, func(x), x);
   map(evalf, subs(x=t, fprime))
end:
sprime := proc(t, func)
   sqrt(dotprod(Velocity(t,func),Velocity(t,func)))
end:
VCurvature := proc(t, func)
   local TanPrime, TanPrimeEval;
      TanPrime := map(diff,Tangent(x,func),x);
      TanPrimeEval := map(evalf,subs(x=t,
         eval(TanPrime)));
   evalm((1/sprime(t,func))*TanPrimeEval)
end:
scurvature := proc(t,func)
   sqrt(dotprod(VCurvature(t,func),
         VCurvature(t,func)))
end:
CurvVec := proc(t,func)
   polygonplot3d([func(t),convert(evalm(func(t)+
      VCurvature(t,func)), list)], color=magenta)
end:
CurvVec2 := proc(t,func)
   polygonplot3d([func(t),convert(evalm(func(t)+
      VCurvature(t,func)), list)],
      color=magenta, thickness=2)
end:
GaussMap:= proc(func)
   local func1, func2;
      func1 := map(diff, func(x1,x2), x1);
      func2 := map(diff, func(x1,x2), x2);
```

```
    vecnormalize(crossprod(func1,func2))
  end:
  GMapEval := proc(func,u,v)
    evalf(subs(x1=u,x2=v, GaussMap(func)))
  end:
  FirstFundFormDet := proc(func)
    local func1, func2, g11, g12, g22;
      func1 := map(diff, func(x1,x2), x1);
      func2 := map(diff, func(x1,x2), x2);
      g11 := dotprod(func1,func1);
      g12 := dotprod(func1,func2);
      g22 := dotprod(func2,func2);
    g11*g22 - g12^2
  end:
  SurfaceNorm := proc(func)
    local func1, func2;
      func1 := map(diff, func(x1,x2), x1);
      func2 := map(diff, func(x1,x2), x2);
    vecnormalize(crossprod(func1,func2))
  end:
  SecondFundFormDet := proc(func)
    local func11, func12, func22, L11, L12,L22;
      func11 := map(diff, func(x1,x2), x1, x1);
      func12 := map(diff, func(x1,x2), x1, x2);
      func22 := map(diff, func(x1,x2), x2, x2);
      L11 := dotprod(func11, SurfaceNorm(func,
        x1, x2), orthogonal);
      L12 := dotprod(func12, SurfaceNorm(func,
        x1, x2), orthogonal);
      L22 := dotprod(func22, SurfaceNorm(func,
        x1, x2), orthogonal);
    L11*L22 - L12^2
  end:
  GaussCurv := proc(func,u,v)
    evalf(subs(x1=u, x2=v, SecondFundFormDet(func)/
      FirstFundFormDet(func)))
  end:
```

Computer Exercise 1.6: Strake

In this exercise, you can vary the inner radius of a strake.

```
> with(plots):
> setoptions3d(scaling=constrained);
```

First, input a radius between 0 and 2:

```
> r := 1;
```

Create a cylinder with that radius:

```
> cylinder := plot3d([r*cos(u), r*sin(u), v],
      u=0..2*Pi, v=0..2, color=green):
```

Create a strake with inner radius `r` and outer radius `2`:

```
> strake := plot3d([((1-t)*r+ 2*t)*cos(u),
       ((1-t)*r +2*t)*sin(u), u/Pi], u=0..2*Pi,
       t=0..1, color=blue):
```

Display them together:

```
> display({strake, cylinder});
```

We suggest viewing the surfaces in the style "`patch w/o grid`" with one of the lighting schemes. Try various values for `r`, including `r=0` and `r=2`. When you are finished with this exercise, hit the enter key after the following line:

```
> r := 'r';
```

This will make `r` a variable again. If you don't do this, the computer will continue to think that `r=1` (or whatever your last radius value was).

Computer Exercise 1.7: Surfaces of Revolution

In this exercise, we will take a plane curve and turn it into a surface of revolution.

```
> with(plots):
> setoptions3d(scaling=constrained);
```

First, the plane curve:

```
> f:= (x) -> sin(x) + 2;
```

Now we create the surface of revolution, defined by (`u`,`v`) goes to (`u`, `f`(`u`)cos(`v`), `f`(`u`)sin(`v`)):

```
> plot3d([u, f(u)*cos(v), f(u)*sin(v)], u=-Pi..Pi,
       v=0..2*Pi, axes=normal, orientation=[90,90],
       color=blue);
```

Notice that when the surface first appears, we are looking down the y-axis. The upper boundary of the surface is `f(x)` and the lower boundary is `-f(x)`. Try rotating the surface to see how it looks from various angles. Experiment with other plane curves by changing the definition of `f` on your worksheet. Note that you may want to vary the range of `u` in the "`plot3d`" line. Can you make a cylinder? a cone? a sphere? a paraboloid? Keep these surfaces in mind as you do Problem 1.7.

Computer Exercise 1.9: Surface as Graph of a Function

This exercise allows you to test out your answers to Problem 1.9.

```
> with(plots):
> setoptions3d(scaling=constrained):
```

First, input a function of `x` and `y` (the default surface is a monkey saddle):

```
> f:= (x,y) -> (1/2)*x*(x^2 - 3*y^2);
```

Then display it:

```
> plot3d(f(x,y),x=-1..1,y=-1..1,color=blue);
```

We suggest using the style "`patch`" and one of the lighting schemes. Remember that when you change your surface equation, you may want to vary the bounds on x and y in the "`plot3d`" statement.

Computer Exercise 2.2: Tangent Vectors to Curves

In this exercise, you can input a space curve and see the **tangent vectors** at various points.

```
> with(plots):
> with(linalg):
> setoptions3d(scaling=constrained);
> read(`diffgeo_defs`);
```

First, the space curve:

```
> f:= (x) -> [cos(x), sin(x), x/7];
```

Now we define the domain (from `a` to `b`) and the number of divisions of the domain (`div`). Eventually, we will find the tangent vector at the endpoints (`points`) of each division.

```
> a:=0;
> b:=4*Pi;
> div:=10;
> points := {seq( a + i*(b-a)/div, i=0..div)}:
```

Now we get the tangent vector by differentiating `f` to get the velocity and then normalizing.

```
> Tangent(t,f);
```

Here we create the vectors that will appear on the screen.

```
> Vectors := map(Tanvec, points, f):
```

Here we calculate the coordinates we need to put the curve on the screen.

```
> Curve := spacecurve(f(x),x=a..b,thickness=2,
      shading=zgrayscale):
```

And finally, we display the curve and its tangent vectors.

```
> display({Curve} union Vectors);
```

Now try other curves by changing `f` on your worksheet. Remember you may also want to change your domain and/or the number of divisions.

Computer Exercise 2.3: Curvature and Tangent Vectors

In this exercise, you can input a curve and see the **curvature and tangent vectors** at various points.

```
> with(plots):
> with(linalg):
> read(`diffgeo_defs`):
> setoptions3d(scaling=constrained):
```

First, the space curve:

```
> f := (x) -> [cos(x), sin(x), x/7];
```

Now we define the domain (from `a` to `b`) and the number of divisions of the domain (`div`). Eventually, we will find the tangent vector at the endpoints (`points`) of each division.

```
> a:=0;
> b:=4*Pi;
> div:=10;
> points := {seq(a + i*(b-a)/div, i=0..div)}:
```

Here we create the vectors that will appear on the screen.

```
> TVectors := map(Tanvec, points, f):
```

```
> CVectors := map(CurvVec, points,f):
```

Here we calculate the coordinates we need to put the curve on the screen.

```
> Curve := spacecurve(f(x),x=a..b,thickness=2,
      shading=zgrayscale):
```

And finally, we display the curve and its tangent vectors.

```
> display({Curve} union TVectors union CVectors);
```

The default curve is a helix. Experiment with different helices (try different values of `c` and `d` in the line

```
> f := (x) -> [cos(x), sin(x), x/7];
```

and see how the curvature vector changes). Try the logarithmic spiral as well–vary its domain to see different positions of the curve. What do you observe?

Computer Exercise 2.4a: Osculating Planes

In this exercise, you can input a space curve and see the **osculating plane**, tangent vector, and normal vector at various points.

```
> with(plots):
> with(linalg):
> setoptions3d( scaling=constrained );
> read(`diffgeo_defs`):
```

The default curve is a helix. First, input the curve:

```
> f:= (x) -> [2*cos(x), 2*sin(x), x/5];
```

Now define the curve domain (from `a` to `b`) and the number of divisions in the domain (`div`). Eventually, we will find the osculating plane at the endpoints of each division.

```
> a:=0;
> b:=4*Pi;
> div:=5;
> points := {seq( a + i*(b-a)/div, i=0..div)}:
```

Here we calculate the coordinates we need to put the curve on the screen.

```
> Curve := spacecurve(f(x), x=a..b, thickness=2,
      shading=zgrayscale):
```

Now we define the osculating plane, which is spanned by the tangent vector and the normal vector.

```
> OscPlane := proc(t)
   plot3d(evalm(x*Tangent(t,f)+ y*CNormal(t,f)+ f(t)), x=-1/2..1/2,
   y=-1/2..1/2, color=yellow, numpoints=81)
end:
```

Here we create the vectors and planes that will appear on the screen.

```
> TVectors := map(Tanvec, points, f):
> NVectors := map(Normvec, points, f):
> Planes := map(OscPlane, points):
```

And now we display them.

```
> display({Curve} union TVectors union NVectors union Planes);
```

Computer Exercise 2.4b: Osculating Circles

In this exercise, you can input a space curve and see the **osculating circle**, tangent, and curvature vectors at a given point.

```
> with(plots):
> with(linalg):
> macro(purple=COLOR(RGB, .5,0,.5));
> read(`diffgeo_defs`):
> setoptions3d(scaling=constrained);
```

First, the space curve:

```
> f:= (x) -> [1.5*cos(x), 1.5*sin(x), x/3];
```

Now we define the domain (from `a` to `b`) and pick the point where we'll find the osculating circle by choosing a point in the domain.

```
> a:=0;
> b:=4*Pi;
> Point := Pi/3;
```

Here we calculate the coordinates we need to put the curve on the screen.

```
> Curve := spacecurve(f(x), x=a..b, thickness=2,
   shading=zgrayscale):
```

Now we calculate the coordinates of the tangent and curvature vectors so that we can put them on the screen.

```
> TVector := Tanvec(Point,f):
> CVector := CurvVec(Point,f):
```

Now we find the osculating circle. Note that in the plane, the circle of radius `1/scurvature` centered at `[0, 1/scurvature]` is given by

```
[(1/scurvature)*cos(x),(1/scurvature)*(1+sin(x))]
```

(Here `scurvature` is the scalar curvature. `VCurvature` is the curvature vector.) To move this circle to the osculating plane, we map [1,0] to the tangent vector and [0,1] to `(1/scurvature))*VCurvature`.

```
> OscCircle := proc(t,func,x)
   evalm((1/scurvature(t,func)) * cos(x) * Tangent(t,func) +
            ((1/scurvature(t,func))^2) * (1 + sin(x)) *
   VCurvature(t,func) + func(t))
  end:
```

Now we compute the coordinates we need to put the circle on the screen:

```
> Circle := spacecurve(OscCircle(Point, f, x), x=0..2*Pi,
   color=purple):
```

Finally, we display everything.

```
> display({Curve, Circle, TVector, CVector});
```

Computer Exercise 2.6: Frenét Frame

In this exercise, you can input a space curve and see the Frenét frame at various points.

```
> with(plots):
> with(linalg):
```

```
> setoptions3d(scaling=constrained);
> read(`diffgeo_defs`):
```

First, the space curve:

```
> f:= (x) -> [2*cos(x), 2*sin(x), x/5];
```

Now we define the domain (from `a` to `b`) and the number of divisions of the domain (`div`). Eventually, we will find the Frenét frame at the endpoints (`points`) of each division.

```
> a:=0;
> b:=4*Pi;
> div:=10;
> points := {seq( a + i*(b-a)/div, i=0..div)}:
```

Here we calculate the coordinates we need to put the curve on the screen.

```
> Curve := spacecurve(f(x), x=a..b, thickness=2,
   shading=zgrayscale):
```

Here we create the vectors that will appear on the screen. The tangent vectors are green, the normals are red, and the binormals are blue. For the definitions of the functions `Tanvec`, `Normvec`, and `BiVec`, please see the included file.

```
> TVectors := map(Tanvec, points, f):
> NVectors := map(Normvec, points, f):
> BVectors := map(BiVec, points,f):
```

Finally, we put everything on the screen.

```
> display({Curve} union TVectors union NVectors union BVectors);
```

Computer Exercise 3.1: Tangent Planes to Surfaces

In this exercise, you can input a surface and a point and see the tangent plane at that point. The default surface is a torus.

```
> with(plots):
> with(linalg):
> read(`diffgeo_defs`):
```

First, the surface domain (`x` goes from `a1` to `a2`, `y` from `b1` to `b2`) and a specific point in that domain `(pointx,pointy)`:

```
> a1 := 0;
> a2 := 2*Pi;
> b1 := 0;
> b2 := 2*Pi;
> pointx := Pi/3;
> pointy := 3*Pi/4;
```

The surface:

```
> f := (x,y) -> [(cos(x) + 2)*cos(y),(cos(x) + 2)*sin(y), sin(x)];
> Surface := plot3d(f(x,y), x=a1..a2, y=b1..b2, color=blue):
```

We find the tangent plane by finding the partial derivative of `f` at the chosen point.

```
> XDeriv := map(diff, f(x,y), x);
> XDerivEval := evalf(subs(x=pointx, y=pointy,XDeriv)):
> YDeriv := map(diff, f(x,y),y);
> YDerivEval := evalf(subs(x=pointx, y=pointy, YDeriv)):
```

```
> Plane := plot3d(convert(evalm(x*XDerivEval + y*YDerivEval +
   f(pointx,pointy)), list), x=-1..1, y=-1..1, color=green,
   numpoints=100):
```

Find the coordinate curves :

```
> XCurveEq := (t) -> f(t,pointy);
> XCurve := spacecurve(XCurveEq(t), t=a1..a2, color=yellow,
   thickness=2):
> YCurveEq := (t) -> f(pointx,t);
> YCurve := spacecurve(YCurveEq(t), t=b1..b2, color=orange,
   thickness=2):
```

Putting the vectors on the screen:

```
> XVector := polygonplot3d([f(pointx,pointy),
   convert(evalm(f(pointx,pointy)+ XDerivEval), list)],
   color=yellow, thickness=2):
> YVector := polygonplot3d([f(pointx,pointy),
   convert(evalm(f(pointx,pointy)+ YDerivEval), list)],
   color=orange, thickness=2):
```

Display everything:

```
> display({Surface, Plane, XVector, YVector, XCurve, YCurve});
```

Computer Exercise 3.2a: Curves on a Surface

In this exercise, you can input a surface, its domain, and a curve in its domain, and see the image of the curve on the surface. The default surface is a torus.

```
> with(plots):
> setoptions3d(scaling=constrained);
```

First, the domain (`x` goes from `a1` to `a2`, `y` from `b1` to `b2`):

```
> a1:= 0;
> a2:= 2*Pi;
> b1:= 0;
> b2:= 2*Pi;
> domain:= plot3d([x,y,0], x=a1..a2, y=b1..b2, color=blue,
   style=wireframe):
```

The curve in the domain (that is, a plane curve):

```
> c1 := 0;
> c2 := 2*Pi;
```

The domain of the curve is from `c1` to `c2`.

```
> f1 := (x) -> x;
> f2 := (x) -> Pi;
> DomCurve := spacecurve([f1(x),f2(x),0], x=c1..c2, color=yellow,
   thickness=2):
```

The surface:

```
> g := (x,y) -> [(cos(x) + 2)*cos(y),(cos(x) + 2)*sin(y), sin(x)];
> Surface := plot3d(g(x,y),x=a1..a2,y=b1..b2, color=blue):
```

And the curve on the surface:

```
> SurfCurveEq:= (x)->subs(u=f1(x), v=f2(x), g(u,v));
> SurfCurve:= spacecurve(SurfCurveEq(x), x=c1..c2,
   color=yellow,thickness=2):
```

Now display everything. You will get two separate windows. We recommend viewing the surface in the "`patch`" style, possibly with one of the lighting schemes.

```
> display({domain, DomCurve});
> display({Surface, SurfCurve});
```

You've seen what happens to the line `y=Pi` when it is mapped to the torus. What happens to other straight lines in the domain of the torus? Are their images intrinsically straight in the torus?

Computer Exercise 3.2b: Extrinsic Curvature Vectors

In this exercise, you can input a surface, its domain, and a curve in its domain. You'll see the image of the curve on the surface and the extrinsic curvature vector of this new curve. This exercise is primarily a tool for exploring Problem **3.2**.

```
> with(plots):
> setoptions3d(scaling=constrained);
> read(`diffgeo_defs`):
```

First, the domain (`x` goes from `a1` to `a2`, `y` goes from `b1` to `b2`):

```
> a1:= 0;
> a2:= 2*Pi;
> b1:= 0;
> b2:= 4;
> domain:= plot3d([x,y,0], x=a1..a2, y=b1..b2, color=blue,
   style=wireframe):
```

The curve in the domain (that is, a plane curve):

```
> c1 := 0;
```

The domain of the plane curve goes from `c1` to `c2`.

```
> c2 := 2*Pi;
> Point := Pi;
> f1:= (x) -> x;
> f2:= (x) -> 0.5*x;
> DomCurve:= spacecurve([f1(x),f2(x),0], x=c1..c2, color=yellow,
   thickness=2):
```

The surface:

```
> g := (x,y) -> [cos(x), sin(x), y];
> Surface := plot3d(g(x,y), x=a1..a2, y=b1..b2, color=blue):
```

And the curve on the surface:

```
> SurfCurveEq := (x) -> subs(u=f1(x), v=f2(x), g(u,v));
> SurfCurve:= spacecurve(SurfCurveEq(x),  x=a1..a2,
   color=yellow,thickness=2):
```

Now we find the extrinsic curvature vector of the surface curve.

```
> CVector := CurvVec2(Point, SurfCurveEq):
```

Now display everything. You will get two separate windows. Note: We recommend that you look at the surface in the wireframe style. This will make it easier to see where the curvature vector is.

```
> display({domain, DomCurve});
> display({Surface, SurfCurve, CVector});
```

You may also want to restrict the domain of your surface (for example, look at only half a cylinder), again to make it easier to see the curvature vector.

Computer Exercise 3.3: The Three Curvature Vectors

In this exercise, you can input a surface, a curve on that surface, and a point, and see the extrinsic, normal, and geodesic curvature vectors at that point. The default surface is half a cylinder.

```
> with(plots):
> setoptions3d(scaling=constrained);
> read(`diffgeo_defs`):
```

First, the domain (`x` goes from `a1` to `a2`, `y` goes from `b1` to `b2`):

```
> a1:= 0;
> a2:= Pi;
> b1:= 0;
> b2:= 4;
> domain:= plot3d([x,y,0], x=a1..a2, y=b1..b2, color=blue,
    style=wireframe):
```

The curve in the domain (that is, a plane curve): The domain of the plane curve goes from `c1` to `c2`. `Point` is the point at which we'll show the curvature vectors.

```
> c1 := 1;
> c2 := 3;
> Point := 2;
> f1:= (x) -> x;
> f2:= (x) -> 1 + (x-2)^2;
> domcurve := spacecurve([f1(x), f2(x),0], x=c1..c2, color=yellow,
    thickness=2):
```

The surface:

```
> g := (x,y) -> [cos(x), sin(x), y];
> surf:= plot3d(g(x,y), x=a1..a2, y=b1..b2,   color=blue):
```

And the curve on the surface:

```
> SurfCurveEq := (x) -> subs(u=f1(x), v=f2(x), g(u,v));
> SurfCurve:= spacecurve(SurfCurveEq(x), x=c1..c2,
    color=yellow,thickness=2):
```

Now we find the extrinsic curvature vector of the surface curve and do the calculations we need to put it on the screen.

```
> ExtVector := CurvVec2(Point, SurfCurveEq):
> ExtCurv := VCurvature(Point, SurfCurvEq):
```

Now we break the curvature vector into its normal and geodesic components. First, find the surface normal and get the tangent plane:

```
> XDeriv := map(diff, g(x,y), x);
> XDerivEval := evalf(subs(x=f1(Point), y=f2(Point), XDeriv)):
> YDeriv := map(diff, g(x,y),y);
> YDerivEval := evalf(subs(x=f1(Point), y=f2(Point), YDeriv)):
> SurfaceNormal := vecnormalize(crossprod (XDerivEval, YDerivEval)):
```

```
> Plane := plot3d(convert(evalm(x*XDerivEval + y*YDerivEval +
   g(f1(Point), f2(Point))), list), x=-1..1, y=-1..1, color=green,
   numpoints=64):
```

Now project the extrinsic curvature vector onto the surface normal to get the normal curvature vector. The geodesic curvature vector is the projection of the extrinsic curvature vector onto the tangent plane, but using the fact that (extrinsic curvature vector = normal curvature vector + geodesic curvature vector) saves computation time.

```
> NormCurv := evalm((dotprod
   (ExtCurv,SurfaceNormal))*SurfaceNormal):
> GeoCurv := evalm(ExtCurv - NormCurv):
> NormVector := polygonplot3d ([g(f1(Point), f2(Point)),
   convert(add(g(f1(Point), f2(Point)), NormCurv), list)],
   color=black, thickness=2):
> GeoVector := polygonplot3d ([g(f1(Point), f2(Point)),
   convert(add(g(f1(Point), f2(Point)), GeoCurv), list)],
   color=coral, thickness=2):
```

Now display everything. You will get two separate windows. Note: We recommend that you look at the surface in the wireframe style. This will make it easier to see where the curvature vectors are.

```
> display({domain, domcurve});
> display({surf, SurfCurve, GeoVector, NormVector, Plane,
   ExtVector});
```

Computer Exercise 3.4: Ruled Surfaces

In this exercise, we take a curve (`alpha`) and a unit vector (`r`) and create a ruled surface.

```
> with(plots):
> setoptions3d(scaling=constrained):
```

First, define the curve and its domain (from `a1` to `a2`):

```
> a1:=0;
> a2:=2*Pi;
> alpha := (t) -> [cos(t), sin(t), t/5];
```

Now define the vector:

```
> r := (t) -> [0, cos(t), sin(t)];
```

Now define the surface and the domain of `s` (from `b1` to `b2`). `s` multiplies the vector `r` and determines the width of the surface.

```
> b1:= -1/2;
> b2:= 1/2;
> Surface := (t,s) -> evalm(alpha(t)+ s*r(t));
```

Now choose the number of divisions (`div`) of the curve domain. We'll draw the rulings (calculated by `Rule`) at the endpoints (`points`) of each division.

```
> div :=10;
> points := {seq(a1+i*(a2-a1)/div, i=0..div)}:
> Rule := proc(t)
   polygonplot3d([evalm(alpha(t) + b1*r(t)), evalm(alpha(t) +
   b2*r(t))], color=navy, thickness=2)
end:
```

Now we calculate the coordinates we need to put everything on the screen.

```
> RuledSurf := plot3d(Surface(t,s), t=a1..a2, s=b1..b2,
   style=wireframe):
> Curve := spacecurve(alpha(t), t=a1..a2, color=navy, thickness=2):
> Rulings := map(Rule, points):
```

Finally, we display everything.

```
> display({RuledSurf, Curve} union Rulings);
```

Computer Exercise 5.2: Non-dissectable Polyhedron

This exercise will display on the screen the polyhedron pictured in Figure 5.4. We suggest using "Patch w/ contour." The programming here isn't especially important (or interesting). Feel free to just hit enter after prompt without reading too carefully—the point is to get the polyhedron on the screen and play with it.

```
> with(plots):
```

First, we define a function that takes three points and sides in the triangle they determine.

```
> Triangle := proc(vec1,vec2,vec3)
   evalm(x*(vec2-vec1)+ y*(1-x)*(vec3-vec1)+ vec1)
end:
> MakeFace := proc(vec)
   plot3d(Triangle(vec[1],vec[2],vec[3]),
      x=0..1, y=0..1, color=[vec[1][3],
      vec[2][3],vec[3][3]], numpoints=36)
end:
```

Now we list the vertices that determine each face of the polyhedron.

```
> a := Pi/12;
> h:=1;
> PreFaces :=
   {[[1,0,0], [-1/2,sqrt(3)/2,0],  [-1/2,-sqrt(3)/2,0]], [[cos(a),
   sin(a),h], [cos(a+2*Pi/3), sin(a+2*Pi/3), h],  [cos(a+4*Pi/3),
   sin(a+4*Pi/3), h]],
   [[1,0,0], [cos(a), sin(a), h], [-1/2, -sqrt(3)/2, 0]],
   [[1,0,0],[cos(a), sin(a), h], [cos(a+2*Pi/3), sin(a+2*Pi/3), h]],
   [[1,0,0], [-1/2, sqrt(3)/2, 0], [cos(a+2*Pi/3),  sin(a+2*Pi/3),
   h]], [[cos(a), sin(a), h], [-1/2, -sqrt(3)/2,0], [cos(a+4*Pi/3),
   sin(a+4*Pi/3), h]], [[-1/2, sqrt(3)/2, 0], [-1/2, -sqrt(3)/2, 0],
   [cos(a+4*Pi/3), sin(a+4*Pi/3), h]], [[-1/2, sqrt(3)/2,0],
   [cos(a+2*Pi/3), sin(a+2*Pi/3), h], [cos(a+4*Pi/3), sin(a+4*Pi/3),
   h]]}:
```

Now we make the actual faces out of these vertices and display the polyhedron.

```
> Faces := map(MakeFace, PreFaces):
> display(Faces);
```

Computer Exercise 5.5: Sign of (Gaussian) Curvature

In this exercise you can display a surface with areas of positive (Gaussian) curvature magenta and areas of negative (Gaussian) curvature blue. The formula used to compute the curvature is developed in Problem **7.1**.

```
> with(plots):
> with(linalg):
> read(`diffgeo_defs`):
> setoptions3d(scaling=constrained):
```

First, define the surface:

```
> f := (x,y) -> [(cos(x) + 2)*cos(y), (cos(x) + 2) *sin(y), sin(x)]:
```

Now display it, coloring areas of positive Gaussian curvature magenta and areas of negative Gaussian curvature blue.

```
> Surface := plot3d(f(x,y), x=0..2*Pi, y=0..2*Pi, color=
   [Heaviside(GaussCurv(f,x,y)), 0,1]):
> display(Surface);>
```

Computer Exercises 6.1: Multiple Principle Directions

This exercise goes with Problem **6.1.c** and will allow you to display the surfaces mentioned there. Remember that `f` must be twice differentiable and that we need `f(0)= 0 = f'(0)` and `f(x)= f(-x)`.

```
> with(plots);
> setoptions3d(scaling=constrained);
> f := (theta) -> 1 - cos(4*theta);> cylinderplot([r,theta,
   f(theta)*r^2], r=0..1/2, theta=0..2*Pi, color=orange);
```

You may want to restrict the domain of `theta` and look at only, say, half or three-quarters of your surface. We recommend using the patch style and one of the lighting schemes.

Computer Exercise 6.3: Gauss Map

In this exercise, you can input a surface and see the image of a portion of it under the Gauss map. The default surface is a torus.

```
> with(plots):
> with(linalg):
> setoptions3d(scaling=constrained):
> read(`diffgeo_defs`):
```

First, the domains. `Surfdomain` is the domain of the surface (`x` goes from `a1` to `a2`, `y` goes from `b1` to `b2`). `Gaussdomain` is a subset of `Surfdomain` (`x` goes from `c1` to `c2`, `y` goes from `d1` to `d2`). You will see the image of the `Gaussdomain` under the Gauss map. We will also outline the `Gaussdomain` to make it easier to see what is going on.

```
> a1 := 0;
> a2 := 2*Pi;
> b1 := 0;
> b2 := 2*Pi;
> c1 := 2*Pi/3;
> c2 := 4*Pi/3;
> d1 := 2*Pi/3;
> d2 := 4*Pi/3;
> surfdomain := plot3d([x,y,0], x=a1..a2, y=b1..b2, color=blue,
   numpoints=100):
> Gaussdomain := plot3d([x,y,0], x=c1..c2, y=d1..d2, color=yellow,
   numpoints=144):
> outline1 := spacecurve([x,d1,0], x=c1..c2, color=green,
   thickness=2):
> outline2 := spacecurve([c1,y,0], y=d1..d2, color=red,
   thickness=2):
> outline3 := spacecurve([x,d2,0], x=c1..c2, color=magenta,
   thickness=2):
> outline4 := spacecurve([c2,y,0], y=d1..d2, color=black,
   thickness=2):
```

The surface and the image of the `Gaussdomain` and its outline in the surface (`SurfPatch` and the `surflines`):

```
> f := (x,y) -> [(cos(x) +2) *cos(y), (cos(x) +2) *sin(y), sin(x)];
> Surface := plot3d(f(x,y), x=a1..a2, y=b1..b2, color=blue):
> SurfPatch := plot3d(f(x,y), x=c1..c2, y=d1..d2, color=yellow):
> surfline1 := spacecurve(f(x,d1), x=c1..c2, color=green,
   thickness=2, numpoints=12):
> surfline2 := spacecurve(f(c1,y), y=d1..d2, color=red, thickness=2,
   numpoints=12):
> surfline3 := spacecurve(f(x,d2), x=c1..c2, color=magenta,
   thickness=2, numpoints=12):
> surfline4 := spacecurve(f(c2,y), y=d1..d2, color=black,
   thickness=2, numpoints=12):
```

Now the sphere for the Gauss map:

```
> SphereEq := (x,y) -> [sin(x)*cos(y), sin(x)*sin(y), cos(x)];
> Sphere := plot3d(SphereEq(x,y), x=0..Pi, y=0..2*Pi,
   color=turquoise, numpoints=225):
```

The image of the `Gaussdomain` and its outline:

```
> GaussImage := plot3d(GMapEval(f,x,y), x=c1..c2, y=d1..d2,
   color=yellow, numpoints=225):
> Gline1 := spacecurve(GMapEval(f,x,d1), x=c1..c2, color=green,
   thickness=2,  numpoints=10):
> Gline2 := spacecurve(GMapEval(f,c1,y), y=d1..d2, color=red,
   thickness=2,  numpoints=10):
> Gline3 := spacecurve(GMapEval(f,x,d2), x=c1..c2, color=magenta,
   thickness=2, numpoints=10):
> Gline4 := spacecurve(GMapEval(f,c2,y), y=d1..d2, color=black,
   thickness=2, numpoints=10):
```

Now we display everything. You may want to look at the surface window and the `Gausswindow` in the wireframe style—it makes it easier to see the outline of the yellow patch.

```
> display({surfdomain, Gaussdomain, outline1, outline2, outline3,
   outline4});
```

```
> display({Surface, SurfPatch, surfline1, surfline2, surfline3,
   surfline4});
> display({Sphere, GaussImage, Gline1, Gline2, Gline3, Gline4});
```

Computer Exercise 6.6: Helicoid to Catenoid

In this exercise, you can look at a helicoid, a catenoid, and the intermediate surfaces.

```
> with(plots):
> setoptions3d(scaling=constrained):
```

`a` is a constant that affects the shape of the surfaces.

```
> a:=2;
```

Helicat is a function that interpolates between the helicoid

$$(x,y) \longrightarrow [x\cos(ay),\ x\sin(ay),\ y]$$

and the catenoid

$$(x,y) \rightarrow \left[\tfrac{1}{a}\sqrt{1+(ax)^2}\,\cos(ay), \tfrac{1}{a}\sqrt{1+(ax)^2}\,\sin(ay), \tfrac{1}{a}\operatorname{arcsinh}(ax)\right].$$

`Helicat(x,y,0)` is the helicoid, `Helicat(x,y,1)` is the catenoid, and `Helicat(x,y,t)`, where `t` is between `0` and `1` is an intermediate surface.

```
> Helicat := (x,y,t)->              [((1/a)*(sqrt(1+(a*x)^2))*t+
   (1-t)*x)*cos(a*y), ((1/a)*(sqrt(1+ (a*x)^2))*t+
   (1-t)*x)*sin(a*y), ((1/a)*arcsinh(a*x))*t + (1-t)*y];
> Surface := plot3d(Helicat(x,y,1), x=-1..1, y=0..2*Pi/a,
   color=green):
> display(Surface);
```

Are the intermediate surfaces minimal?