Exercise 1
Scientific Data Visualization
CSCE 5320 – Spring 2021
Distributed: Thursday, January 14
Due: Thursday, February 4

[Solutions to this assignment must be submitted via CANVAS prior to midnight on the due date. Submissions no more than one day late will not be penalized. Submissions up to one week late will be penalized 10 points. Submissions more than week late and less than two weeks late will be penalized 20 points. Submissions will not be accepted after two weeks. THIS IS AN INDIVIDUAL ASSIGNMENT]

Purpose: Learn to use the language/package how to generate a warping (elevation plot) of a 2D function. It is expected (but not required) that you will use the same environment in future assignments.

What to do: Consider the graph of a two-variable function $z = f(x,y)$ in Section 2.1 of Chapter 2. Assume you know the ranges of interest $[x_{min},x_{max}]$ and $[y_{min},y_{max}]$ of the two independent variables.

Now let $f(x,y) = e^{-(x^2+y^2)}$. Consider the range $X=[-1x1]$ and $Y=[-1x1]$. You will implement two cases. For case (i), divide the *X* and *Y* ranges into a 30x30 grid. provide the elevation plot superimposed on a domain plane and provide the elevation plot without the domain plane (see Figure). For case (ii), divide the *X* and *Y* ranges into a 10x10 grid. provide the elevation plot superimposed on a domain plane and provide the elevation plot without the domain plane.

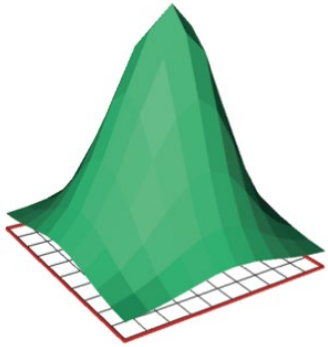

Figure: Elevation plot superimposed on domain plane

Grading consideration: If you are successful in creating both instances of each case (that is, with and without domain plane) the maximum grade will be 100 points. If one or more of the four plots is missing, the maximum grade will be 90 points.

Hand-in: (i) The four elevation plots; (ii) the computer code; (iii) a list of significant sources of information (websites, etc.); (iv) a discussion of what you found to be most challenging about the assignment (10 points allocated on the grading scale).

The Future. This assignment will form the basis of several soon-to-be-assigned tasks – adding Phong or perhaps Gouraud shading, texture, and transparency.

Hints: OpenGL code snips are on next page.

```
#include <math.h>                                   //For expf()
#include <GLUT/glut.h>                              //GLUT library
#include "Quad.h"                                   //the Quad class definition
//*************************************************************************
class Quad                              //A simple class for passing a quad polygon to OpenGL
{
public:

        Quad()                  //Constructor: Starts drawing the quad
        { glBegin(GL_QUADS); }
    ~Quad()
     {}

void    addPoint(float x, float y, float z)         //Adds a new vertex to the quad
        { glVertex3f(x,y,z); }

void    addNormal(float* n)     //Adds a new normal to the quad
        { glNormal3f(n[0],n[1],n[2]); }

void  draw()                    //Drawing function: ends the quad definition. This also draws the quad.
        { glEnd(); }
};
//*************************************************************************
```

```
float X_min,X_max;                          //X = [X_min,X_max]
float Y_min,Y_max;                          //Y = [Y_min,Y_max]
int   N_x,N_y;
float dx = (X_max-X_min)/(N_x-1);           //x size of a cell
float dy = (Y_max-Y_min)/(N_y-1);           //y size of a cell
float f(float,float);                       //the function to visualize

for(float x=X_min;x<=X_max-dx;x+=dx)
  for(float y=Y_min;y<=Y_max-dy;y+=dy)
  {
     Quad q;
     q.addPoint(x,y,f(x,y));
     q.addPoint(x+dx,y,f(x+dx,y));
     q.addPoint(x+dx,y+dy,f(x+dx,y+dy));
     q.addPoint(x,y+dy,f(x,y+dy));
     q.draw();
  }
```

Listing 2.1. Drawing a height plot.

```
#include <math.h>

class Point3d
{
public:
 union {
  float data[3];
  struct { float x,y,z; };                    //coordinates
 };

                              Point3d(float xx=0, float yy=0, float zz=0) : x(xx), y(yy), z(zz)
                              { }
                              Point3d(const float* p): x(p[0]),y(p[1]),z(p[2])
                              { }

 void      flip() {x=-x; y=-y; z=-z; }
 float             dist(const Point3d &p) const { return sqrtf( (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y) + (z-p.z)*(z-
p.z) ); }
 float             dist2(const Point3d &p) const { return (x-p.x)*(x-p.x) + (y-p.y)*(y-p.y) + (z-p.z)*(z-p.z); }
 float             norm() const { return sqrtf(x*x+y*y+z*z); }
 float             norm2() const { return x*x+y*y+z*z; }
 float             dot(const Point3d &v1) const { return x*v1.x + y*v1.y + z*v1.z; }
 float     dot(const float* v) const  { return x*v[0] + y*v[1] + z*v[2]; }
 bool              nonzero() const { return x!=0 || y!=0 || z!=0; }

 float& operator[](int idx) { return data[idx]; }

 operator          float*() { return &x; }
 operator          const float*() const { return &x; }
 bool      isApprox(const Point3d& p,float tol) const { return fabs(x-p.x)+fabs(y-p.y)+fabs(z-p.z)<tol; }

 Point3d    cross(const Point3d &v2) const { return Point3d(y*v2.z-v2.y*z, v2.x*z-x*v2.z, x*v2.y-v2.x*y); }
 Point3d           operator-(const Point3d &p) const { return Point3d(x-p.x, y-p.y, z-p.z); }
 Point3d    operator-() const { return Point3d(-x,-y,-z); }
 Point3d           operator+(const Point3d &p) const { return Point3d(x+p.x, y+p.y, z+p.z); }
 Point3d&   operator+=(const Point3d &p) { x+=p.x; y+=p.y; z+=p.z; return *this; }
 Point3d&   operator-=(const Point3d &p) { x-=p.x; y-=p.y; z-=p.z; return *this;}
 Point3d           operator*(float k) const { return Point3d(k*x,k*y,k*z); }
 Point3d&   operator*=(float k) { x*=k; y*=k; z*=k; return *this; }

 bool operator<(const Point3d &p) const { return x<p.x && y<p.y && z<p.z; };

 Point3d           operator/(float k) const { return Point3d(x/k, y/k, z/k); }
 Point3d&   operator/=(float k) { x/=k; y/=k; z/=k; return *this; }
```

```
 float          normalize() { float n = norm(); if (n<1.0e-6) n=1; x/=n; y/=n; z/=n; return n; }

 Point3d(const Point3d &p): x(p.x),y(p.y),z(p.z)
                        { }

 Point3d&       operator=(const Point3d &p)
                        {
                          if (&p==this) return *this;
                          x=p.x; y=p.y; z=p.z;
                          return *this;
                        }

 Point3d&       operator=(const float* p)
                        {
                          x=p[0]; y=p[1]; z=p[2];
                          return *this;
                        }

void            interpolate(const Point3d& a, const Point3d& b,float t)
                        { float tt=1-t; x = a.x*t+b.x*tt; y = a.y*t+b.y*tt; z = a.z*t+b.z*tt; }

 float          distPlane(const Point3d &normal, const Point3d &ppoint)
{
  return (x-ppoint.x)*normal.x + (y-ppoint.y)*normal.y + (z-ppoint.z)*normal.z;
}

 //set this to a vector perpendicular to vec
 void setPerp(const Point3d &vec)
{
  float a = fabsf(vec.x), b=fabsf(vec.y), c=fabsf(vec.z);

  x = y = z = 0.0f;

  if(a<b && a<c) x = 1.0f;
  else if(b<c) y = 1.0f;
  else z = 1.0f;

  *this = cross(vec);
 }
};
```