Object-Oriented Scientific Visualization

Galina Istatkova

Abstract: *In this paper C++ classes for function visualization in MFC Windows program are described. Classes are developed in Microsoft C++ Visual. Using these classes 3D polygonal approximation of function can be rendered as color map using Windows GDI or as wireframe, coloured, flat or smooth shading surface using OpenGL. Computing normal vectors for the enlightenment of function surface is discussed. Spline interpolation and saving function image to JPEG files are discussed. Flat and smooth function images generated by Demo program are given.*

Key words: *Object-oriented scientific visualization, 3D Function Image, flat and smooth shading, OpenGL, C++ Visual.*

INTRODUCTION

Function visualization is a routine operation in scientific applications, but is heavy enough for programming. Function visualization was developed as separate module - collections of C++ classes that can be used in many applications. These classes are developed using Microsoft C++ Visual and intended for object-oriented MFC program in Windows. The basic classes are CXYRegGrid and CXYZFun. Class CXYRegGrid is used for visualization of 2D function based on Windows GDI (Graphics Device Interface). Regular X-Y grid with arbitrary combination of linear and logarithmic scales is created by methods of this class. 3D function can be rendered in X-Y grid as colourmap. Class CXYZFun is intended for 3D rendering of u=F(x,y). Using OpenGL u=F(x,y) can be rendered as wireframe, coloured, flat or smooth shading surface. OpenGL is an standard procedural software interface for producing 3D graphics originally developed by Silicon Graphics[4]. After Windows NT 3.5 OpenGL is part of Microsoft Windows. Using CXYRegGrid u=F(x,y) can be rendered as colourmap. The basic goal of these classes development is significantly decreasing the development time of scientific MFC program in Windows. In this paper, 3D function rendering using CXYZFun and CXYRegGrid classes are described.

1. Creating Polygonal Approximation of function

Class CXYZFun doesn't use MFC root class CObject as base class. Instead of it structure LAPPARAM1 is used as base class. LAPPARAM1 parameters (min and max colour, size of grid, step over the grid) and some others data-members of class are set by constructors. For calculating the examples of 3D function surfaces some simple functions such as sinvx^2+y^2, $x^2/2p+y^2/2q$, $x^2/p-y^2/q$, are defined in CXYZFun. Pointers to these functions and some others parameters are saved in static table. Entry in this table is:

```
Typedef struct tag FUNENTRY{
        bfun pfun;// pointer to function
        char funname[80];// analytic function definition
        WRECT rc;// size of grid
        double h;} FUNENTRY;
```

Type of pointer to function is:

```
typedef double  (*bfun )( double, double  );
```

Polygonal approximation of function is calculated over the regular coordinate grid in (X, Z) plane. Coordinates of vertices are stored in three objects of class CDoubleArray: m_xreg (x coordinate) m_yreg (z coordinate), m_ureg (y coordinate corresponded to function value). Function vertices are calculated with ComputeRegGrid(void). The part of code is:

```
double *px,*py,*pu; // pointers to m_xreg, m_yreg, m_zreg
px = m_xreg; py = m_yreg; pu = m_ureg;
double x,y,min,max; // min and max value of function
min = max = (*m_pfun)(wrc.xmin,wrc.ymin);
y = wrc.ymin;
for ( int i = 0;i < m_m; i++){
   x = wrc.xmin;
```

```
    for (int j=0;j<m_n;j++){
            *pu = (*m_pfun)(x,y); // calculate  u= f(x,y) for [i,j]
            if (*pu < min) min = *pu;
            if (*pu > max) max = *pu;
            pu++;
            *px++ = x; *py++ = y;
            x += h;}
    y += h;}
```

To calculate function defined in another module (imported function) ComputeRegGrid(const FUNENTRY& funpar) is used. This method sets an entry for imported function in the function table and then calls ComputeRegGrid(void).

To visualize data calculated in another program SetData(int n, int m, double h, WREC wrc, double* x,double* y, double* u) method is used. This function sets m_xreg, m_yreg, m_zreg from arrays x,y,z. Created or imported polygonal approximation of function can be visualized as colour map using CXYRegGrid object or as wireframe, colored, flat or smooth shading surface using OpenGL.

2. Color Map Representation

CXYregGrid and CXYZFun objects are embedded in document or view class in MFC program. For colour map representation, CXYRegGrid object must be created according to function parameters. Code for setting parameters of CXYRegGrid object m_fungrid for CXYZFun object m_fun embedded in document class is:

```
GRIDPARAM gr = pDoc ->m_fungrid.GetParam();//Get GRIDPARAM parameters
CString str("Function is ");
str += pDoc ->m_fun.GetFunName();
strcpy( gr.subtitle.title,str);// Set analytic function definition into subtitle
gr.leg.crmax = pDoc ->m_fun.GetMaxColor();//Set values for ColorMap Legend
gr.leg.crmin = pDoc ->m_fun.GetMinColor();
gr.leg.max = pDoc ->m_fun.GetMaxValue() ;
gr.leg.min = pDoc ->m_fun.GetMinValue() ;
gr.wrc = pDoc->m_fun.GetWorldWindow();// Set X-Y grid rectangle with function regular grid size
pDoc ->m_fungrid.Set( gr ); // Set new X-Y grid parameters
```

Structure GRIDPARAM includes all parameters of CXYRegGrid and is used as base class of CXYRegGrid. Pointer to CXYRegGrid object is passed as parameter in DrawRegPointS for colourmap representation:

```
void CXYZFun::DrawRegPointS(CDC * pDC, CXYRegGrid* pgrid, BOOL border/*=TRUE*/)
{
   for ( int i=m_m-1;i >= 0 ;i--){
            for (int j=m_n-1;j >= 0; j --){
            DrawRegPoint(pDC,  pgrid, i, j);}// Draws coloured rectangle for [i,j] point of function grid
            }
```

To render coloured rectangle for [i,j] point DrawRegPointS calls DrawRegPoint:

```
void CXYZFun::DrawRegPoint(CDC * pDC, CXYRegGrid* pgrid,int i,int j)
{
   double x,y,u;
// CDoubleArray is linear and for two dimensional grid linear index must be calculated
   int index = m_n*i + j;
   x = m_xreg[index]; y = m_yreg[index]; u = m_ureg[index];
   CBrush brush; COLORREF cr;; WRECT wrc;
   cr = ConvertToColor(u);// Convert function value to colour
   brush.CreateSolidBrush(cr);
            wrc.xmin = x-h/2.0f; // define WRECT for color rectangle
            wrc.xmax = x+h/2.0f; wrc.ymin = y-h/2.0f; wrc.ymax = y+h/2.0f;
   pgrid -> DrawWRECT( pDC,  wrc,&brush, 0);// DrawWRECT is CXYRegGrid primitive for colormap
}
```

Rectangle colour is calculated by function ConvertToColor(double u) that converts function value to R,G,B components. Code for calculating is:

```
// Get R,G,B components of min(crmin) and max(crmax) colour
BYTE rmin = GetRValue(crmin), bmin = GetBValue(crmin), gmin = GetGValue(crmin);
```

```
BYTE rmax = GetRValue(crmax), bmax = GetBValue(crmax), gmax = GetGValue(crmax);
double du ;
du = (m_umax - m_umin > 0.0f)?( (u-m_umin)/(m_umax-m_umin) ): (0.0f);
BYTE r =  rmin + (BYTE)( du*(double)(rmax-rmin));// Calculate colour components
BYTE g =  gmin + (BYTE)( du*(double)(gmax-gmin));
BYTE b =  bmin + (BYTE)( du*(double)(bmax-bmin));
COLORREF cr = RGB(r,g,b);
```

3. Wireframe and coloured model of surface

Wireframe and coloured model are the simplest ways for surface visualization in OpenGL: The three or four neighbouring points are connected to define OpenGL polygon primitive. Using GL_LINE attribute for polygon, wireframe model is created. For coloured model, GL_FILL attribute is used. Vertex colour is calculated using function value (as in color map). Hidden-surface removal is done by OpenGL using the depth buffer (sometimes called a z-buffer)[4]. One method RenderFun is used for wireframe and colouerd models:

```
void CXYZFun::RenderFun(GLenum fillmode, GLenum polygonstyle)
{
  int index;
  GLfloat v0[3], v1[3], color[3];
  ::glPolygonMode( GL_FRONT, fillmode ); // Set drawing mode for polygon
  ::glPolygonMode( GL_BACK, fillmode );
  for ( int i = 0;i < m_m-1; i++){
        ::glBegin( polygonstyle );
        for (int j=0;j<m_n;j++){
              index = m_n*i + j;// [i,j]
              v0[0] =(GLfloat) m_xreg[index];
               v0[1] = (GLfloat)m_ureg[index];
              v0[2] =(GLfloat)m_yreg[index];
              index += m_n;// [i+1,j]
              v1[0] = (GLfloat)m_xreg[index];
              v1[1] = (GLfloat)m_ureg[index];
              v1[2] =(GLfloat)m_yreg[index];
              ConvertToColor( v0[1],color);
              ::glColor3fv( color);
              ::glVertex3fv( v0 );
              ::glVertex3fv( v1 ); }
        ::glEnd();}
}
```

4. Flat and Smooth shading of surface using normal vectors

Flat and smooth shading of surface uses OpenGL lighting model. To use the enlightenment normal vectors to surface must be calculated. Normal vector defines orientation of surface relative to light sources. OpenGL uses this vector to determine how much light each pixel of a given surface receives. When one normal is given alongside the polygon, polygon's pixels are enlightened with the same colour value. The result is known as flat shading rendering. To realize an even more aesthetic enlightenment, we must give to OpenGL one normal per every vertex of surface. The result is good looking and is known as smooth shading rendering. Using the three consecutive vertices of polygon v_1, v_2, v_3, normal n is calculated as cross product:

$n = [v_1-v_2]x[v_2-v_3]$.

Vertices of polygon are selected keeping polygon orientation consistent. It is very important to get the quality of lighting. In CXYZFun method RenderFun_normal() realizes flat shading using triangles as polygons. The part of code for computing normal vector and drawing triangle is:

```
// Compute vector normal
d1[0] = v0[0] - v1[0]; d1[1] = v0[1] - v1[1]; d1[2] = v0[2] - v1[2];
d2[0] = v1[0] - v2[0]; d2[1] = v1[1] - v2[1]; d2[2] = v1[2] - v2[2];
normcrossprod(d1, d2, norm);// This function computes vector normal
  ::glBegin(GL_TRIANGLES); // Render triangle
```

```
ConvertToColor( v0[1],color);
::glColor3fv( color);
::glNormal3fv(norm);
::glVertex3fv( v0 );
ConvertToColor( v1[1],color);
::glColor3fv( color);
::glNormal3fv(norm);
::glVertex3fv( v1 );
ConvertToColor( v2[1],color);
::glColor3fv( color);
::glNormal3fv(norm);
::glVertex3fv( v2 );
::glEnd();
```

Flat shading of $u = 3\sin\sqrt{x^2+y^2}$ is shown in fig.1.

Smooth shading is implemented using quadrangles (fig.2). Algorithm for calculating normal vectors for all vertices is:

- Calculate the normal for all quadrangles;
- Define four neighbouring quadrangles for [i,j] vertex;
- Average the normal vectors for neighbouring quadrangles;

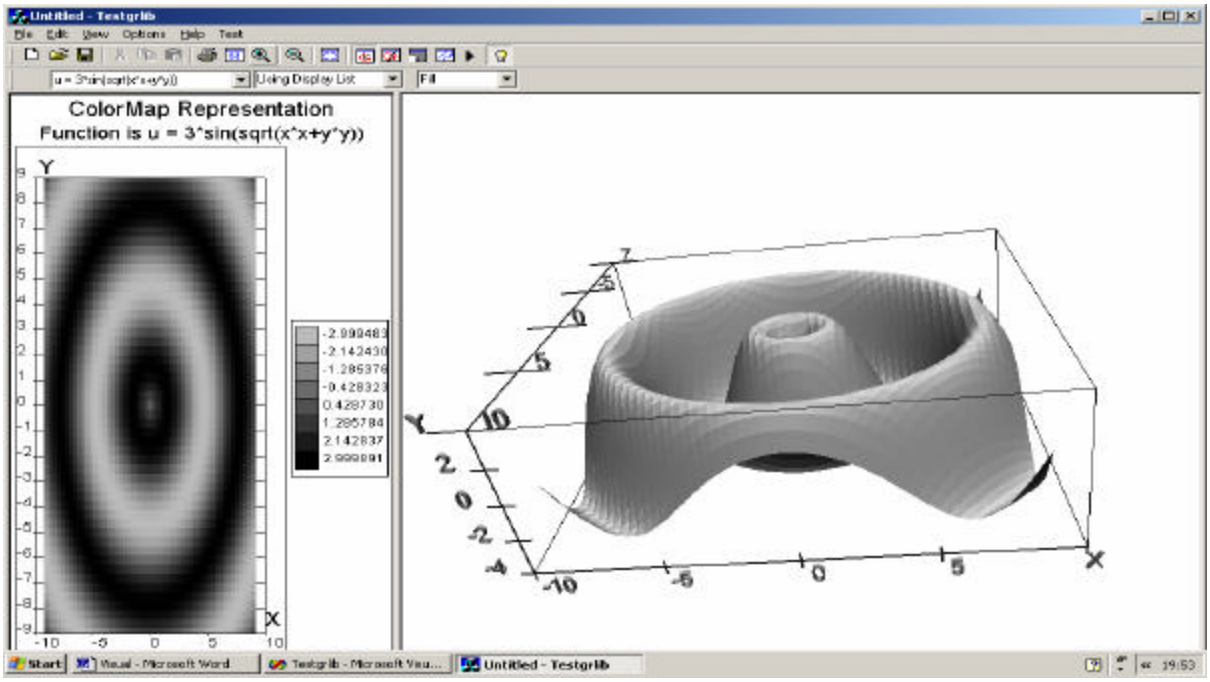

Fig 1. Flat shading.

5. Spline Interpolation of Function

Smooth and flat shading of $3\sin\sqrt{x^2+y^2}$ surfaces (fig.1-2) are drawn by approximating them with large number of polygons - 2500. If spline interpolation is used then only a few control points are needed. OpenGL supports almost all splines in use today, including B-splines, NURBS (Non-Uniform Rational B-Spline) surfaces, Bezier curves and surfaces, and Hermite splines[4]. NURBS interpolation for function defined in CXYZFun class is used in method DrawNurbs2(Glenum mode). DrawNurbs2 uses three function from GLU (OpenGL Utility Library):

- glMap2f defines NURBS parameters such as number of control points;
- glMapGrid2 specifies the linear grid mappings between the i and j integer grid coordinates to the u and v floating-point evaluation map coordinates.
- glEvalMesh evaluates a series of evenly spaced map domain values.

Function image can be saved to JPEG file using method CXYZFun::WriteJPEGFile(char *filename,int Width, int Height). This method use for JPEG compression dynamic library mkOpenGLJPEGImage. This library can be downloaded from site www.comp.nus.edu.sg (National University of Singapore, School of Computing).

6. Bounding Cuboid

Bounding cuboid around the function image (fig. 1-2) is implemented in class CParal. Marker's values are drawing using function DrawString(const char* s) of class COutlineFont, embedded in class CParal. Class COutlineFont had to be developed because OpenGL doesn't provide direct font support as Window GDI. There are three basic ways to output text in OpenGL:

- Creation bitmap for each character of font;
- Creation texture containing an entire character set;
- Creation 3D geometric model for each character of font.

Class COutlineFont uses the third method. 3D geometric model of characters are created in constructor using OpenGL Windows API wglUseFontOutlines function. This function creates a display list for every character of font using Windows font model currently selected in Windows device context. Geometric model coordinates are in normalized coordinate system [1:0]x[1:0]. Before drawing, character model is scaled according to the cuboid size and the number of "good" intervals. The number of "good" interval is defined using Lewart algorithm [3].

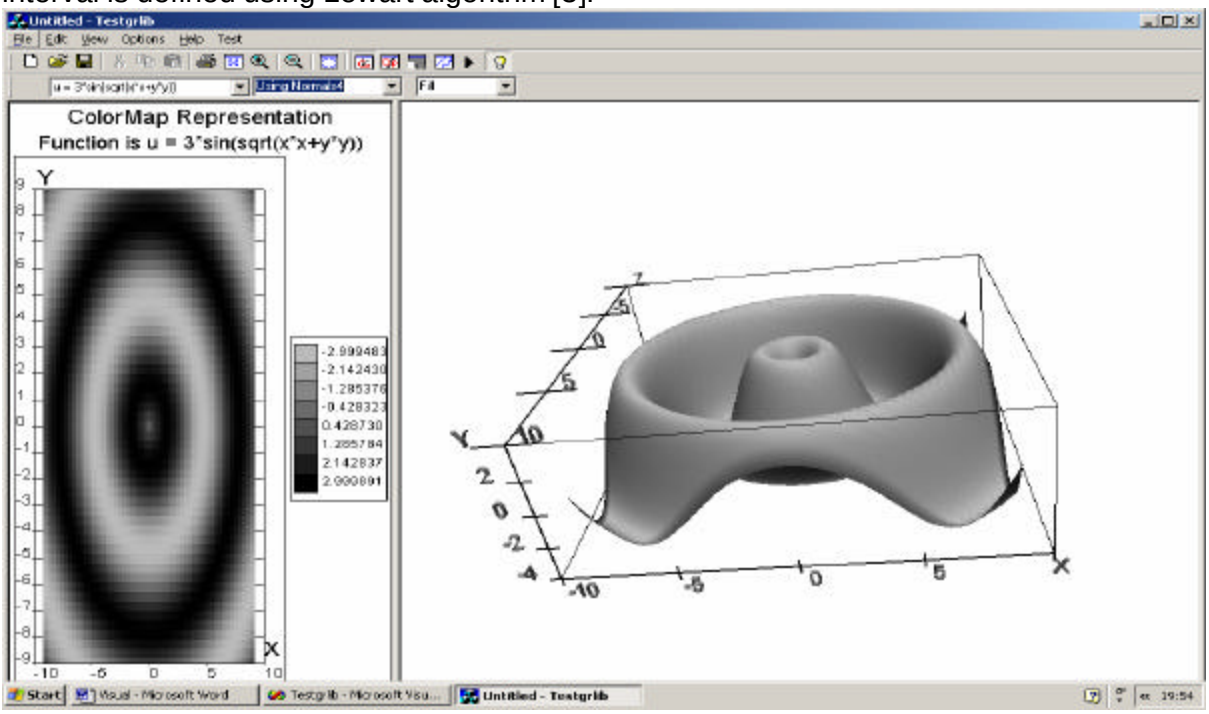

Fig. 2. Smooth shading.

7. Demo Program

The main window of Demo program using CXYZFun and CXYRegGrid classes is shown in fig.1-2. The program uses SDI (Single Document Interface) with two panes. CXYZFun and CXYRegGrid are embedded objects in document class[2]. Colour map representation of u=F(x,y) is rendered in the first pane. In the second pane function is visualized as 3D surface. Three List box contols embedded in CDialogBar control are intended for function and drawing mode selection. Colors, light and material properties and some others parameters are set with modal and modeless dialog. Standard MFC CView class is used as base class in the first pane. Colour map is drawn in function OnDraw. Part of code is:

```
pDoc->m_fungrid.Draw( pDC );// m_fungrid is CXYRegGrid object embedded in CDocument class
pDoc->m_fun.DrawRegPointS( pDC,&pDoc->m_fungrid,FALSE);// m_fun is object of CXYZFun
```

For the second pane, there isn't base view class in MFC. To use OpenGL in MFC program, a new view class, such as COpenGLView must be developed[5]. In Internet various implementation of COpenGLView class can be founded. In the Demo program COpenGLView class from ClassGL library developed in Solid Graphics (www.solidgraphics.com). is used. In addition to standard functionality of an OpenGL window, this class provides several interaction mode interfaces, support for objects manipulation and navigation in 3D space, printing and scale control methods. Using this class Demo program realizes zoomin, zoomout, windowing, rotation and animation. Function surface is rendered in DrawScene – virtual function of COpenGLView class. Part of code is:

```
switch ( m_renderstyle ) // m_renderstyle is currently selected rendering style
{
case NORMALS:
pDoc->m_fun.RenderFun_normal();// Flat shading
break;
case NORMALS4:
pDoc->m_fun.Render_normal4();// Smooth shading
break;
case COLORQUADS:
pDoc->m_fun.RenderFun( GL_FILL,GL_QUAD_STRIP);// Coloured surface
break;
….
}
m_paral->Draw();// render bounding cuboid
```

Class CXYZFun is independent from COpenGLView class and can be used with version that is needed for application.

8. Conclusion

CXYRegGrid and some others embedded as objects in it: CLegend (legend window), CRegScale (linear or logarithmic scale), CTitle(title and subtitle) are collected in Graph2D library. This library is an object-oriented release of the procedural library, implemented in C [1]. Graph2D library is intended for 2D applications based on Windows GDI and can be used by programmer has not special knowledge of computer graphics. Class CXYFun implemented in Graph2D is 2D analogy of CXYZFun and simplifies y=F(x) rendering. To use class CXYZFun it is necessary to know some basic conception of OpenGL and how OpenGL running on a Windows platform. Rendering methods of CXYZFun based on OpenGL are intended for high-quality 3D graphics application.

REFERENCES

[1] ?????????, ?., ?. ???????. ???????? ?????????? ?? ?? ?? ?????????? ????????? ?? ????? ?? ?????? ??????? ? ??????????. ?????????? ?????????? ? ???????????'95. ??????? ?????? ???????. ????? 1995.

[2] ?????????? ?????? ??., Visual C ++, ?????? ???????. ???-?????-???, 1998, ISBN 954-685-0025.

[3] ??????, ?.?. ????????? ?????????????? ?????? ?????? ? ???? ????????. ???????????, 14,1983, 76-79.

[4] Nejder, Jackie, Tom Davis and MasonWoo. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Release 1, Reading, MA: Addison-Wesley, 1993. ISBN 0-201-63274-8. Red Book.

[5] Fosner Ron. – OpenGL Without Pain:Creating a Reusable 3D View Class for MFC. Microsoft System Journal,1995.

ABOUT THE AUTHOR

Engineer, research worker Galina Istatkova , Institute of Computer and Communication System, Bulgarian Academy of Science. Phone: 73-29-51(137), GSM - 098 89 30 59, E-mail: g_istatkova@mail.bg.