A Three Dimensional Mesh Viewer

New Mexico
Supercomputing Challenge
Final Report
April 4, 2007

Team 59
Manzano High School

Team Members:
Chen Zhao
Luke Tyler
Johnathan Fisher

Teacher:
Steven Schum

Project Mentor:
Alexander Booker

Table of Contents:

Executive Summary:

Visualization and quality examination of a mesh or grid is important to numerical simulations that solve science and engineering problems in a discretized domain. Assignment of boundary conditions at desired cell faces is equally important and sometimes difficult to realize if the computational domain is geometrically complex. We plan to write a graphical user interface (GUI) software to view and examine any three-dimensional volume mesh. The GUI reads a previously generated mesh data file in certain formats and displays the mesh on the screen. In addition, the GUI has the following features on the mesh:

1. zoom in/out
2. rotation and translation
3. interrogation for quality assessment
4. modification via redistribution of vertices
5. selection of any mesh identities (cell, face, line and vertex)
6. output of any selected mesh identities
7. combination of any selected faces for boundary condition assignment

Problem Description:

3D meshes have been used for many years now but no one has made a program to modify the points and measure the volume/surface area. Meshes can be generated but not modified. Team 59 is designing a program to modify the points on a mesh in a 3D work zone. The user will input a file filled with coordinates of points on a 3D object and the program will take those coordinates and make a 3D render of the mesh. The user will be able to drag points on the mesh and make various calculations of the mesh.

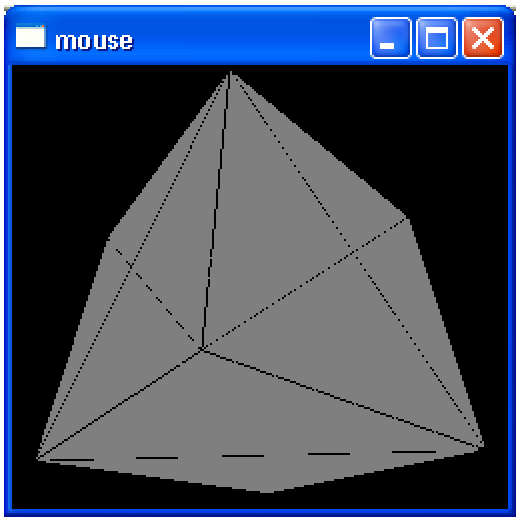
Problem Solution:

Using C++ aided with the OpenGL graphics system, we will create a program that renders the mesh and allows the user to rotate, measure, and modify the mesh in a real time environment. With OpenGL our program will run quickly and efficiently.
In writing our code, we input a text file with coordinates and connectivity points. (See Table)

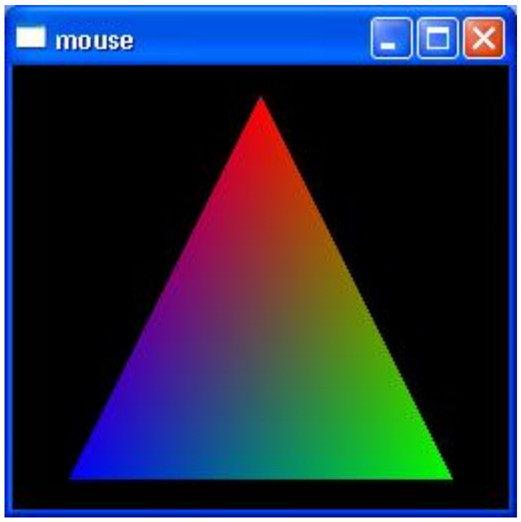
Results of Our Study:

Unfortunately, we weren’t able to get many results for our project. We had difficulties learning the languages of OpenGL and C++. One of the things we had trouble with was learning the mouse events for rotation, translation and zooming. However, we were able to learn some of the commands of OpenGL which will help us in later projects and programming. We were also able to get a simple program running that displays the mesh and allows the user to translate and rotate.

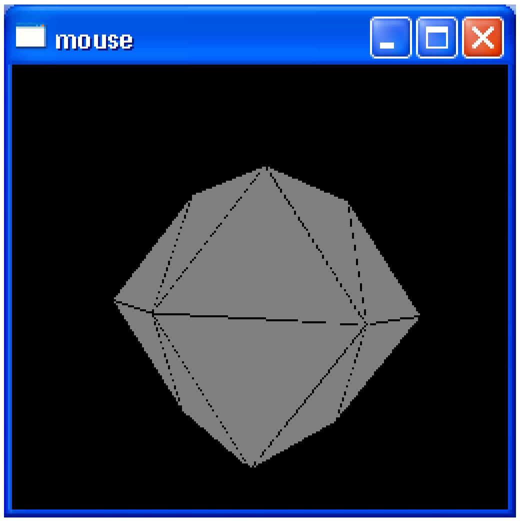
Screenshots:



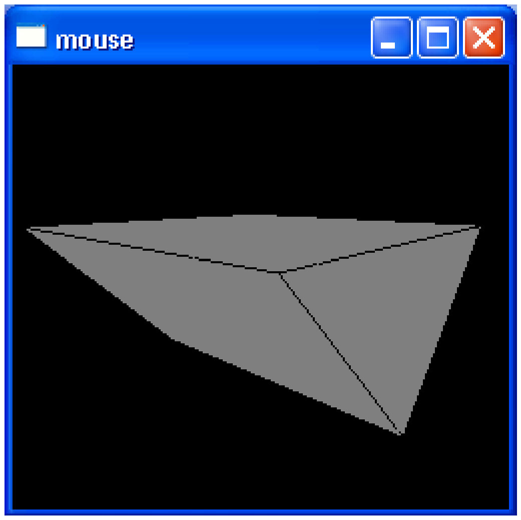
Cube



Colored Triangle



Rough Sphere



Triangular Prism

Conclusions:

We were able to write code to display the mesh of color and we were able to write code that utilizes the mouse in order to rotate and translate the shape. Unfortunately, we were unable to calculate various properties of the meshes themselves. However, we have learned much OpenGL and C++.

Achievements:

- Display 3-D meshes on the screen
- Translation in order to move the mesh around
- Rotation to see all sides of the mesh clearly
- Knowledge!

Acknowledgements:

Steven Schum: For inspiring us and helping us learn the required materials.

Alex Booker: For helping out with C++ and OpenGL.

NM AiSC program and people: For making this all possible.

Bibliography:

http://nehe.gamedev.net/

http://mview.sourceforge.net/

http://www.geuz.org/gmsh/

http://geolab.larc.nasa.gov/Volume/Doc/index.htm

http://www.codeproject.com/opengl/wrl_viewer.asp

Appendix A – Code

```
#include <fstream.h>
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdio.h>


void Readfile(int &nvertex, int &nface, int npface[], int nconn[][4],
float xcoord[], float ycoord[], float zcoord[]);

enum {
    PAN = 1,
    ROTATE,
    ZOOM
};


HDC hDC;
HPALETTE hPalette = 0;
GLfloat trans[3];
GLfloat rot[2];

static void update(int state, int ox, int nx, int oy, int ny)
{
    int dx = ox - nx;
    int dy = ny - oy;

    switch(state) {
    case PAN:
      trans[0] -= dx / 100.0f;
      trans[1] -= dy / 100.0f;
      break;
    case ROTATE:
      rot[0] += (dy * 180.0f) / 500.0f;
      rot[1] -= (dx * 180.0f) / 500.0f;
#define clamp(x) x = x > 360.0f ? x-360.0f : x < -360.0f ? x+=360.0f : x
      clamp(rot[0]);
      clamp(rot[1]);
      break;
    case ZOOM:
      trans[2] -= (dx+dy) / 100.0f;
      break;
    }
}


void
init()
{
    glEnable(GL_DEPTH_TEST);
}

void
reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (float)width/height, 0.001, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -3.0f);
}

void
display()
{
      float xcoord[100];
      float ycoord[100];
      float zcoord[100];
      int   nvertex;
      int   nface;
      int   npface[100];
      int   nconn[100][4];
      int nv, nv1, nv2, jp;

      //Readfile(nvertex, nface, npface, nconn, xcoord, ycoord, zcoord);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef(trans[0], trans[1], trans[2]);
      glTranslatef(0.0f, 0.0f, 1.0f);
    glRotatef(rot[0], 1.0f, 0.0f, 0.0f);
    glRotatef(rot[1], 0.0f, 1.0f, 0.0f);
/*
      glBegin(GL_TRIANGLES);
            glColor3f(1.0f, 0.0f, 0.0f);
            glVertex3f(0.0, 1.0, 0.0);
            glColor3f(0.0f, 1.0f, 0.0f);
            glVertex3f(1.0, -1.0, 0.0);
            glColor3f(0.0f, 0.0f, 1.0f);
            glVertex3f(-1.0, -1.0, 0.0);
      glEnd();

*/


      for(int i=0; i<nface; i++) {
            if(npface[i]==3) {
                  glBegin(GL_TRIANGLES);
                        glColor3f(0.5f, 0.5f, 0.5f);
                        nv=nconn[i][0];                      //  first
vertex of face i
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                        nv=nconn[i][1];                      //  second
vertex of face i
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                        nv=nconn[i][2];                      //  third
vertex of face i
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                  glEnd();
                  glBegin(GL_LINES);
                        glColor3f(0.0f, 0.0f, 0.0f);
                        for(int j=0; j<3; j++) {
                              jp=j+1;
                              if(j==2) jp=0;
                              nv1=nconn[i][j];
                              nv2=nconn[i][jp];
                              glVertex3f(xcoord[nv1], ycoord[nv1],
zcoord[nv1]);
```

```
                              glVertex3f(xcoord[nv2], ycoord[nv2], 
zcoord[nv2]);
                        }
                  glEnd();

            } else if(npface[i]==4) {
                  glBegin(GL_QUADS);
                        glColor3f(0.5f, 0.5f, 0.5f);
                        nv=nconn[i][0];
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                        nv=nconn[i][1];
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                        nv=nconn[i][2];
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                        nv=nconn[i][3];
                        glVertex3f(xcoord[nv], ycoord[nv], zcoord[nv]);
                  glEnd();
                  glBegin(GL_LINES);
                        glColor3f(0.0f, 0.0f, 0.0f);
                        for(int j=0; j<4; j++) {
                              jp=j+1;
                              if(j==3) jp=0;
                              nv1=nconn[i][j];
                              nv2=nconn[i][jp];
                              glVertex3f(xcoord[nv1], ycoord[nv1], 
zcoord[nv1]);
                              glVertex3f(xcoord[nv2], ycoord[nv2], 
zcoord[nv2]);
                        }
                  glEnd();
            }
      }

*/
    glPopMatrix();
    glFlush();
    SwapBuffers(hDC);               /* nop if singlebuffered */
}


LONG WINAPI
WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    static PAINTSTRUCT ps;
    static GLboolean left  = GL_FALSE;    /* left button currently down? 
*/
    static GLboolean right = GL_FALSE;    /* right button currently 
down? */
    static GLuint    state   = 0;   /* mouse state flag */
    static int omx, omy, mx, my;

    switch(uMsg) {
    case WM_PAINT:
      display();
      BeginPaint(hWnd, &ps);
      EndPaint(hWnd, &ps);
      return 0;

    case WM_SIZE:
      reshape(LOWORD(lParam), HIWORD(lParam));
      PostMessage(hWnd, WM_PAINT, 0, 0);
      return 0;

    case WM_CHAR:
```

```
  switch (wParam) {
  case 27:                /* ESC key */
      PostQuitMessage(0);
      break;
  }
  return 0;

case WM_LBUTTONDOWN:
case WM_RBUTTONDOWN:
  /* if we don't set the capture we won't get mouse move
       messages when the mouse moves outside the window. */
  SetCapture(hWnd);
  mx = LOWORD(lParam);
  my = HIWORD(lParam);
  if (uMsg == WM_LBUTTONDOWN)
      state |= PAN;
  if (uMsg == WM_RBUTTONDOWN)
      state |= ROTATE;
  return 0;

case WM_LBUTTONUP:
case WM_RBUTTONUP:
  /* remember to release the capture when we are finished. */
  ReleaseCapture();
  state = 0;
  return 0;

case WM_MOUSEMOVE:
  if (state) {
      omx = mx;
      omy = my;
      mx = LOWORD(lParam);
      my = HIWORD(lParam);
      /* Win32 is pretty braindead about the x, y position that
         it returns when the mouse is off the left or top edge
         of the window (due to them being unsigned). therefore,
         roll the Win32's 0..2^16 pointer co-ord range to the
         more amenable (and useful) 0..+/-2^15. */
      if(mx & 1 << 15) mx -= (1 << 16);
      if(my & 1 << 15) my -= (1 << 16);
      update(state, omx, mx, omy, my);
      PostMessage(hWnd, WM_PAINT, 0, 0);
  }
  return 0;

case WM_PALETTECHANGED:
  if (hWnd == (HWND)wParam)
      break;
  /* fall through to WM_QUERYNEWPALETTE */

case WM_QUERYNEWPALETTE:
  if (hPalette) {
      UnrealizeObject(hPalette);
      SelectPalette(hDC, hPalette, FALSE);
      RealizePalette(hDC);
      return TRUE;
  }
  return FALSE;

case WM_CLOSE:
  PostQuitMessage(0);
  return 0;
}
```

```
    return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

HWND
CreateOpenGLWindow(char* title, int x, int y, int width, int height,
               BYTE type, DWORD flags)
{
    int         n, pf;
    HWND        hWnd;
    WNDCLASS    wc;
    LOGPALETTE* lpPal;
    PIXELFORMATDESCRIPTOR pfd;
    static HINSTANCE hInstance = 0;

    /* only register the window class once - use hInstance as a flag. */
    if (!hInstance) {
      hInstance = GetModuleHandle(NULL);
      wc.style         = CS_OWNDC;
      wc.lpfnWndProc   = (WNDPROC)WindowProc;
      wc.cbClsExtra    = 0;
      wc.cbWndExtra    = 0;
      wc.hInstance     = hInstance;
      wc.hIcon         = LoadIcon(NULL, IDI_WINLOGO);
      wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
      wc.hbrBackground = NULL;
      wc.lpszMenuName  = NULL;
      wc.lpszClassName = "OpenGL";

      if (!RegisterClass(&wc)) {
          MessageBox(NULL, "RegisterClass() failed:  "
                   "Cannot register window class.", "Error", MB_OK);
          return NULL;
      }
    }

    hWnd = CreateWindow("OpenGL", title, WS_OVERLAPPEDWINDOW |
                  WS_CLIPSIBLINGS | WS_CLIPCHILDREN,
                  x, y, width, height, NULL, NULL, hInstance, NULL);

    if (hWnd == NULL) {
      MessageBox(NULL, "CreateWindow() failed:  Cannot create a
window.",
               "Error", MB_OK);
      return NULL;
    }

    hDC = GetDC(hWnd);

    /* there is no guarantee that the contents of the stack that become
       the pfd are zeroed, therefore _make sure_ to clear these bits. */
    memset(&pfd, 0, sizeof(pfd));
    pfd.nSize        = sizeof(pfd);
    pfd.nVersion     = 1;
    pfd.dwFlags      = PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | flags;
    pfd.iPixelType   = type;
    pfd.cDepthBits   = 32;
    pfd.cColorBits   = 32;

    pf = ChoosePixelFormat(hDC, &pfd);
    if (pf == 0) {
      MessageBox(NULL, "ChoosePixelFormat() failed:  "
               "Cannot find a suitable pixel format.", "Error", MB_OK);
      return 0;
    }
    if (SetPixelFormat(hDC, pf, &pfd) == FALSE) {
      MessageBox(NULL, "SetPixelFormat() failed:  "
                 "Cannot set format specified.", "Error", MB_OK);
      return 0;
    }

    DescribePixelFormat(hDC, pf, sizeof(PIXELFORMATDESCRIPTOR), &pfd);

    if (pfd.dwFlags & PFD_NEED_PALETTE ||
      pfd.iPixelType == PFD_TYPE_COLORINDEX) {

      n = 1 << pfd.cColorBits;
      if (n > 256) n = 256;

      lpPal = (LOGPALETTE*)malloc(sizeof(LOGPALETTE) +
                             sizeof(PALETTEENTRY) * n);
      memset(lpPal, 0, sizeof(LOGPALETTE) + sizeof(PALETTEENTRY) * n);
      lpPal->palVersion = 0x300;
      lpPal->palNumEntries = n;

      GetSystemPaletteEntries(hDC, 0, n, &lpPal->palPalEntry[0]);

      /* if the pixel type is RGBA, then we want to make an RGB ramp,
         otherwise (color index) set individual colors. */
      if (pfd.iPixelType == PFD_TYPE_RGBA) {
          int redMask = (1 << pfd.cRedBits) - 1;
          int greenMask = (1 << pfd.cGreenBits) - 1;
          int blueMask = (1 << pfd.cBlueBits) - 1;
          int i;

          /* fill in the entries with an RGB color ramp. */
          for (i = 0; i < n; ++i) {
            lpPal->palPalEntry[i].peRed =
                (((i >> pfd.cRedShift)   & redMask)   * 255) / redMask;
            lpPal->palPalEntry[i].peGreen =
                (((i >> pfd.cGreenShift) & greenMask) * 255) /
greenMask;
            lpPal->palPalEntry[i].peBlue =
                (((i >> pfd.cBlueShift)  & blueMask)  * 255) / blueMask;
            lpPal->palPalEntry[i].peFlags = 0;
          }
      } else {
          lpPal->palPalEntry[0].peRed = 0;
          lpPal->palPalEntry[0].peGreen = 0;
          lpPal->palPalEntry[0].peBlue = 0;
          lpPal->palPalEntry[0].peFlags = PC_NOCOLLAPSE;
          lpPal->palPalEntry[1].peRed = 255;
          lpPal->palPalEntry[1].peGreen = 0;
          lpPal->palPalEntry[1].peBlue = 0;
          lpPal->palPalEntry[1].peFlags = PC_NOCOLLAPSE;
          lpPal->palPalEntry[2].peRed = 0;
          lpPal->palPalEntry[2].peGreen = 255;
          lpPal->palPalEntry[2].peBlue = 0;
          lpPal->palPalEntry[2].peFlags = PC_NOCOLLAPSE;
          lpPal->palPalEntry[3].peRed = 0;
          lpPal->palPalEntry[3].peGreen = 0;
          lpPal->palPalEntry[3].peBlue = 255;
          lpPal->palPalEntry[3].peFlags = PC_NOCOLLAPSE;
      }

      hPalette = CreatePalette(lpPal);
      if (hPalette) {
          SelectPalette(hDC, hPalette, FALSE);
```

```
        RealizePalette(hDC);
      }

      free(lpPal);
    }

    ReleaseDC(hWnd, hDC);

    return hWnd;
}

int WINAPI WinMain(HINSTANCE hCurrentInst, HINSTANCE hPreviousInst,
      LPSTR lpszCmdLine, int nCmdShow)
{
    HGLRC hRC;                        /* opengl context */
    HWND  hWnd;                       /* window */
    MSG   msg;                        /* message */
    DWORD buffer = PFD_DOUBLEBUFFER;      /* buffering type */
    BYTE  color  = PFD_TYPE_RGBA;     /* color type */

    if (strstr(lpszCmdLine, "-sb")) {
      buffer = 0;
    }
    if (strstr(lpszCmdLine, "-ci")) {
      color = PFD_TYPE_COLORINDEX;
    }
    if (strstr(lpszCmdLine, "-h")) {
      MessageBox(NULL, "mouse [-ci] [-sb]\n"
               "  -sb   single buffered\n"
               "  -ci   color index\n",
               "Usage help", MB_ICONINFORMATION);
      exit(0);
    }

    hWnd = CreateOpenGLWindow("mouse", 0, 0, 256, 256, color, buffer);
    if (hWnd == NULL)
      exit(1);

    hDC = GetDC(hWnd);
    hRC = wglCreateContext(hDC);
    wglMakeCurrent(hDC, hRC);

    init();

    ShowWindow(hWnd, nCmdShow);

    while(GetMessage(&msg, hWnd, 0, 0)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    wglMakeCurrent(NULL, NULL);
    ReleaseDC(hWnd, hDC);
    wglDeleteContext(hRC);
    DestroyWindow(hWnd);
    if (hPalette)
      DeleteObject(hPalette);

    return msg.wParam;
}


void Readfile(int &nvertex, int &nface, int npface[], int nconn[][4],
float xcoord[], float ycoord[], float zcoord[])
```

```
{
      ifstream fin;
      fin.open("input.txt");

      fin >> nvertex;
      for(int i=0; i<nvertex; i++)
            fin >> xcoord[i] >> ycoord[i] >> zcoord[i];

      fin >> nface;
      for(int j=0; j<nface; j++) {
            fin >> npface[j];
            if(npface[j]==3)
                  fin >> nconn[j][0] >> nconn[j][1] >> nconn[j][2];
            else if(npface[j]==4)
                  fin >> nconn[j][0] >> nconn[j][1] >> nconn[j][2] >>
nconn[j][3];
      }
}
```