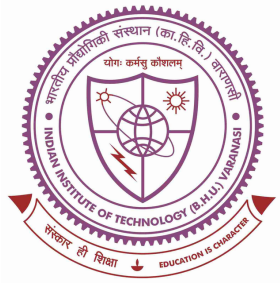
INDIAN INSTITUTE OF TECHNOLOGY
VARANASI



EXPLORATORY PROJECT

SUBMITTED TO
DEPARTMENT OF MINING ENGINEERING

SESSION- **2018-19**

UNDER THE GUIDANCE OF:
DR. ASHOK JAISWAL
DEPT. OF MINING ENGINEERING

SUBMITTED BY:
MOBASSHIR ALI
ROLL NO.- 17155049
MINING ENGINEERING

ACKNOWLEDGEMENT

In the accomplishment of this project successfully, many people have best owned upon me their blessings and the heart pledged support, this time I am utilizing to thank all the people who have been concerned with this project

I would like to express my deep gratitude to **Dr. Ashok Jaiswal**, my research supervisors, for their patient guidance, enthusiastic encouragement and useful critiques of this research work.

My grateful thanks are also extended to my friends of my project group for their help in keeping my progress on schedule.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

Mobasshir Ali
Dept. of Mining Engineering

CERTIFICATE

This is to certify that **Mobasshir Ali**, a student of Department of Mining Engineering- Batch 2017, has successfully completed the exploratory project "TO MAKE A PLUGIN FOR AUTOCAD AND GENERATING COAL SEAM MODEL FOR THE GIVEN BOREHOLE LOG DATA" under the guidance of **Dr. Ashok Jaiswal** during the session 2018- 2019.

Dr. Ashok Jaiswal
[Supervisor]

Prof. S.K. Sharma
[Head of department]

INTRODUCTION

Mining Simulation Softwares:-
Mining simulation is the computer-based modeling of a real open-pit or underground mining.

Mining simulation (Mine Modelling) offers a way forward, providing mining output statistics and dynamic views of operations for analysis, optimization, and experimentation, all without operational interruption.

These softwares provides the mining industry with the most advanced 3D geological modelling, mine design and production planning solutions.

Examples of softwares are:

1. Surpac
2. Surfer
3. Vulcan
4. Visual Land pro 2000
5. Simio

TITLE

Plugin Development for AutoCAD

In this project , we will be working with the AutoCAD .NET Application Programming Interface (API) and the C# programming language to create a ‘plug-in’ – a module that loads into AutoCAD to extend its functionality.

OBJECTIVE

"TO MAKE A PLUGIN FOR AUTOCAD AND GENERATING COAL SEAM MODEL FOR THE GIVEN BORE HOLE LOG DATA"

THEORY

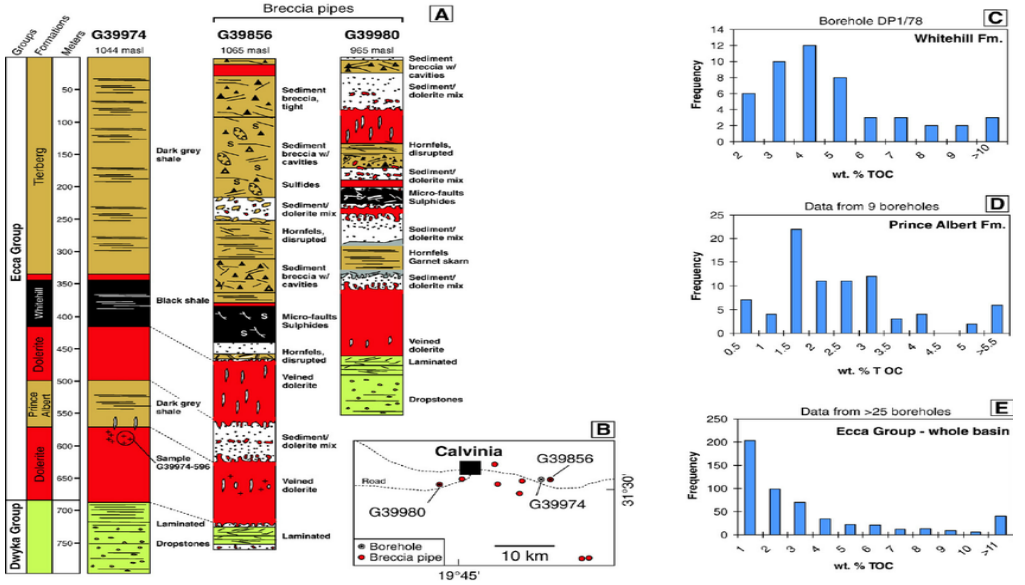
Definition of coal seam
: a bed of coal usually thick enough to be profitably mined

A coal seam is a dark brown or black banded deposit of coalthat is visible within layers of rock. These seams are located underground and can be mined using either deep mining or strip mining techniques depending on their proximity to the surface. These seams undergo normal coal formation and serve as a conventional coal resource. The reserves of coal are immense, and are the largest of all of the fossil fuels

Borehole
Borehole logging is the practice of making a detailed record (a well log) of the geologic formations penetrated by a borehole. The log may be based either on visual inspection of samples brought to the surface (geological logs) or on physical measurements made by instruments lowered into the hole (geophysical logs). Some types of geophysical well logs can be done during any phase of a well's history: drilling, completing, producing, or abandoning.

Borehole data:
The interpretations stored in the Borehole Geology database are made from borehole logs that show the geology encountered at depth within each borehole. In many cases, these logs were created by the geotechnical companies responsible for drilling the holes, and supplied to the BGS

In other cases, the logs may have been made by our own geologists, either at the time of drilling, or subsequently from core samples



PROCEDURE

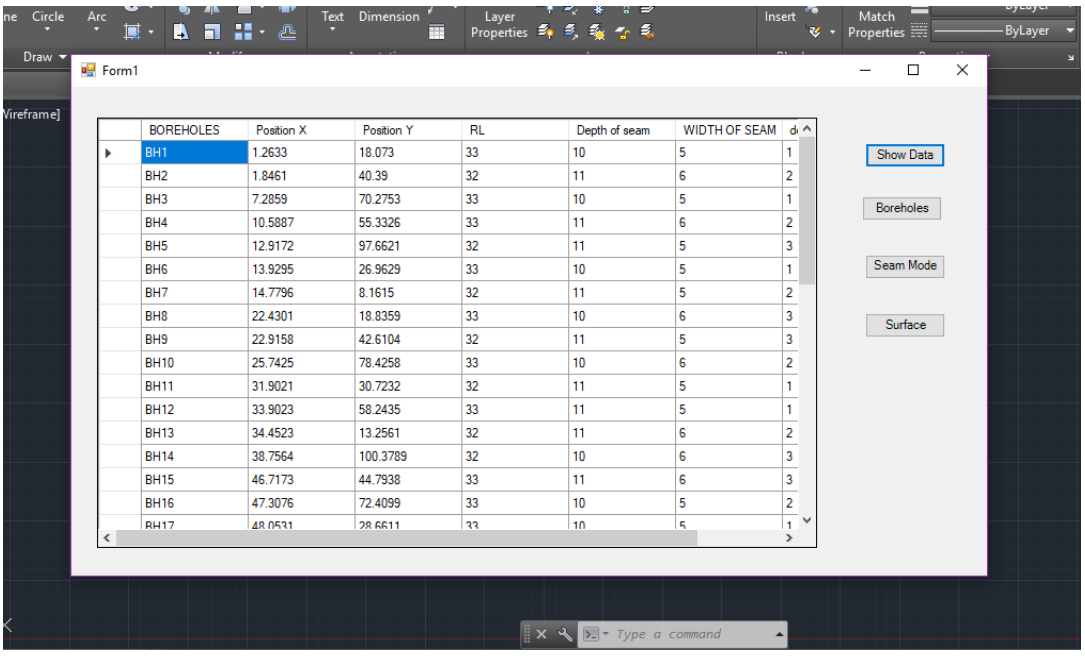
1. Getting familiar with AutoCAD and to draw simple geometries like lines, circles.

2. In Visual Basic, Creation of Plug-in by adding refrence files of AutoCAD.
   Creation of AutoCAD plugin command.

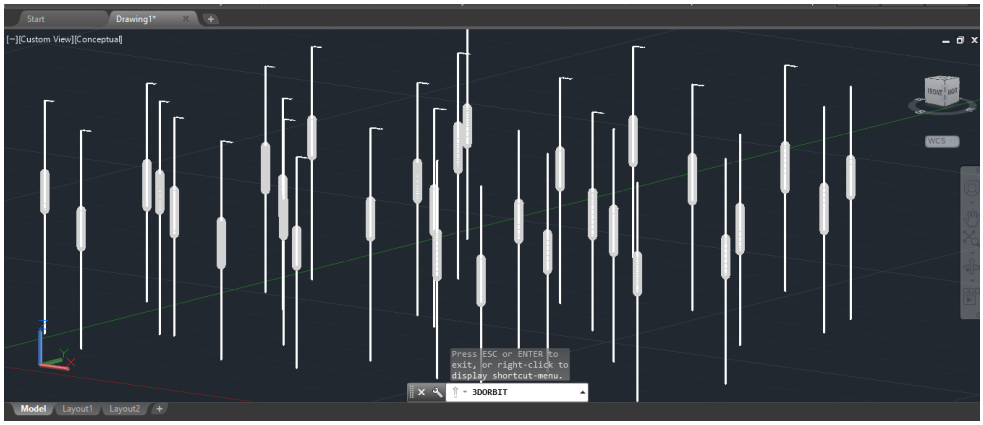
3. Code to open a window form which ask user to give Bore hole data.
   The window form must also print data so that user must verify it.

4. The provided borehole data must be in .csv format.

6. Two buttons are placed on the window form. One for showing the Bore holes and other to give 3D coal seam model.



7. To draw the boreholes lines, line are drawn from the ground point (R.L) to the bottom point of the borehole.

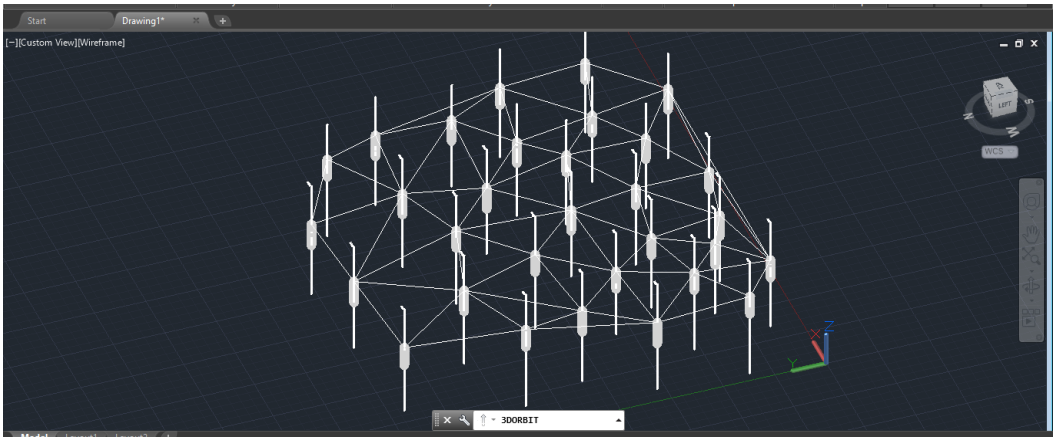


8. To draw the coal seam model:-

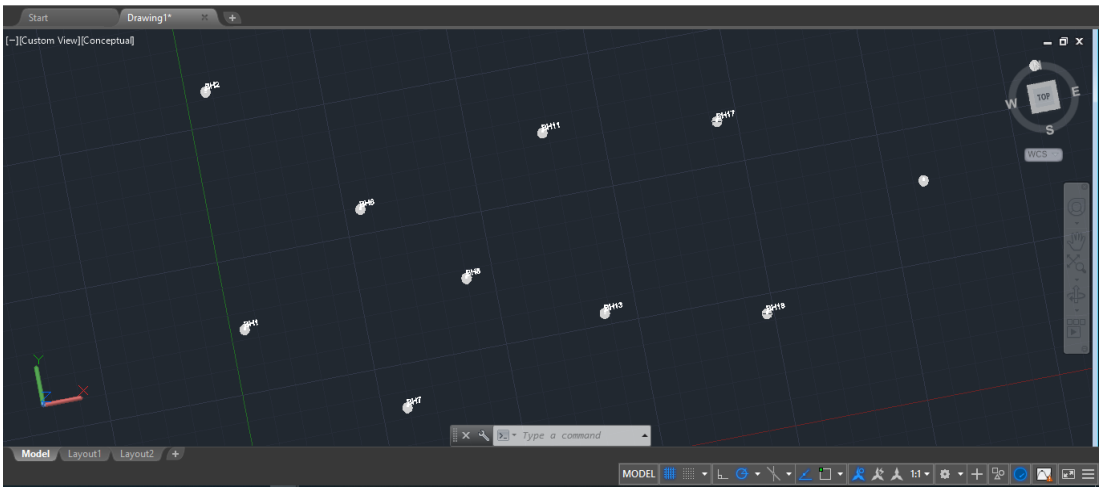
a). The points on upper layer of the coal seam are located.

b). Contouring is done by joining all the points by Triangulation method.

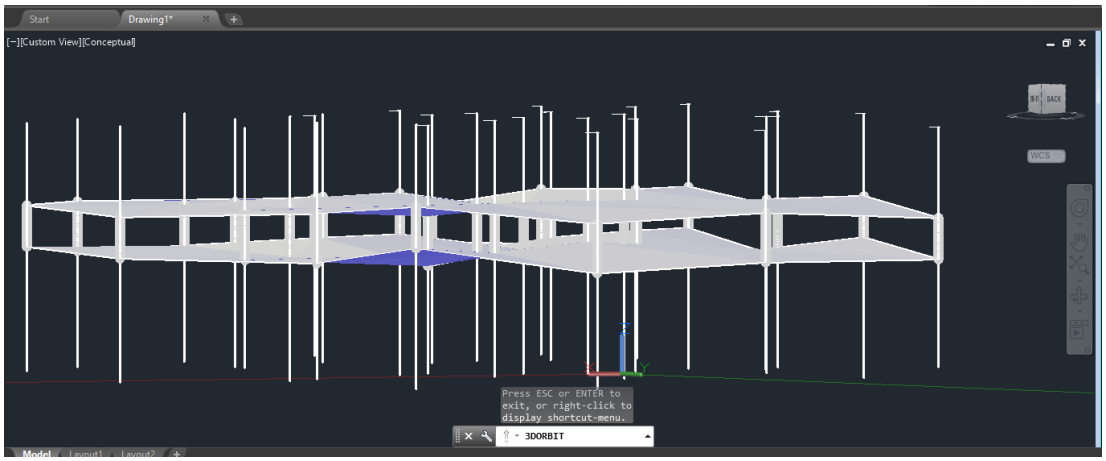
c). Upper layer is created and same is done with the lower layer of coal seam.



9. Additional features are added like giving labels to the boreholes, contouring the ground surface etc.



10. Finally, the plugin is run in AutoCAD and the result are seen in 'Realistic' mode

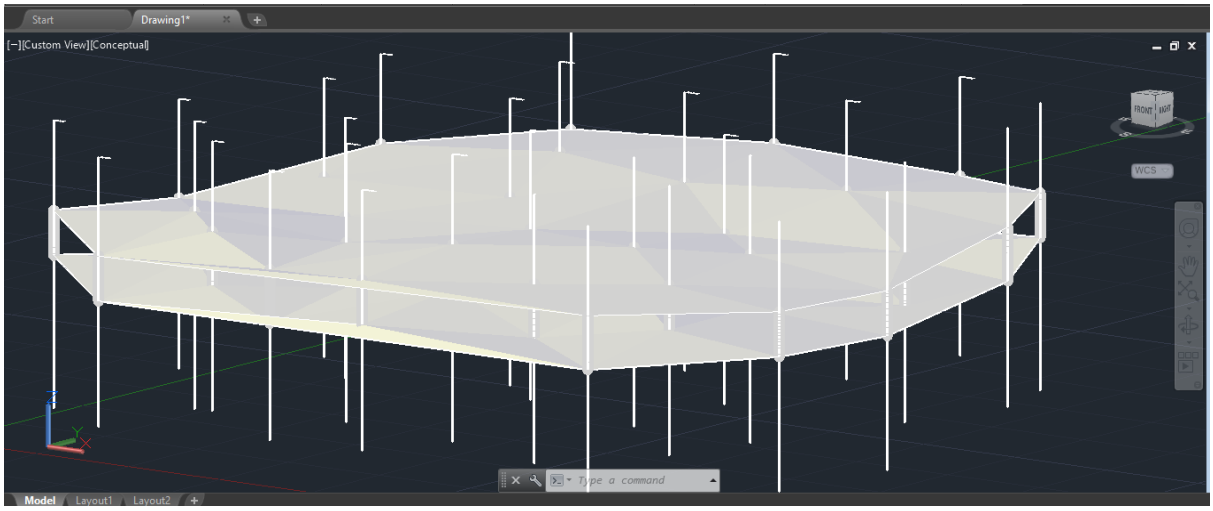


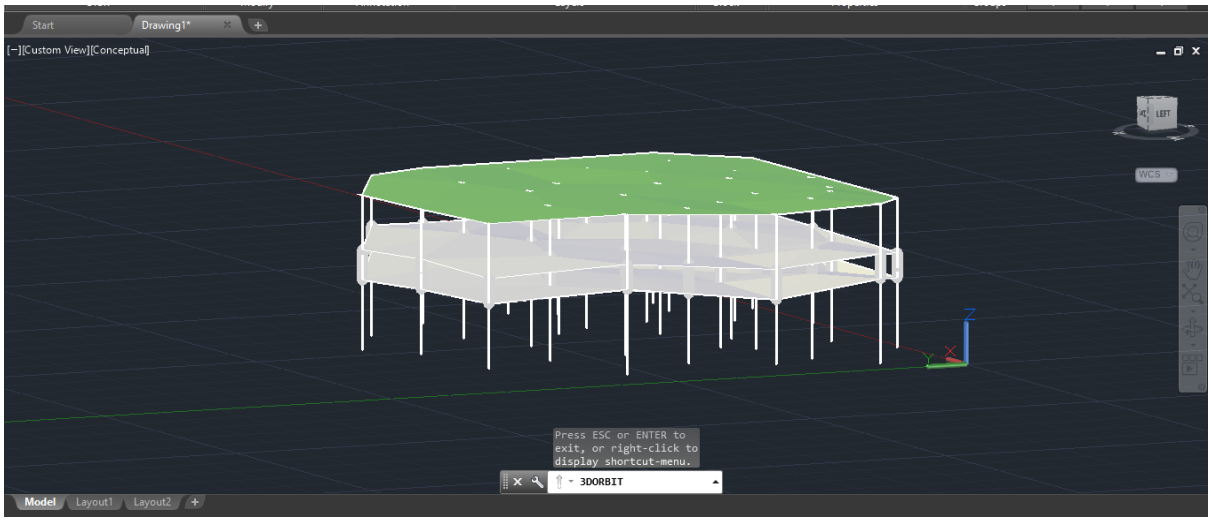
RESULT

The Plugin for the AutoCAD is completed which allows all to assemble and view borehole data and accurtely model coal seam and deposits.

EXAMPLES

Below are some screenshots for the final view of coal seam model:





ISSUES AND FUTURE CONCERNS

1. The space between the upper and lower surface of the coal seam couldn't be properly filled/surfaced. Therefore proper meshing between the layers are required.

2. It is still to be tested on original borehole data that contains data of other ores also.

3. The model is to be made more informative by showing depths and widths of the seam.

REFRENCES

https://knowledge.autodesk.com

Triangulation

https://through-the-interface.typepad.com/through_the_interface/2009/04/triangulating-an-autocad-polyface-mesh-from-a-set-of-points-using-net.html

CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms;
using System.Data.OleDb;
using System.IO;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Geometry;
using Autodesk.AutoCAD.EditorInput;

namespace PluginCsv
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
      if (openFileDialog1.ShowDialog() == DialogResult.OK)
      {
        string path;
        path = openFileDialog1.FileName;
        string ext = Path.GetExtension(path);
        if (ext == ".csv")
        {
          System.Data.DataTable my_data = new System.Data.DataTable();
          string[] raw_text = File.ReadAllLines(path);
          string[] data_col = null;
          int x = 0;

          foreach (string test_line in raw_text)
          {
            //MessageBox.Show(test_line);
            data_col = test_line.Split(',');
            if (x == 0)
            {
              //header
              for (int i1 = 0; i1 < data_col.Count(); i1++)
              {
                my_data.Columns.Add(data_col[i1]);
                //dataGridView1.Columns.Add(data_col[i1]);
              }
              x++;
            }
            else
            {
              //data
              my_data.Rows.Add(data_col);
            }

          }

          dataGridView1.DataSource = my_data;
          int a, i;
          a = dataGridView1.Columns.Count;
          for (i = 0; i < a; i++)
          {
            dataGridView1.Columns[i].SortMode =
DataGridViewColumnSortMode.NotSortable;

          }

        }
      }
    }
```

```
 private void button2_Click(object sender, EventArgs e)
    {
       this.drawpointsupper();
       this.domeshupper();
       this.addlayer1();
       this.drawpointslower();
       this.domeshlower();
    }
    void drawpointsupper()
    {
       Document acDoc =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocu
ment;
       Autodesk.AutoCAD.DatabaseServices.Database acCurDb = acDoc.Database;
       Database db = acDoc.Database;
       // Start a transaction
       using (DocumentLock docLock = acDoc.LockDocument())
       {
          for (int i = 0; i <= dataGridView1.Rows.Count-2; i++)
          {
             using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
             {
                LayerTable ltb = (LayerTable)acTrans.GetObject(db.LayerTableId,
                                          OpenMode.ForRead);
                db.Clayer = ltb["0"];
                BlockTable acBlkTbl;
                acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                OpenMode.ForRead) as BlockTable;
                BlockTableRecord acBlkTblRec;
                acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                OpenMode.ForWrite) as BlockTableRecord;
                using (DBPoint acPoint = new DBPoint(new
Point3d(Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[2].Value),
(Convert.ToDouble(dataGridView1.Rows[i].Cells[3].Value) -
Convert.ToDouble(dataGridView1.Rows[i].Cells[4].Value)))))
                {
                   // Add the new object to the block table record and the transaction
                   acBlkTblRec.AppendEntity(acPoint);
                   acTrans.AddNewlyCreatedDBObject(acPoint, true);
                }
                /*acCurDb.Pdmode = 34;
                acCurDb.Pdsize = 1;*/
                acTrans.Commit();
             }
          }
       }
    }
    void drawpointslower()...
    void addlayer1()

    {

       Document doc =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocu
ment;

       Database db = doc.Database;

       Editor ed = doc.Editor;

       using (DocumentLock docLock = doc.LockDocument())
       {

          using (Transaction tr = db.TransactionManager.StartTransaction())

          {

             LayerTable ltb = (LayerTable)tr.GetObject(db.LayerTableId,

                                       OpenMode.ForRead);

             //create a new layout.

             if (!ltb.Has("NewLayer"))

             {

                ltb.UpgradeOpen();

                LayerTableRecord newLayer = new LayerTableRecord();

                newLayer.Name = "1";
```

```
            newLayer.LineWeight = LineWeight.LineWeight005;
            newLayer.Description = "This is new layer";

            //red color
            /*newLayer.Color =
                  Autodesk.AutoCAD.Colors.Color.FromRgb(255, 0, 0);
                  */

            ltb.Add(newLayer);
            tr.AddNewlyCreatedDBObject(newLayer, true);
          }

          tr.Commit();
          //make it as current
          db.Clayer = ltb["1"];
        }
      }
    }
    public bool circum(
double x1, double y1, double x2,
double y2, double x3, double y3,
ref double xc, ref double yc, ref double r)
    {
      // Calculation of circumscribed circle coordinates and
      // squared radius

      const double eps = 1e-6;
      const double big = 1e12;
      bool result = true;
      double m1, m2, mx1, mx2, my1, my2, dx, dy;

      if ((Math.Abs(y1 - y2) < eps) && (Math.Abs(y2 - y3) < eps))
      {
        result = false;
        xc = x1; yc = y1; r = big;
      }
      else
      {
        if (Math.Abs(y2 - y1) < eps)
        {
          m2 = -(x3 - x2) / (y3 - y2);
          mx2 = (x2 + x3) / 2;
          my2 = (y2 + y3) / 2;
          xc = (x2 + x1) / 2;
          yc = m2 * (xc - mx2) + my2;
        }
```

```
else if (Math.Abs(y3 - y2) < eps)
        {
           m1 = -(x2 - x1) / (y2 - y1);
           mx1 = (x1 + x2) / 2;
           my1 = (y1 + y2) / 2;
           xc = (x3 + x2) / 2;
           yc = m1 * (xc - mx1) + my1;
        }
        else
        {
           m1 = -(x2 - x1) / (y2 - y1);
           m2 = -(x3 - x2) / (y3 - y2);
           if (Math.Abs(m1 - m2) < eps)
           {
              result = false;
              xc = x1;
              yc = y1;
              r = big;
           }
           else
           {
              mx1 = (x1 + x2) / 2;
              mx2 = (x2 + x3) / 2;
              my1 = (y1 + y2) / 2;
              my2 = (y2 + y3) / 2;
              xc = (m1 * mx1 - m2 * mx2 + my2 - my1) / (m1 - m2);
              yc = m1 * (xc - mx1) + my1;
           }
        }
     }
     dx = x2 - xc;
     dy = y2 - yc;
     r = dx * dx + dy * dy;
     return result;
  }
```

```
 void domeshupper()
     {
        const int maxpoints = 32767;


        Document doc =

 Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocu
ment;
        Database db = doc.Database;
        Editor ed = doc.Editor;
        TypedValue[] tvs = new TypedValue[2];
        tvs.SetValue(new TypedValue((int)DxfCode.LayerName, "0"), 0);
        tvs.SetValue(new TypedValue((int)DxfCode.Start, "POINT"), 1);
        SelectionFilter sf =
         new SelectionFilter(tvs);
        PromptSelectionResult psr = ed.SelectAll(sf);

        if (psr.Status == PromptStatus.Error) return;
        if (psr.Status == PromptStatus.Cancel) return;

        SelectionSet ss = psr.Value;
        int npts = ss.Count;
        if (npts < 3)
        {
           ed.WriteMessage("Minimum 3 points must be selected!");
           return;
        }
        if (npts > maxpoints)
        {
           ed.WriteMessage("Maximum nuber of points exceeded!");
           return;
        }

        int i, j, k, ntri, ned, status1 = 0, status2 = 0;
        bool status;

        // Point coordinates

        double[] ptx = new double[maxpoints + 3];
        double[] pty = new double[maxpoints + 3];
        double[] ptz = new double[maxpoints + 3];

        // Triangle definitions

        int[] pt1 = new int[maxpoints * 2 + 1];
        int[] pt2 = new int[maxpoints * 2 + 1];
        int[] pt3 = new int[maxpoints * 2 + 1];
```

```
// Circumscribed circle

    double[] cex = new double[maxpoints * 2 + 1];
    double[] cey = new double[maxpoints * 2 + 1];
    double[] rad = new double[maxpoints * 2 + 1];
    double xmin, ymin, xmax, ymax, dx, dy, xmid, ymid;
    int[] ed1 = new int[maxpoints * 2 + 1];
    int[] ed2 = new int[maxpoints * 2 + 1];

    ObjectId[] idarray = ss.GetObjectIds();
    Transaction tr =
      db.TransactionManager.StartTransaction();
    using (DocumentLock docLock = doc.LockDocument())
    {
      using (tr)
      {
        DBPoint ent;
        k = 0;
        for (i = 0; i < npts; i++)
        {
          ent =
            (DBPoint)tr.GetObject(idarray[k], OpenMode.ForRead, false);
          ptx[i] = ent.Position[0];
          pty[i] = ent.Position[1];
          ptz[i] = ent.Position[2];
          for (j = 0; j < i; j++)
            if ((ptx[i] == ptx[j]) && (pty[i] == pty[j]))
            {
              i--; npts--; status2++;
            }
          k++;
        }
        tr.Commit();
      }
    }

    if (status2 > 0)
      ed.WriteMessage(
        "\nIgnored {0} point(s) with same coordinates.",
        status2
      );
```

```
// Circumscribed circle

    double[] cex = new double[maxpoints * 2 + 1];
    double[] cey = new double[maxpoints * 2 + 1];
    double[] rad = new double[maxpoints * 2 + 1];
    double xmin, ymin, xmax, ymax, dx, dy, xmid, ymid;
    int[] ed1 = new int[maxpoints * 2 + 1];
    int[] ed2 = new int[maxpoints * 2 + 1];

    ObjectId[] idarray = ss.GetObjectIds();
    Transaction tr =
      db.TransactionManager.StartTransaction();
    using (DocumentLock docLock = doc.LockDocument())
    {
      using (tr)
      {
        DBPoint ent;
        k = 0;
        for (i = 0; i < npts; i++)
        {
          ent =
            (DBPoint)tr.GetObject(idarray[k], OpenMode.ForRead, false);
          ptx[i] = ent.Position[0];
          pty[i] = ent.Position[1];
          ptz[i] = ent.Position[2];
          for (j = 0; j < i; j++)
            if ((ptx[i] == ptx[j]) && (pty[i] == pty[j]))
            {
              i--; npts--; status2++;
            }
          k++;
        }
        tr.Commit();
      }
    }

    if (status2 > 0)
      ed.WriteMessage(
        "\nIgnored {0} point(s) with same coordinates.",
        status2
      );
```

```
// main loop
for (i = 0; i < npts; i++)
{
   ned = 0;
   xmin = ptx[i]; ymin = pty[i];
   j = 0;
   while (j < ntri)
   {
      dx = cex[j] - xmin; dy = cey[j] - ymin;
      if (((dx * dx) + (dy * dy)) < rad[j])
      {
         ed1[ned] = pt1[j]; ed2[ned] = pt2[j];
         ned++;
         ed1[ned] = pt2[j]; ed2[ned] = pt3[j];
         ned++;
         ed1[ned] = pt3[j]; ed2[ned] = pt1[j];
         ned++;
         ntri--;
         pt1[j] = pt1[ntri];
         pt2[j] = pt2[ntri];
         pt3[j] = pt3[ntri];
         cex[j] = cex[ntri];
         cey[j] = cey[ntri];
         rad[j] = rad[ntri];
         j--;
      }
      j++;
   }

   for (j = 0; j < ned - 1; j++)
      for (k = j + 1; k < ned; k++)
         if ((ed1[j] == ed2[k]) && (ed2[j] == ed1[k]))
         {
            ed1[j] = -1; ed2[j] = -1; ed1[k] = -1; ed2[k] = -1;
         }

   for (j = 0; j < ned; j++)
      if ((ed1[j] >= 0) && (ed2[j] >= 0))
      {
         pt1[ntri] = ed1[j]; pt2[ntri] = ed2[j]; pt3[ntri] = i;
         status =
          circum(
            ptx[pt1[ntri]], pty[pt1[ntri]], ptx[pt2[ntri]],
            pty[pt2[ntri]], ptx[pt3[ntri]], pty[pt3[ntri]],
            ref cex[ntri], ref cey[ntri], ref rad[ntri]
          );
         if (!status)
         {
            status1++;
         }
         ntri++;
      }
}
```

```
// removal of outer triangles
    i = 0;
    while (i < ntri)
    {
      if ((pt1[i] >= npts) || (pt2[i] >= npts) || (pt3[i] >= npts))
      {
        ntri--;
        pt1[i] = pt1[ntri];
        pt2[i] = pt2[ntri];
        pt3[i] = pt3[ntri];
        cex[i] = cex[ntri];
        cey[i] = cey[ntri];
        rad[i] = rad[ntri];
        i--;
      }
      i++;
    }

    tr = db.TransactionManager.StartTransaction();
    using (DocumentLock docLock = doc.LockDocument())
    {

      using (tr)
      {
        BlockTable bt =
         (BlockTable)tr.GetObject(
           db.BlockTableId,
           OpenMode.ForRead,
           false
         );
        BlockTableRecord btr =
         (BlockTableRecord)tr.GetObject(
           bt[BlockTableRecord.ModelSpace],
           OpenMode.ForWrite,
           false
         );

        PolyFaceMesh pfm = new PolyFaceMesh();
        btr.AppendEntity(pfm);
        tr.AddNewlyCreatedDBObject(pfm, true);
        for (i = 0; i < npts; i++)
        {
          PolyFaceMeshVertex vert =
           new PolyFaceMeshVertex(
             new Point3d(ptx[i], pty[i], ptz[i])
           );
          pfm.AppendVertex(vert);
          tr.AddNewlyCreatedDBObject(vert, true);
        }
```

```
            for (i = 0; i < ntri; i++)
            {
               FaceRecord face =
                new FaceRecord(
                 (short)(pt1[i] + 1),
                 (short)(pt2[i] + 1),
                 (short)(pt3[i] + 1),
                 0
                );
               pfm.AppendFaceRecord(face);
               tr.AddNewlyCreatedDBObject(face, true);
            }
            tr.Commit();
         }
      }
      if (status1 > 0)
         ed.WriteMessage(
          "\nWarning! {0} thin triangle(s) found!" +
          " Wrong result possible!",
          status1
         );
      Autodesk.AutoCAD.ApplicationServices.Application.UpdateScreen();
   }
void domeshlower()...

    private void button3_Click(object sender, EventArgs e)
    {
       this.addlayer2();
       this.surfacelabel();
       this.boreline();
       this.seamline();
    }
    void addlayer2()...
    void surfacelabel()
    {
       Document acDoc =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocu
ment;
       Autodesk.AutoCAD.DatabaseServices.Database acCurDb = acDoc.Database;
       // Start a transaction
       using (DocumentLock docLock = acDoc.LockDocument())
       {
          for (int i = 0; i <= dataGridView1.Rows.Count - 2; i++)
          {
             using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
             {
                BlockTable acBlkTbl;
                acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                OpenMode.ForRead) as BlockTable;
                BlockTableRecord acBlkTblRec;
                acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                OpenMode.ForWrite) as BlockTableRecord;
                using (DBText acText = new DBText())
                {
                   acText.Position = new
Point3d(Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[2].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[3].Value));
                   acText.Height = 1.5;
                   acText.TextString =
Convert.ToString(dataGridView1.Rows[i].Cells[0].Value);
                   acBlkTblRec.AppendEntity(acText);
                   acTrans.AddNewlyCreatedDBObject(acText, true);
                }
                acTrans.Commit();
             }
          }
       }
    }
```

```
 void boreline()
    {
       Document acDoc =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocu
ment;
       Autodesk.AutoCAD.DatabaseServices.Database acCurDb = acDoc.Database;
       Database db = acDoc.Database;
       // Start a transaction
       using (DocumentLock docLock = acDoc.LockDocument())
       {
          db.LineWeightDisplay = true;
          for (int i = 0; i <= dataGridView1.Rows.Count - 2; i++)
          {
             using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
             {
                BlockTable acBlkTbl;
                acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                OpenMode.ForRead) as BlockTable;
                BlockTableRecord acBlkTblRec;
                acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                OpenMode.ForWrite) as BlockTableRecord;
                Line acLine = new Line(new
Point3d(Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[2].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[3].Value)), new
Point3d(Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[2].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[6].Value)));
                acLine.LineWeight = LineWeight.LineWeight030;
                acBlkTblRec.AppendEntity(acLine);
                   acTrans.AddNewlyCreatedDBObject(acLine, true);
                acTrans.Commit();

             }

          }
       }
    }
    void seamline()...

    private void button4_Click(object sender, EventArgs e)
    {
       this.addlayer3();
       this.surfacepoints();
       this.dosurfacemesh();
    }
    void surfacepoints()
    {
       Document acDoc =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager.MdiActiveDocu
ment;
       Autodesk.AutoCAD.DatabaseServices.Database acCurDb = acDoc.Database;
       // Start a transaction
       using (DocumentLock docLock = acDoc.LockDocument())
       {
          for (int i = 0; i <= dataGridView1.Rows.Count - 2; i++)
          {
             using (Transaction acTrans = acCurDb.TransactionManager.StartTransaction())
             {
                BlockTable acBlkTbl;
                acBlkTbl = acTrans.GetObject(acCurDb.BlockTableId,
                OpenMode.ForRead) as BlockTable;
                BlockTableRecord acBlkTblRec;
                acBlkTblRec = acTrans.GetObject(acBlkTbl[BlockTableRecord.ModelSpace],
                OpenMode.ForWrite) as BlockTableRecord;
                using (DBPoint acPoint = new DBPoint(new
Point3d(Convert.ToDouble(dataGridView1.Rows[i].Cells[1].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[2].Value),
Convert.ToDouble(dataGridView1.Rows[i].Cells[3].Value))))
                {
                   acPoint.Color = Autodesk.AutoCAD.Colors.Color.FromRgb(0, 0, 255);
                   // Add the new object to the block table record and the transaction
                   acBlkTblRec.AppendEntity(acPoint);
                   acTrans.AddNewlyCreatedDBObject(acPoint, true);
                }
                /*acCurDb.Pdmode = 34;
                acCurDb.Pdsize = 1;*/
                acTrans.Commit();
             }
          }
       }
    }
    void dosurfacemesh()

    void addlayer3()...
 }
}
```