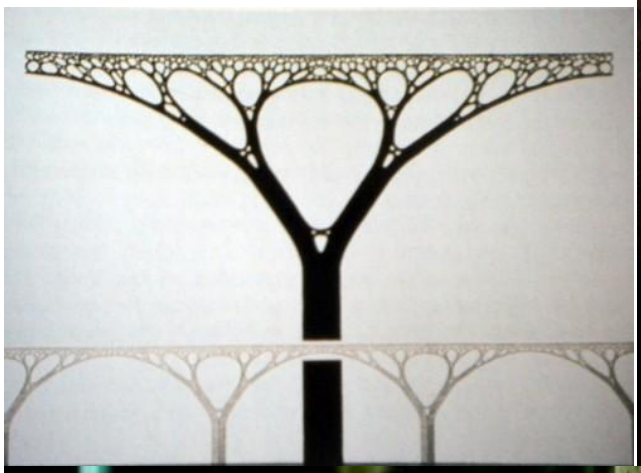
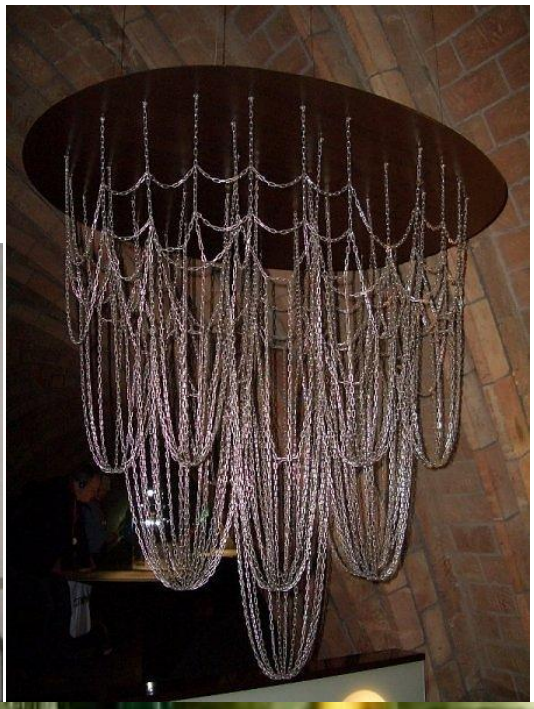
Geodetická křivka na MESH ploše pomocí C#

Eduard Sojka

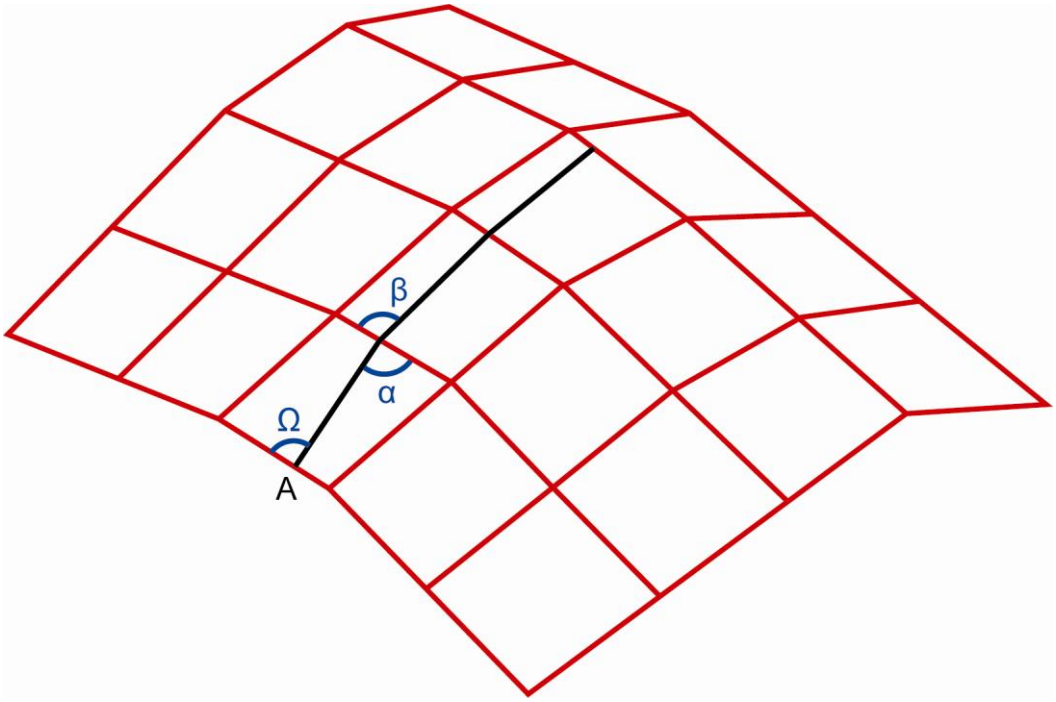
Geodetická křivka na ploše je taková křivka, která spojuje 2 body na této ploše nejkratší možnou cestou. Taková křivka má zajímavou vlastnost: pokud její "mateřskou" plochu narovnáme do roviny, pak tato křivka se rozvine na přímku. V praxi to znamená, že můžeme pomocí zohýbaných latí vytvořit síť, která vytváří požadovaný tvar. Na následujících obrázcích vidíme několik příkladů využití v architektuře.

V Rhinu můžeme pomocí příkazu ShortPath vytvořit geodetickou křivku na NURBS plochách. V praxi je ale někdy výhodné pracovat s MESH plochami, neboť to je formát, který bývá výsledkem fyzikálních simulací při hledání optimálního tvaru. Jako příklad takových simulací můžeme uvést metodu zavěšovaných řetězů používanou Frei Ottem a Antoni Gaudím.





Pro tvorbu geodetických křivek na ploše použijeme metodu blízké metodě konečných prvků. Vstupní instance programu jsou MESH plocha, počáteční bod křivky a úhel pod kterým má křivka na ploše “vybíhat”.



Bod A je bod zadaný uživatelem. Od něj budeme geodetickou křivku dále počítat. Pokud Bod A umístíme do uzlového bodu(bude se nacházet v průsečíku tvořících=červených úseček) program nebude pracovat.

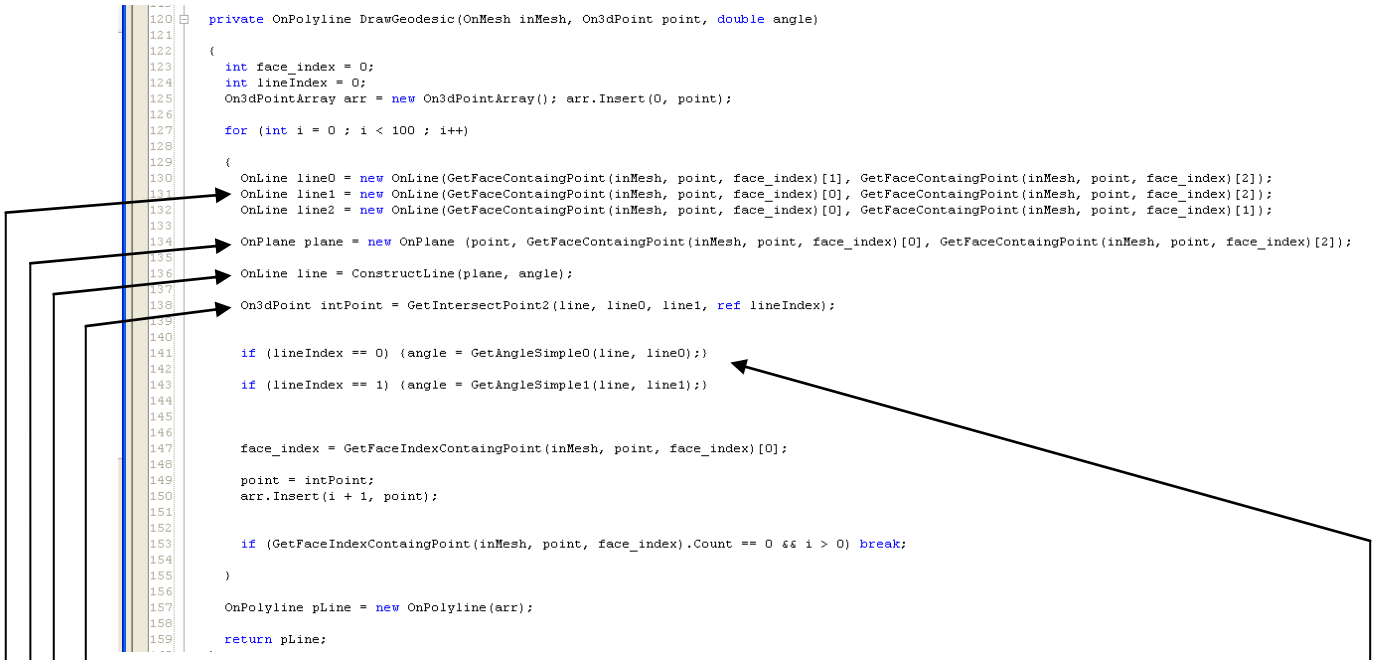
Dále zadáme úhel Ω, který má křivka svírat s hranou plochy.

Dále pro tvorbu křivky podmínka:

$\alpha=\beta$

Pro tvorbu geodetických křivek na ploše použijeme metodu blízké metodě konečných prvků. Vstupní instance programu jsou MESH plocha, počáteční bod křivky a úhel pod kterým má křivka na ploše “vybíhat”.

Nastiňme nyní, jak algorytmus pracuje:



```
120  private OnPolyline DrawGeodesic(OnMesh inMesh, On3dPoint point, double angle)
121
122  {
123    int face_index = 0;
124    int lineIndex = 0;
125    On3dPointArray arr = new On3dPointArray(); arr.Insert(0, point);
126
127    for (int i = 0 ; i < 100 ; i++)
128
129    {
130      OnLine line0 = new OnLine(GetFaceContaingPoint(inMesh, point, face_index)[1], GetFaceContaingPoint(inMesh, point, face_index)[2]);
131      OnLine line1 = new OnLine(GetFaceContaingPoint(inMesh, point, face_index)[0], GetFaceContaingPoint(inMesh, point, face_index)[2]);
132      OnLine line2 = new OnLine(GetFaceContaingPoint(inMesh, point, face_index)[0], GetFaceContaingPoint(inMesh, point, face_index)[1]);
133
134      OnPlane plane = new OnPlane (point, GetFaceContaingPoint(inMesh, point, face_index)[0], GetFaceContaingPoint(inMesh, point, face_index)[2]);
135
136      OnLine line = ConstructLine(plane, angle);
137
138      On3dPoint intPoint = GetIntersectPoint2(line, line0, line1, ref lineIndex);
139
140
141        if (lineIndex == 0) {angle = GetAngleSimple0(line, line0);}
142
143        if (lineIndex == 1) {angle = GetAngleSimple1(line, line1);}
144
145
146
147        face_index = GetFaceIndexContaingPoint(inMesh, point, face_index)[0];
148
149        point = intPoint;
150        arr.Insert(i + 1, point);
151
152
153        if (GetFaceIndexContaingPoint(inMesh, point, face_index).Count == 0 && i > 0) break;
154
155      }
156
157    OnPolyline pLine = new OnPolyline(arr);
158
159    return pLine;
```

Line0 , line1 a *line2* jsou tři úsečky, z nichž každá reprezentuje jednu stranu meshFacu, ve kterém se nachází zadaný bod (MeshFace je název označující dílčí plochu celé Mesh plochy, dalé budu používat termín ”ploška ” kvůli přehlednějšímu skloňování). Zde je třeba upozornit, že algorytmus umí pracovat s MESH plochami s pouze trojúhelníkovými ploškami. Pokud máme k dispozici plochu s jinými než jen trojúhelníkovými ploškami použijeme příkaz Rhina *TriangulateMesh*.

Vytvoříme object typu *plane*. Počátek je zadaný bod *a plane* je určen rovinou plošky. Osa x je jedna ze stran plošky.

ConstructLine je přímka, která začíná v zadaném bodě a svírá zadaný úhel s osou x vytvořeného *plane*.

Najdeme průsečík *ConstructLine* s jednou se dvou zbývajících úseček. Metoda GetIntersectPoint2 nám vrátí index této úsečky a tak poznáme, se kterou se *ConstructLine* proťala .

Pomocí metody *GetAngle* zjistíme úhel, který naše úsečky svírají.

Tento proces se opakuje dlouho dokud nedojdeme k hraniční plošce a je ukončen metodou *break*.

Následuje popis jednotlivých metod:

Následující metoda vrací index plošky, ve které se nachází náš zadaný bod. Každá ploška má tento svůj unikátní index, což je číslo od nuly do celkového počtu plošek na ploše.

```
160  private List<int> GetFaceIndexContaingPoint(OnMesh inMesh, On3dPoint point, int face_index)
161  {
162
163    List<int> indexList = new List<int>();
164
165    for ( int i = 0 ; i < inMesh.FaceCount(); i++)
166
167    {
168
169      OnMeshFace face = inMesh.m_F[i];
170      int index = face.get_vi(0);
171      OnMeshVertexRef reff = inMesh.VertexRef(index);
172      On3dPoint pt0 = reff.Point();
173
174      index = face.get_vi(1);
175      reff = inMesh.VertexRef(index);
176      On3dPoint pt1 = reff.Point();
177
178      index = face.get_vi(2);
179      reff = inMesh.VertexRef(index);
180      On3dPoint pt2 = reff.Point();
181
182      int j = 0;
183
184      if (IsBetween(point, pt0, pt1) && i != face_index) {j++;}
185      if (IsBetween(point, pt1, pt2) && i != face_index) {j++;}
186      if (IsBetween(point, pt2, pt0) && i != face_index) {j++;}
187
188      if (j == 1) {indexList.Add(i);}
189
190    }
191
192    return indexList;
193  }
```

V této metodě je rovněž použita dílčí metoda, kterou jsem pojmenoval *IsBetween.* Do metody vložíme tři body, z nichž první je bodem testovaným. Metoda zjistí, zda se testovaný bod nachází mezi zbývajícími dvěma body:

```
253  private bool IsBetween(On3dPoint testedPoint, On3dPoint pt1, On3dPoint pt2)
254  {
255
256
257    On3dVector vec1 = pt1 - testedPoint;
258    On3dVector vec2 = testedPoint - pt2;
259
260    vec1.Unitize();
261    vec2.Unitize();
262
263    double d = OnUtil.ON_DotProduct(vec1, vec2);
264
265    bool verify = false;
266
267    if (d > 0.9999 && d < 1.0001 ) {verify = true;}
268
269    return verify;
270
271
272
273
274  }
```

Jednoduchá metoda *GetFacePoints* vrátí vrcholu zadané plošky:

```
275  private List<On3dPoint> GetFacePoints(OnMesh mesh, OnMeshFace face)
276  {
277    List<On3dPoint> points = new List<On3dPoint>();
278
279
280    int index = face.get_vi(0);
281    OnMeshVertexRef reff = mesh.VertexRef(index);
282    points.Add(reff.Point());
283
284    index = face.get_vi(1);
285    reff = mesh.VertexRef(index);
286    points.Add(reff.Point());
287
288
289    index = face.get_vi(2);
290    reff = mesh.VertexRef(index);
291    points.Add(reff.Point());
292
293    return points;
294  }
```

GetAngleSimple vrátí úhel dvou úseček. Použity jsou dvě verze, které se liší pořadím odečtených vektorů.

```
295  private double  GetAngleSimple0(OnLine line1, OnLine line2)
296
297  {
298
299
300    On3dVector vec1 = line1.from - line1.to;
301    On3dVector vec2 = line2.from - line2.to;
302
303    vec1.Unitize();
304    vec2.Unitize();
305
306    double d = OnUtil.ON_DotProduct(vec1, vec2);
307    double angle = Math.Acos(d);
308
309
310    return angle;
311
312  }
```

```
313  private double  GetAngleSimple1(OnLine line1, OnLine line2)
314
315  {
316
317
318    On3dVector vec1 = line1.to - line1.from;
319    On3dVector vec2 = line2.from - line2.to;
320
321    vec1.Unitize();
322    vec2.Unitize();
323
324    double d = OnUtil.ON_DotProduct(vec1, vec2);
325    double angle = Math.Acos(d);
326
327
328    return angle;
329
330  }
```

ConstructLine vytvoří přímku na zadaném *plane* a se zadaným úhlem, který bude svírat s osou x tohoto *plane.*

```
331   private OnLine ConstructLine(OnPlane basePlane, double angle)
332   {
333
334     On3dPoint ConPoint = basePlane.PointAt(Math.Cos(angle), Math.Sin(angle));
335     OnLine line = new OnLine(basePlane.origin, ConPoint);
336
337     return line;
338
339   }
```

GetIntersectPoint vrátí průsečík dvou úseček:

```
private On3dPoint GetIntersectPoint2(OnLine conLine, OnLine line1, OnLine line2, ref int lineIndex)


{

  On3dPoint intersectPoint = new On3dPoint (1, 1, 1);
  lineIndex = -11;

  double par1 = 0;
  double par2 = 0;

  double par3 = 0;
  double par4 = 0;

  OnUtil.ON_Intersect(conLine, line1, ref par1, ref par2);

  if ( par2 < 1 && par2 > 0)

  {
    intersectPoint = conLine.PointAt(par1);
    lineIndex = 0;
  }


  if ( par2 > 1 || par2 < 0)

  {
    OnUtil.ON_Intersect(conLine, line2, ref par3, ref par4);
    intersectPoint = conLine.PointAt(par3);
    lineIndex = 1;
  }

  return intersectPoint;


}
```

GetFaceContainingPoint vrátí plošku, ve které se nachází hledaný bod.

```
private List<On3dPoint> GetFaceContaingPoint(OnMesh inMesh, On3dPoint point, int face_index)
{

  List<On3dPoint> pointList = new List<On3dPoint>();

  for ( int i = 0 ; i < inMesh.FaceCount(); i++)

  {
    OnMeshFace face = inMesh.m_F[i];
    int index = face.get_vi(0);
    OnMeshVertexRef reff = inMesh.VertexRef(index);
    On3dPoint pt0 = reff.Point();

    index = face.get_vi(1);
    reff = inMesh.VertexRef(index);
    On3dPoint pt1 = reff.Point();

    index = face.get_vi(2);
    reff = inMesh.VertexRef(index);
    On3dPoint pt2 = reff.Point();

    int j = 0;
    int k = 0;
    int l = 0;

    if (IsBetween(point, pt0, pt1) && i != face_index) {j++;}
    if (IsBetween(point, pt1, pt2) && i != face_index) {k++;}
    if (IsBetween(point, pt2, pt0) && i != face_index) {l++;}

    if (j == 1)

    {
      pointList.Add(pt0);
      pointList.Add(pt1);
      pointList.Add(pt2);
    }


    if (k == 1)

    {
      pointList.Add(pt1);
      pointList.Add(pt2);
      pointList.Add(pt0);
    }

    if (l == 1)

    {
      pointList.Add(pt2);
      pointList.Add(pt0);
      pointList.Add(pt1);
    }
  }
  return pointList;
}
```