```cpp
#include "StdAfx.h"
#include "SMPProfile.h"


extern int LibMessage(LPCTSTR str, int flags = MB_OK);


bool SMPProfile::updateParameters() {
    this->points.clear();

    if (this->angleAlpha >= this->cutAngleGamma) {
        LibMessage(_T("Неверные углы наклоны"));
        return false;
    }

    this->delta_x = 0;
    this->delta_y = 0;

    double delta_rad = 0;
    if (this->isByInnerRadius() && this->cutRoundingType && (this->cutFilletRadius > 0)) {
        double angle = (90.0 - this->angleAlpha - this->cutAngleGamma) / 2;
        double length = this->cutFilletRadius / sind(angle);
        double length_x = length * sind(this->angleAlpha + angle);
        double length_y = length * sind(this->cutAngleGamma + angle);
        delta_rad = length_x - this->cutFilletRadius;
    }

    if (this->isCircle()) {
        this->innerRadius = this->size + delta_rad;
        this->outerRadius = this->innerRadius;
        this->side = this->innerRadius;
    }
    else if (this->isNGon()) {
        unsigned short count = this->getNGonSideCount();
        double angle_s = 360.0 / count;
        double angle_c = (180.0 - angle_s) / 2;
        if (this->isByInnerRadius()) {
            this->innerRadius = this->size;
        }
        else if (this->isByOuterRadius()) {
            this->innerRadius = this->size * sind(angle_c);
        }
        else if (this->isBySide()) {
            this->innerRadius = this->size * sind(angle_c) * sind(angle_c) / sind(angle_s);
        }
        if (this->isByInnerRadius()) {
            this->innerRadius += delta_rad;
        }
        this->outerRadius = this->innerRadius / sind(angle_c);
        if (this->isByOuterRadius()) {
            delta_rad = this->roundingRadius / sind(angle_c) - this->roundingRadius;
            this->outerRadius += delta_rad;
```

```
53              this->innerRadius = this->outerRadius * sind(angle_c);
54          }
55          this->side = this->outerRadius * sind(angle_s / 2) / sind
              (angle_c);
56      }
57      else if (this->isTrigon()) {
58          if (this->isByInnerRadius()) {
59              this->innerRadius = this->size;
60          }
61          else if (this->isByOuterRadius()) {
62              this->innerRadius = this->size * sind(40);
63          }
64          else if (this->isBySide()) {
65              this->innerRadius = this->size * sind(40) * sind(80) / sind
                (60);
66          }
67          if (this->isByInnerRadius()) {
68              this->innerRadius += delta_rad;
69          }
70          this->outerRadius = this->innerRadius / sind(40);
71          if (this->isByOuterRadius()) {
72              delta_rad = this->roundingRadius / sind(40) - this-
                >roundingRadius;
73              this->outerRadius += delta_rad;
74              this->innerRadius = this->outerRadius * sind(40);
75          }
76          this->side = this->outerRadius * sind(60) / sind(80);
77      }
78      else if (this->isRhombus()) {
79          double angle = this->getRhombusAngle() / 2;
80          if (this->isByInnerRadius()) {
81              this->innerRadius = this->size;
82          }
83          else if (this->isByOuterRadius()) {
84              this->innerRadius = this->size * sind(angle);
85          }
86          else if (this->isBySide()) {
87              this->innerRadius = this->size * sind(angle) * cosd(angle);
88          }
89          if (this->isByInnerRadius()) {
90              this->innerRadius += delta_rad;
91          }
92          this->outerRadius = this->innerRadius / sind(angle);
93          if (this->isByOuterRadius()) {
94              delta_rad = this->roundingRadius / sind(angle / 2) - this-
                >roundingRadius;
95              this->outerRadius += delta_rad;
96              this->innerRadius = this->outerRadius * sind(angle);
97          }
98          this->side = this->outerRadius / cosd(angle);
99      }
100
101     if (this->hasHole && (this->holeRadius > this->innerRadius)) {
102         this->holeRadius = this->innerRadius;
103     }
104     return true;
105 `
```

```
106
107 void SMPProfile::createModel() {
108     this->drawBase();
109     this->addExtrusion();
110     this->addCutEvolution();
111     if (this->hasHole) {
112         this->addHole();
113     }
114 }
115
116 void SMPProfile::addCutEvolution() {
117     IEntityPtr cut_cut = this->part->NewEntity(o3d_cutEvolution);
118     ICutEvolutionDefinitionPtr cut_cut_definition = cut_cut-
        >GetDefinition();
119     cut_cut->SetName(_T("Режущая кромка"));
120
121     IEntityPtr cut_sketch = this->part->NewEntity(o3d_sketch);
122     ISketchDefinitionPtr cut_sketch_definition = cut_sketch-
        >GetDefinition();
123     cut_sketch->SetName(_T("Профиль режущей кромки"));
124
125     IBodyPtr body = this->part->GetMainBody();
126     IFaceCollectionPtr face_collection = body->FaceCollection();
127     IFaceDefinitionPtr face_base = this->findEvolutionFaces
        (face_collection);
128
129     IEntityCollectionPtr edges = cut_cut_definition->PathPartArray();
130     IEdgeCollectionPtr edge_collection = face_base->EdgeCollection();
131     this->findEvolutionEdges(edges, edge_collection, face_base);
132
133     IEntityPtr plane = this->createEvolutionPlane();
134     cut_sketch_definition->SetPlane(plane);
135     cut_sketch->Create();
136
137     this->createEvolutionSketch(cut_sketch, cut_sketch_definition);
138     cut_cut_definition->SetSketch(cut_sketch);
139     cut_cut->Create();
140
141     if (!this->isCircle() && (this->roundingRadius > 0)) {
142         body = this->part->GetMainBody();
143         IEntityPtr fillet_rounding = this->part->NewEntity(o3d_fillet);
144         IFilletDefinitionPtr fillet_rounding_definition =
            fillet_rounding->GetDefinition();
145         fillet_rounding->SetName(_T("Скругление боковых рёбер
            пластины"));
146         fillet_rounding_definition->SetRadius(this->roundingRadius);
147
148         IEntityCollectionPtr fillet_rounding_objects =
            fillet_rounding_definition->Array();
149         IFaceCollectionPtr face_rounding_collection = body-
            >FaceCollection();
150         this->addRounding(fillet_rounding_objects,
            face_rounding_collection);
151         fillet_rounding->Create();
152
153         if (this->cutRoundingType) {
```

```
154             body = this->part->GetMainBody();
155             IEntityPtr fillet_cut_edges_rounding = this->part-
                  >NewEntity(o3d_fillet);
156             IFilletDefinitionPtr fillet_cut_edges_rounding_definition =
                   fillet_cut_edges_rounding->GetDefinition();
157             fillet_cut_edges_rounding->SetName(_T("Скругление углов
                  режущей кромки"));
158             double radius = this->roundingRadius * cosd(this-
                  >cutAngleGamma);
159             fillet_cut_edges_rounding_definition->SetRadius(radius);
160
161             IEntityCollectionPtr fillet_cut_edges_rounding_objects =
                  fillet_cut_edges_rounding_definition->Array();
162             IFaceCollectionPtr face_cut_edges_rounding_collection =
                  body->FaceCollection();
163             this->addEvolutionEdgesRounding
                  (fillet_cut_edges_rounding_objects,
                  face_cut_edges_rounding_collection);
164             fillet_cut_edges_rounding->Create();
165         }
166     }
167
168     body = this->part->GetMainBody();
169     if (this->cutRoundingType) {
170         IEntityPtr fillet = this->part->NewEntity(o3d_fillet);
171         IFilletDefinitionPtr fillet_definition = fillet->GetDefinition
              ();
172         fillet->SetName(_T("Скругление режущей кромки"));
173         fillet_definition->SetRadius(this->cutFilletRadius);
174
175         IEntityCollectionPtr rounding_objects = fillet_definition-
              >Array();
176         IFaceCollectionPtr face_cut_rounding_collection = body-
              >FaceCollection();
177         this->findEvolutionRoundingEdges(rounding_objects,
              face_cut_rounding_collection);
178         fillet->Create();
179     }
180     else {
181         IEntityPtr chamfer = this->part->NewEntity(o3d_chamfer);
182         IChamferDefinitionPtr chamfer_definition = chamfer-
              >GetDefinition();
183         chamfer->SetName(_T("Фаска режущей кромки"));
184         double x = this->cutChamferLength * sind(this-
              >cutChamferAngle);
185         double y = this->cutChamferLength * cosd(this-
              >cutChamferAngle);
186         chamfer_definition->SetChamferParam(true, x, y);
187
188         IEntityCollectionPtr rounding_objects = chamfer_definition-
              >Array();
189         IFaceCollectionPtr face_cut_rounding_collection = body-
              >FaceCollection();
190         this->findEvolutionRoundingEdges(rounding_objects,
```

```
            face_cut_rounding_collection);
191         chamfer->Create();
192     }
193 }
194
195 IFaceDefinitionPtr SMPProfile::findEvolutionFaces(IFaceCollectionPtr face_collection) {
196     double x, y;
197     tie(x, y) = this->getPointFirst();
198     IFaceDefinitionPtr face_base = NULL;
199     IFaceDefinitionPtr face = face_collection->First();
200     for (size_t i = 0; i < face_collection->GetCount(); i++) {
201         if (face->IsPlanar()) {
202             ISurfacePtr surface = face->GetSurface();
203             double x1, y1, z1, x2, y2, z2;
204             surface->GetGabarit(&x1, &y1, &z1, &x2, &y2, &z2);
205             if ((round(z1) == this->Z) &&
206                 (round(z2) == this->Z)) {
207                 face_base = face;
208             }
209         }
210         face = face_collection->Next();
211     }
212     return face_base;
213 }
214
215 void SMPProfile::findEvolutionEdges(IEntityCollectionPtr edges,
216     IEdgeCollectionPtr edge_collection,
217     IFaceDefinitionPtr face_base) {
218     IEdgeDefinitionPtr edge = NULL;
219     if ((this->isPentagon()) ||
220         (this->isTriangle()) ||
221         (this->isTrigon())) {
222         edge = edge_collection->Last();
223         for (size_t i = 0; i < edge_collection->GetCount(); i++) {
224             if (edge != NULL) {
225                 edges->Add(edge->GetEntity());
226             }
227             edge = edge_collection->Prev();
228         }
229     }
230     else {
231         edge = edge_collection->First();
232         for (size_t i = 0; i < edge_collection->GetCount(); i++) {
233             if (edge != NULL) {
234                 edges->Add(edge->GetEntity());
235             }
236             edge = edge_collection->Next();
237         }
238     }
239 }
240
241 IEntityPtr SMPProfile::createEvolutionPlane() {
242     IEntityPtr plane_angle = this->part->NewEntity(o3d_planeAngle);
243     IPlaneAngleDefinitionPtr plane_angle_d = plane_angle->GetDefinition
        ();
```

```
244     plane_angle->SetName(_T("Плоскость под углом для профиля режущей
          кромки"));
245
246     plane_angle_d->SetPlane(this->part->GetDefaultEntity
          (o3d_planeXOZ));
247     plane_angle_d->SetAxis(this->part->GetDefaultEntity(o3d_axisOZ));
248
249     double x1, y1, x2, y2;
250     tie(x1, y1) = this->getPointFirst();
251     tie(x2, y2) = this->getPointLast();
252
253     double x = x2 - x1;
254     double y = y2 - y1;
255     double length = sqrt(pow(x, 2) + pow(y, 2));
256     double angle_rotate = asind(x / length);
257     plane_angle_d->SetAngle(angle_rotate);
258     plane_angle->Update();
259
260     IEntityPtr plane_offset = this->part->NewEntity(o3d_planeOffset);
261     IPlaneOffsetDefinitionPtr plane_offset_d = plane_offset-
          >GetDefinition();
262     plane_offset->SetName(_T("Плоскость профиля режущей кромки"));
263     plane_offset_d->SetPlane(plane_angle);
264     double offset = 0;
265     if (this->isNGon()) {
266         offset = y2 + length;
267     }
268     else if (this->isRhombus()) {
269         double angle = this->getRhombusAngle() / 2;
270         double d1 = this->outerRadius * tand(angle);
271         offset = d1 * sind(abs(angle_rotate));
272     }
273     else if (this->isTrigon()) {
274         offset = -this->innerRadius * cotd(80);
275     }
276     plane_offset_d->SetOffset(offset);
277     plane_offset->Update();
278     return plane_offset;
279 }
280
281 bool SMPProfile::createEvolutionSketch(IEntityPtr cut_sketch,
      ISketchDefinitionPtr cut_sketch_definition) {
282     double delta_x = 0;
283     if (this->isTrigon()) {
284         delta_x = this->outerRadius * sind(40);
285     }
286     else if (this->isRhombus()) {
287         double angle = this->getRhombusAngle();
288         delta_x = (this->side / 2) * sind(angle);
289     }
290     else {
291         delta_x = this->innerRadius;
292     }
293     double x1 = this->X + delta_x;
294     double y1 = this->Z;
295     double x2 = x1 + this->cutLength * sind(this->cutAngleGamma);
```

```
296     double y2 = y1 - this->cutLength * cosd(this->cutAngleGamma);
297
298     double angle_r = 180.0 - this->cutAngleGamma;
299     double rad = this->cutRadius;
300     double xc = x2 + rad * cosd(angle_r);
301     double yc = y2 + rad * sind(angle_r);
302     if (yc + rad < this->Z) {
303         LibMessage(_T("Маленький радиус режущей кромки"));
304         return false;
305     }
306
307     double angle_a1 = asind((yc - y1) / rad) + 180.0;
308     double angle_a2 = -this->cutAngleGamma;
309     double x3 = xc + rad * sind(angle_a1 + 90.0);
310     double y3 = y1;
311     if (x3 <= 0) {
312         LibMessage(_T("Большая режущая кромка"));
313         return false;
314     }
315
316     short arc_direction = 1;
317     if (this->isTrigon()) {
318         x1 = -x1;
319         x2 = -x2;
320         x3 = -x3;
321         xc = -xc;
322         angle_a1 = 180.0 - angle_a1;
323         angle_a2 = 180.0 - angle_a2;
324         arc_direction = -arc_direction;
325     }
326     cut_sketch_definition->BeginEdit();
327     LineSeg(x1, y1, x2, y2, ksCurveStyleEnum::ksCSNormal);
328     LineSeg(x1, y1, x3, y3, ksCurveStyleEnum::ksCSNormal);
329     ArcByAngle(xc, yc, rad, angle_a1, angle_a2, arc_direction,
        ksCurveStyleEnum::ksCSNormal);
330     cut_sketch_definition->EndEdit();
331     return true;
332 }
333
334 void SMPProfile::addEvolutionEdgesRounding(IEntityCollectionPtr
      fillet_objects, IFaceCollectionPtr face_collection) {
335     if (!this->isCircle() && (this->roundingRadius > 0)) {
336         double z1 = 0;
337         double z2 = z1 + this->cutLength * cosd(this->cutAngleGamma);

338         debug("addEvolutionEdgesRounding");
339         debug(z1);
340         debug(z2);
341         IFaceDefinitionPtr face = face_collection->First();
342         for (size_t i = 0; i < face_collection->GetCount(); i++) {
343             if (face->IsPlanar()) {
344                 ISurfacePtr surface = face->GetSurface();
345                 double s_x1, s_y1, s_z1, s_x2, s_y2, s_z2;
346                 surface->GetGabarit(&s_x1, &s_y1, &s_z1, &s_x2, &s_y2,
```

```
347
348                 debug("GetSurface");
349                 debug(s_z1);
350                 debug(s_z2);
351                 if ((round(s_z1) == round(z1)) &&
352                     (round(s_z2) == round(z2))) {
353                     IEdgeCollectionPtr edge_collection = face->EdgeCollection();
354                     IEdgeDefinitionPtr edge = edge_collection->First();
355                     for (size_t j = 0; j < edge_collection->GetCount(); j++) {
356                         ICurve3DPtr curve = edge->GetCurve3D();
357                         double c_x1, c_y1, c_z1, c_x2, c_y2, c_z2;
358                         curve->GetGabarit(&c_x1, &c_y1, &c_z1, &c_x2, &c_y2, &c_z2);
359                         debug("GetCurve3D");
360                         debug(c_z1);
361                         debug(c_z2);
362                         if ((c_z1 != c_z2) &&
363                             round(c_z2) == round(z2)) {
364                             fillet_objects->Add(edge->GetEntity());
365                         }
366                         edge = edge_collection->Next();
367                     }
368                 }
369             }
370             face = face_collection->Next();
371         }
372     }
373 }
374
375 void SMPProfile::findEvolutionRoundingEdges(IEntityCollectionPtr rounding_objects, IFaceCollectionPtr face_collection) {
376     IFaceDefinitionPtr face = face_collection->First();
377     for (size_t i = 0; i < face_collection->GetCount(); i++) {
378         if (this->checkEvolutionRoundingFace(face)) {
379             ISurfacePtr surface = face->GetSurface();
380             double s_x1, s_y1, s_z1, s_x2, s_y2, s_z2;
381             surface->GetGabarit(&s_x1, &s_y1, &s_z1, &s_x2, &s_y2, &s_z2);
382             if (round(s_z1) != round(s_z2)) {
383                 IEdgeCollectionPtr edge_collection = face->EdgeCollection();
384                 IEdgeDefinitionPtr edge = edge_collection->First();
385                 for (size_t j = 0; j < edge_collection->GetCount(); j++) {
386                     ICurve3DPtr curve = edge->GetCurve3D();
387                     double c_x1, c_y1, c_z1, c_x2, c_y2, c_z2;
388                     curve->GetGabarit(&c_x1, &c_y1, &c_z1, &c_x2, &c_y2, &c_z2);
389                     if ((round(c_z1) == this->Z) &&
390                         (round(c_z2) == this->Z)) {
391                         rounding_objects->Add(edge->GetEntity());
392                     }
393                     edge = edge_collection->Next();
```

```
394                     }
395                 }
396             }
397             face = face_collection->Next();
398         }
399     }
400
401     bool SMPProfile::checkEvolutionRoundingFace(IFaceDefinitionPtr face) {
402         bool check = false;
403         if (face != NULL && (face->IsPlanar() || face->IsCylinder() ||
              face->IsCone())) {
404             check = true;
405             if (face->IsCylinder() || face->IsCone()) {
406                 double h, r;
407                 face->GetCylinderParam(&h, &r);
408                 if (round(_x: h) != round(_x: this->getHeight())) {
409                     if (!this->isCircle()) {
410                         if (round(_x: r) != round(_x: this->roundingRadius))
                          {
411                             check = false;
412                         }
413                     }
414                     else {
415                         if (round(_x: r) != round(_x: this->outerRadius)) {
416                             check = false;
417                         }
418                     }
419                 }
420             }
421         }
422         return check;
    }
```