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

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

void SMPConditions::createModel() {
    this->addEmbodiment();
    this->drawBase();
    this->addExtrusion();
    this->addRounding();
    this->addBladePlanes();
    this->addHole();
}

bool SMPConditions::updateParameters() {
    double angle_main = 0;
    this->points.clear();
    if (this->isCircle()) {
        this->updateParametersCircle();
    }
    else if (this->isNGon()) {
        this->updateParametersNGon(&angle_main);
    }
    else if (this->isTrigon()) {
        this->updateParametersTrigon(&angle_main);
    }
    else if (this->isRhombus()) {
        this->updateParametersRhombus(&angle_main);
    }
    if (this->hasHole && (this->holeRadius > this->innerRadius)) {
        this->holeRadius = this->innerRadius;
    }
    return true;
}

void SMPConditions::updateParametersCircle() {
    if (!this->isCircle()) {
        return;
    }
    this->innerRadius = this->size;
    this->outerRadius = this->innerRadius;
    this->side = this->innerRadius;
}

void SMPConditions::updateParametersNGon(double* angle_main) {
    if (!this->isNGon()) {
        return;
    }
    unsigned short count = this->getNGonSideCount();
    double angle_vertex = 360.0 / count;
    double angle_rad = (180.0 - angle_vertex) / 2;
    *angle_main = angle_rad;
    if (this->isByInnerRadius()) {
        this->innerRadius = this->size;
    }
    else if (this->isByOuterRadius()) {
```

```
57          this->innerRadius = this->size * sind(angle_rad);
58      }
59      else if (this->isBySide()) {
60          this->innerRadius = this->size * sind(angle_rad) * sind
              (angle_rad) / sind(angle_vertex);
61      }
62      this->outerRadius = this->innerRadius / sind(angle_rad);
63      this->delta_max = this->outerRadius;
64      if (this->isByOuterRadius()) {
65          double delta_rad = this->roundingRadius / sind(angle_rad) -
              this->roundingRadius;
66          this->outerRadius += delta_rad;
67          this->innerRadius = this->outerRadius * sind(angle_rad);
68      }
69      this->side = this->outerRadius * sind(angle_vertex / 2) / sind
          (angle_rad);
70      if (!this->isByOuterRadius()) {
71          double delta_rad = this->roundingRadius / sind(angle_rad) -
              this->roundingRadius;
72          this->outerRadius -= delta_rad;
73      }
74  }
75
76  void SMPConditions::updateParametersTrigon(double* angle_main) {
77      if (!this->isTrigon()) {
78          return;
79      }
80      double angle_vertex = 60;
81      double angle_greater = 80;
82      double angle_less = 40;
83      *angle_main = angle_less;
84      if (this->isByInnerRadius()) {
85          this->innerRadius = this->size;
86      }
87      else if (this->isByOuterRadius()) {
88          this->innerRadius = this->size * sind(angle_less);
89      }
90      else if (this->isBySide()) {
91          this->innerRadius = this->size * sind(angle_less) * sind
              (angle_greater) / sind(angle_vertex);
92      }
93      this->outerRadius = this->innerRadius / sind(angle_less);
94      this->delta_max = this->outerRadius;
95      if (this->isByOuterRadius()) {
96          double delta_rad = this->roundingRadius / sind(angle_less) -
              this->roundingRadius;
97          this->outerRadius += delta_rad;
98          this->innerRadius = this->outerRadius * sind(angle_less);
99      }
100     this->side = this->outerRadius * sind(angle_vertex) / sind
          (angle_greater);
101     if (!this->isByOuterRadius()) {
102         double delta_rad = this->roundingRadius / sind(angle_less) -
              this->roundingRadius;
103         this->outerRadius -= delta_rad;
104
```

```
105 }
106
107 void SMPConditions::updateParametersRhombus(double* angle_main) {
108     if (!this->isRhombus()) {
109         return;
110     }
111     double angle = this->getRhombusAngle() / 2;
112     *angle_main = angle;
113     if (this->isByInnerRadius()) {
114         this->innerRadius = this->size;
115     }
116     else if (this->isByOuterRadius()) {
117         this->innerRadius = this->size * sind(angle);
118     }
119     else if (this->isBySide()) {
120         this->innerRadius = this->size * sind(angle) * cosd(angle);
121     }
122     this->outerRadius = this->innerRadius / sind(angle);
123     this->delta_max = this->outerRadius;
124     if (this->isByOuterRadius()) {
125         double delta_rad = this->roundingRadius / sind(angle) - this-
              >roundingRadius;
126         this->outerRadius += delta_rad;
127         this->innerRadius = this->outerRadius * sind(angle);
128     }
129     this->side = this->outerRadius / cosd(angle);
130     if (!this->isByOuterRadius()) {
131         double delta_rad = this->roundingRadius / sind(angle) - this-
              >roundingRadius;
132         this->outerRadius -= delta_rad;
133     }
134 }
135
136 void SMPConditions::addBladePlanes() {
137     if (this->isCircle()) {
138         return;
139     }
140     int count = 0;
141     double angle_vertex = 0;
142     double angle_main = 0;
143     if (this->isNGon()) {
144         count = this->getNGonSideCount();
145         angle_vertex = 360.0 / count;
146         angle_main = (180.0 - angle_vertex) / 2;
147     }
148     else if (this->isTrigon()) {
149         count = 3;
150         angle_vertex = 120;
151         angle_main = 40;
152     }
153     else if (this->isRhombus()) {
154         count = 2;
155         angle_vertex = 180;
156         angle_main = this->getRhombusAngle() / 2;
157     }
158
```

```
159     double delta = this->delta_max - this->outerRadius;
160     double max_x = this->roundingRadius * cosd(angle_main);
161     double max_y = -(this->roundingRadius - this->roundingRadius * cosd
          (angle_main));
162     double s_x, s_y, t_x, t_y;
163     s_x = this->parameterS / 2;
164     t_y = -this->parameterT;
165
166     if (s_x <= max_x) {
167         double cat2 = cat(this->roundingRadius, s_x);
168         s_y = cat2 - this->roundingRadius;
169     }
170     else {
171         s_y = delta - s_x * cotd(angle_main);
172     }
173     if (t_y >= max_y) {
174         double cat1 = (this->roundingRadius - this->parameterT);
175         t_x = -cat(this->roundingRadius, cat1);
176     }
177     else {
178         t_x = -(delta - t_y) * tand(angle_main);
179     }
180
181     double length_x = s_x - t_x;
182     double length_y = s_y - t_y;
183     double direction_angle = atand(length_y / length_x);
184     if (this->angleEtaAuto) {
185         this->angleEta = direction_angle + 90;
186     }
187     else {
188         direction_angle = this->angleEta - 90;
189     }
190
191     double x = this->coordinates.X;
192     double y = this->coordinates.Y;
193     double rotation_angle;
194     if (this->isNGon()) {
195         double begin_angle = atand(t_y / t_x);
196         double begin_length = hyp(t_x, t_y);
197         double offset_x = begin_length * cosd(angle_main +
              begin_angle);
198         double offset_y = -begin_length * sind(angle_main +
              begin_angle);
199         x += this->outerRadius * cosd(angle_vertex / 2) + offset_x;
200         y += this->outerRadius * sind(angle_vertex / 2) + offset_y;
201         rotation_angle = 90 - direction_angle + angle_vertex / 2;
202     }
203     else {
204         x += -t_x;
205         y += this->outerRadius + t_y;
206         rotation_angle = 180 - direction_angle;
207     }
208
209     this->addBladePlane(count, x, y, rotation_angle);
210 }
211
```

```
212 void SMPConditions::addBladePlane(int count, double x, double y, double
       rotation_angle) {
213     double z = this->coordinates.Z;
214     double nutation_angle;
215     double precession_angle = 0;
216     double x_c, y_c;
217     if (rotation_angle > 180) {
218         nutation_angle = -this->angleLambdaD;
219         x_c = 0;
220         y_c = this->cutRadius;
221     }
222     else {
223         nutation_angle = this->angleLambdaD;
224         x_c = this->cutRadius;
225         y_c = 0;
226     }
227
228     IAuxiliaryGeomContainerPtr geom_container(this->part7);
229     ILocalCoordinateSystemPtr local_cs = geom_container-
          >LocalCoordinateSystems->Add();
230     local_cs->Name = _T("ЛСК режущей кромки");
231     local_cs->X = x;
232     local_cs->Y = y;
233     local_cs->Z = z;
234     local_cs->OrientationType = ksOrientationTypeEnum::ksEulerCorners;
235     ILocalCSEulerParamPtr local_cs_parameters = local_cs-
          >GetLocalCSParameters();
236     local_cs_parameters->NutationAngle = nutation_angle;
237     local_cs_parameters->PrecessionAngle = precession_angle;
238     local_cs_parameters->RotationAngle = rotation_angle;
239     local_cs->Update();
240
241     ISketchPtr sketch = this->model_container->Sketchs->Add();
242     sketch->CoordinateSystem = local_cs;
243     sketch->Plane = local_cs->GetDefaultObject
          (ksObj3dTypeEnum::o3d_planeYOZ);
244     sketch->Name = _T("Профиль режущей кромки");
245
246     IFragmentDocumentPtr sketch_document = sketch->BeginEdit();
247     IDrawingContainerPtr drawing_container = this->getDrawingContainer
          (sketch_document);
248     ICirclePtr figure = drawing_container->Circles->Add();
249     figure->Xc = x_c;
250     figure->Yc = y_c;
251     figure->Radius = this->cutRadius;
252     figure->Update();
253     sketch->EndEdit();
254
255     ICutExtrusionPtr extrusion = this->model_container->Extrusions->Add
          (ksObj3dTypeEnum::o3d_cutExtrusion);
256     extrusion->Name = _T("Элемент выдавливания режущей кромки");
257     extrusion->Direction = ksDirectionTypeEnum::dtBoth;
258     extrusion->SetSideParameters(
259         false, ksEndTypeEnum::etThroughAll,
```

```
260         0, 0, true, NULL
261     );
262     extrusion->SetSideParameters(
263         true, ksEndTypeEnum::etThroughAll,
264         0, 0, true, NULL
265     );
266     extrusion->Sketch = sketch;
267     extrusion->Update();
268
269     IPoints3DPtr points = this->model_container->Points3D;
270     IPoint3DPtr point1 = points->Add();
271     IPoint3DPtr point2 = points->Add();
272
273     point1->Name = _T("Точка 1 оси массива режущих кромок");
274     point1->X = this->coordinates.X;
275     point1->Y = this->coordinates.Y;
276     point1->Z = this->coordinates.Z;
277     point1->Update();
278
279     point2->Name = _T("Точка 2 оси массива режущих кромок");
280     point2->X = this->coordinates.X;
281     point2->Y = this->coordinates.Y;
282     point2->Z = this->coordinates.Z + this->getHeight();
283     point2->Update();
284
285     IAxis3DBy2PointsPtr axis = this->model_container->AddObject
          (ksObj3dTypeEnum::o3d_axis2Points);
286     axis->Name = _T("Ось массива режущих кромок");
287     axis->Point1 = point1;
288     axis->Point2 = point2;
289     axis->Update();
290
291     ICircularPatternPtr circular_pattern = this->model_container-
          >FeaturePatterns->Add(ksObj3dTypeEnum::o3d_circularCopy);
292     circular_pattern->Name = _T("Массив режущих кромок");
293     circular_pattern->InitialObjects = (ICutExtrusion*)extrusion;
294     circular_pattern->BuildingType =
          ksCircularPatternBuildingTypeEnum::ksCPSaveAll;
295     circular_pattern->BasePoint = this->part7->GetDefaultObject
          (ksObj3dTypeEnum::o3d_pointCS);
296     circular_pattern->Axis = axis;
297     circular_pattern->Step2 = 360;
298     circular_pattern->Count1 = 1;
299     circular_pattern->Count2 = count;
300     circular_pattern->BoundaryInstancesStepFactor2 = true;
301     circular_pattern->Update();
302 }
303
```