6491-2017 P4: Minkowski Morph (MM) of SPCCs

Dibyendu Mondal

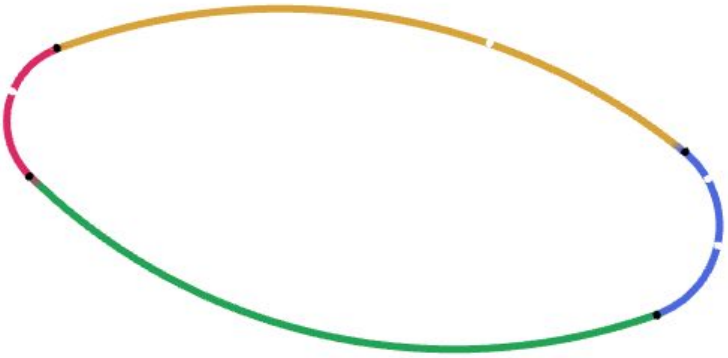
1. Problem Specification

We are given two convex shapes A and B, each bounded by a smooth, piecewise circular curve (SPCC). The sketch must offer some interactive editing tool for designing A and B via click and drag. When the user presses ‘a’, the software should start an animation of a Minkowski Morph (MM) from A to B that lasts about two seconds.

2. Representation of SPCCs

I started with four points that represent two circles. A is the center of one circle, with B being a point on the circumference, and C is the center of the second circle with D being on the circumference. One assumption is that point B will be above the line going through the centers of the two circles, and point D will be below that same line. More assumptions are that these two circles do not intersect.

So basically, the shape is a 4 arc caplet which has 4 control points. I find the centers of the arcs and use the function *drawCircleArcInHat()* to draw the arcs.



3. Minkowski Morph

The problem of Minkowski sum computation arises in many applications including solid modeling, digital geometry processing, robotics, dynamic simulation, and computer animation. The Minkowski sum of two sets P and Q is the set of points $\{p + q \mid p \in P, q \in Q\}$. Minkowski sum has a number of applications. They are useful as a tool to compute collision-free paths in robot motion planning, computer-aided design and manufacturing, satellite layout, penetration depth computation and dynamic simulation. They have also been used for morphing, offset computation, and mathematical morphological operations. Some of the best prior art in this area are mentioned in [1], [2], [3], [4], [5] and [6].

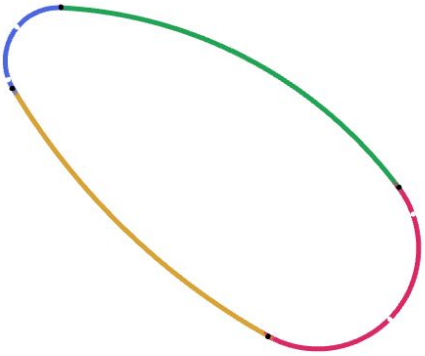
4. Detailed Algorithm

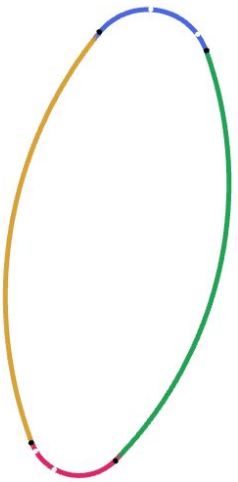
For each of the 4 arcs, I calculate the centers of the arcs and thus find the normal at each of the control points. Two of the centers are known which are A and C. To find the other 2 centers, I do this:

```
vec CB = V(C,B);
vec T = R(U(A,B));
float radius = (c*c - n2(CB))/(2 * dot(CB,T));
vec BQ = T; BQ.scaleBy(radius);
pt Q = P(B,BQ);
float w = atan(radius/c);
vec CM = R(U(C,Q),w).scaleBy(c);
pt M = P(C,CM);
vec QB = U(V(Q,B));
vec QM = U(V(Q,M));
float angle = angle(QB,QM)/2;
float x0 = radius*tan(angle);
pt X = P(B,R(QB).scaleBy(x0));                    // center of arc BM
```

Here we find the points Q, M and X. Q is the corner point of the hat BQM, M is the intersection point of this and the next arc or we can say the end point of this arc. X is the center of the arc BM.
I first find the tangent T at point B by rotating the normal at B by 90º. Next I find the radius of the hat and the find the corner point of the hat Q by scaling the tangent T by radius. Next I find the end point M of the arc BM. Once I get the points Q and M, I find the center X by rotating the vector QB by half of the angle between QB and QM.
Similarly, I find the center Z of the arc DO. Once I have all the control points and the center of the arcs, I calculate the normals by creating a unit vector from the center to the respective control points.

Now we have 4 control points, the centers of the arcs and the normals at the control points of both the SPCCs (A and B). Next I try to find a split on both the SPCCs based on the normals. For each normal on A, I try to find on which arc that normal lies on B and create a split on that arc. I do the same for each normal on B.
The result is shown in the image. The black dots are the control points and the white dots are the splits.





The detailed algorithm for this is as follows:

```
for(int i = 0;i < numArcs;i++)
  {
    for(int j = 0;j < numArcs;j++)
    {
```

```
                if(Qnormals[i].angle()  >=  Pnormals[j].angle()  &&  Qnormals[i].angle()  <
Pnormals[(j+1)%numArcs].angle())
      {
                                                                          PQcontrols.G[i]        =
P(centers0.G[(j+1)%numArcs],d(control0.G[j],centers0.G[(j+1)%numArcs]),U(Qnormals[i]));

        // check if a split already exists between these arcs
        boolean flag = false;
        for(int z = 0; z < k;z+=3)
        {
          if((Arc0.G[z] == control0.G[j]) && (Arc0.G[z+5] == control0.G[(j+1)%numArcs]))
          {
            vec nor = U(Arc0.G[z+1],Arc0.G[z+2]);
            if(nor.angle() > Qnormals[i].angle())
            {
              Arc0.G[z] = PQcontrols.G[i];
              Arc0.G[k++] = control0.G[j];
              Arc0.G[k++] = centers0.G[(j+1)%numArcs];
              Arc0.G[k++] = PQcontrols.G[i];
              count[j]++;
            }
            else
            {
              Arc0.G[z+5] = PQcontrols.G[i];
              Arc0.G[k++] = PQcontrols.G[i];
              Arc0.G[k++] = centers0.G[(j+1)%numArcs];
              Arc0.G[k++] = control0.G[(j+1)%numArcs];
              count[j]++;
            }
            flag = true;
            break;
          }
        }

        if(!flag)
        {
          // storing the 2 arcs
          Arc0.G[k++] = control0.G[j];
          Arc0.G[k++] = centers0.G[(j+1)%numArcs];
          Arc0.G[k++] = PQcontrols.G[i];
          Arc0.G[k++] = PQcontrols.G[i];
          Arc0.G[k++] = centers0.G[(j+1)%numArcs];
          Arc0.G[k++] = control0.G[(j+1)%numArcs];
          count[j]++;
        }
        break;
      }
                else  if(0  <=  Pnormals[j].angle()  &&  Pnormals[j].angle()  <=  PI  &&
Pnormals[(j+1)%numArcs].angle() >= -PI && Pnormals[(j+1)%numArcs].angle() <= 0)
      {
              if((Qnormals[i].angle()  >=  Pnormals[j].angle()  &&  Qnormals[i].angle()  <=  PI)  ||
(Qnormals[i].angle() >= -PI && Qnormals[i].angle() <= Pnormals[(j+1)%numArcs].angle()))
        {
```

```
                                                                    PQcontrols.G[i]       =
P(centers0.G[(j+1)%numArcs],d(control0.G[(j+1)%numArcs],centers0.G[(j+1)%numArcs]),Qnormals[i]);

          // check if a split already exists between these arcs
          boolean flag = false;
        for(int z = 0; z < k;z+=3)
        {
          if((Arc0.G[z] == control0.G[j]) && (Arc0.G[z+5] == control0.G[(j+1)%numArcs]))
          {
            vec nor = U(Arc0.G[z+1],Arc0.G[z+2]);
            if((nor.angle() > Qnormals[i].angle())  || (nor.angle() >= -PI && nor.angle() <= 0 &&
0 <= Qnormals[i].angle() && Qnormals[i].angle() <= PI))
            {
              Arc0.G[z] = PQcontrols.G[i];
              Arc0.G[k++] = control0.G[j];
              Arc0.G[k++] = centers0.G[(j+1)%numArcs];
              Arc0.G[k++] = PQcontrols.G[i];
              count[j]++;
            }
            else
            {
              Arc0.G[z+5] = PQcontrols.G[i];
              Arc0.G[k++] = PQcontrols.G[i];
              Arc0.G[k++] = centers0.G[(j+1)%numArcs];
              Arc0.G[k++] = control0.G[(j+1)%numArcs];
              count[j]++;
            }
            flag = true;
            break;
          }
        }

        if(!flag)
        {
          // storing the 2 arcs
          Arc0.G[k++] = control0.G[j];
          Arc0.G[k++] = centers0.G[(j+1)%numArcs];
          Arc0.G[k++] = PQcontrols.G[i];
          Arc0.G[k++] = PQcontrols.G[i];
          Arc0.G[k++] = centers0.G[(j+1)%numArcs];
          Arc0.G[k++] = control0.G[(j+1)%numArcs];
          count[j]++;
        }
          break;
        }
      }
    }
  }
```

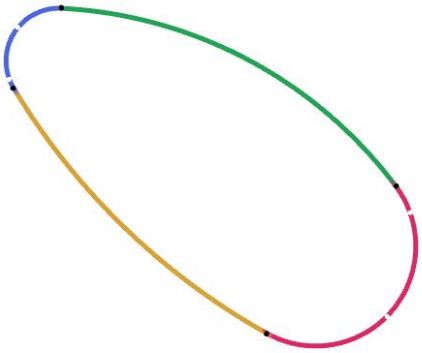
Here, for each normal on Q, I compare the angle of that normal wrt x-axis with 2 consecutive angles of the normals on P. If the angle of normal on Q lies between the 2 angles of normals on P, I create a new control point on P by scaling the unit normal on Q by the radius of the arc on P from the center of that same arc. Thus we get the split on the arc on P. No I create 2 new arcs. First arc starting from the control point of the 1st normal on P and ending at the newly created split control point and second arc starting from the split control point and ending at the control point of the 2nd normal

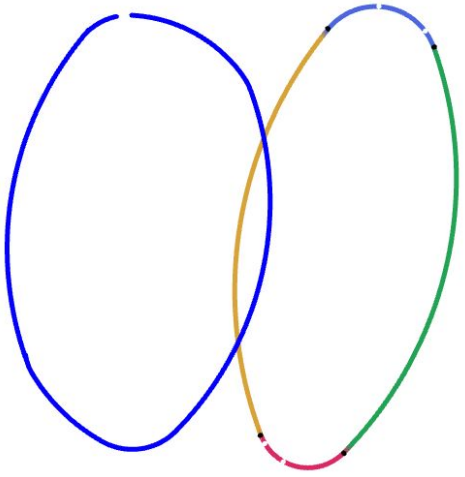
on P. I also check if there is already an existing split in the same arc of P. If there is one then I update the control points of the arcs accordingly to accommodate the new split and create 1 new arc.

I also handled a few special cases. One of them is when the angle of one of the normals on P is < PI and the angle for the next one is > -PI. This happens since the angle varies from -PI to PI with a discontinuity at the end. I fixed this by separately comparing the angles of the normal with PI and -PI as shown in the code. Another special case is when there are no splits in one of the existing arcs of P. If this happens then no new arc is created for the same existing arc. To fix this I kept a counter for number of new arcs created starting from each control point. If no new arc starts from a control point, I create a new arc from that control point to the next control point.
Next is the algorithm for morphing:

```
for(int i = 0;i < S1;i+=3)
  {
    vec normal0 = U(Arc0.G[i+1],Arc0.G[i]);
    for(int j = 0;j < S2;j+=3)
    {
      vec normal1 = U(Arc1.G[j+1],Arc1.G[j]);
      if(abs(angle(normal0,normal1)) < 0.1)  // matching arc found
      {
        newArc.G[k] = L(Arc0.G[i],Arc1.G[j],t);
        newArc.G[k+1] = L(Arc0.G[i+1],Arc1.G[j+1],t);
        newArc.G[k+2] = L(Arc0.G[i+2],Arc1.G[j+2],t);
        drawArc(k);
        k= k+3;
        break;
      }
    }
  }
```

Here, I compare the normals at the 1st control point at arcs on P and Q and check if they have almost the same angle i.e. they are the same arcs. If this is true, I create a new arc by doing a LERP at the 3 control points of both the arcs at time t. Then I draw this newly created arc.

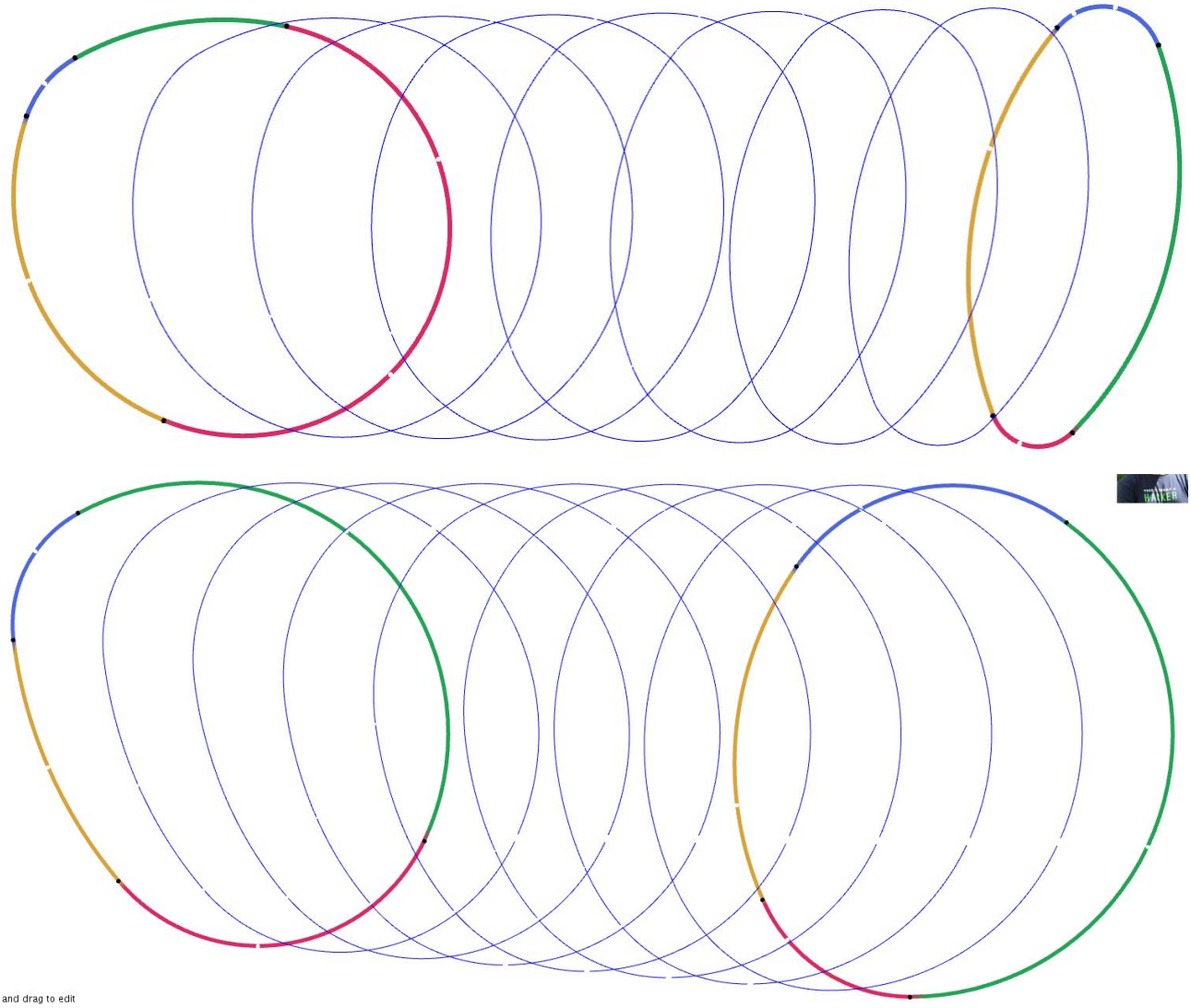




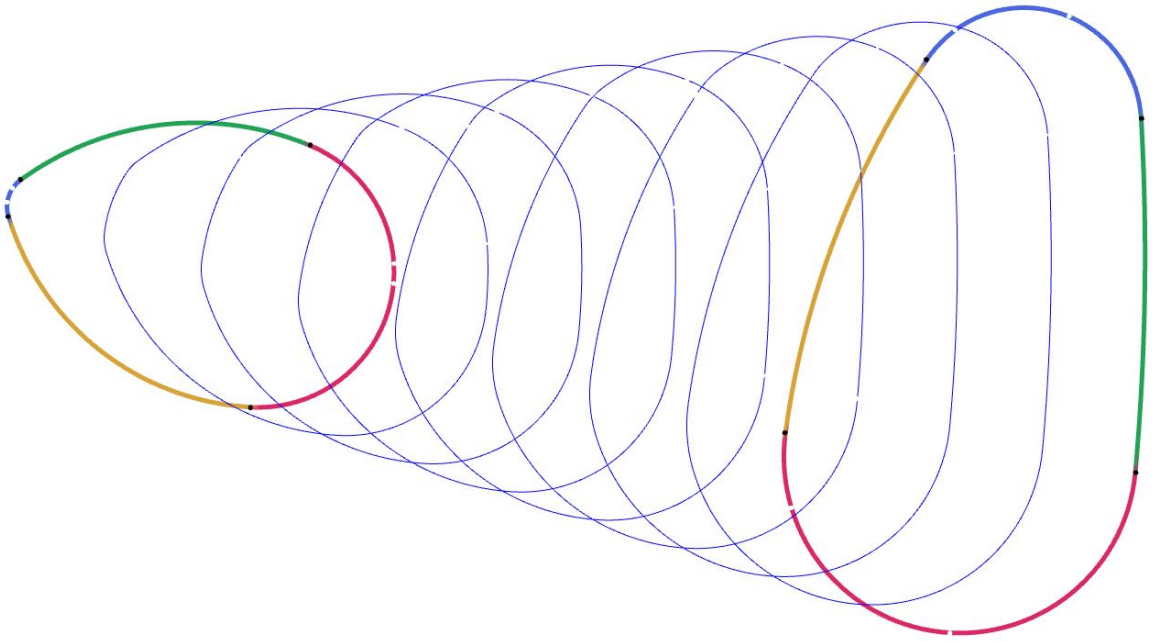
5. Justification

The final morphing should generate a smooth curve which interpolates or warps between SPCC A and SPCC B. If the SPCCs are convex, my algorithm generates smooth morphs between them.

6. Images of Intermediate frames



and drag to edit



7. References

[1] Behandish, Morad, and Horea T. Ilieş. "Analytic methods for geometric modeling via spherical decomposition." *Computer-Aided Design* 70 (2016): 100-115.
[2] Li, Wei, and Sara McMains. "Voxelized Minkowski sum computation on the GPU with robust culling." *Computer-Aided Design* 43.10 (2011): 1270-1283.
[3] Kaul, Anil, and Jarek Rossignac. "Solid-interpolating deformations: construction and animation of PIPs." *Computers & graphics* 16.1 (1992): 107-115.
[4] Rossignac, Jarek, Ioannis Fudos, and Andreas Vasilakis. "Direct rendering of Boolean combinations of self-trimmed surfaces." *Computer-Aided Design* 45.2 (2013): 288-300.
[5] Varadhan, Gokul, and Dinesh Manocha. "Accurate Minkowski sum approximation of polyhedral models." *Graphical Models* 68.4 (2006): 343-355.
[6] Lee, J. H., et al. "Interactive control of geometric shape morphing based on Minkowski sum." *Trans. SCCE* 7.4 (2002): 317-326.