CS6491-2017—Project 4: Minkowski Morph (MM) of SPCCs

Jin Lin

1 Problem specification

Given two convex shapes A and B, each bounded by a smooth, piecewise circular curve (SPCC), and time t $\in$[0, 1], create a Minkowski Morph MM(A, t, B).

2 Representation of PCCs

I apply bi-arc method which uses a set of 6 control points to control the shape of each PCC. The representation of each *Arc* stores:

- Points: point C being the center of the supporting circle (note as *Arc.C*), point A being the starting point (*Arc.A*), and point B (*Arc.B*), being the ending point;
- Vectors: vector N being the unit normal of at the starting point (*Arc.N*);
- Bits: 1 bit to indicate the direction of the arc (*Arc.cw*, clockwise or counter-clockwise, thus our arc is uniquely defined) and 1 bit to indicate the convexity of the arc (*Arc.convex*, true or false). In the figure below, the blue arc, with starting point at A and ending point at B, is clockwise and convex. The yellow arc is counter-clockwise and concave.

3 Overview of Minkowski Morph

Minkowski Sum of two sets A and B is: A$\oplus$B = {a+b, a$\in$A, b$\in$B}, more specifically, A$\oplus$B = {a+b-o, a$\in$A, b$\in$B}. And Minkowski Morph = MM(A, t, B) = (1-t)A$\oplus$tB = {(1-t)a+tb, a$\in$A, b$\in$B, t $\in$[0, 1]}, so Minkowski Morph is a kind of interpolation from point a in A to point b in B with constant velocity.

Reference from PAPER Minkowski by Li McMains in CAD11:

The Minkowski sum of two point sets A and B in R^n is defined

$$A \oplus B = \{a + b \mid a \in A, b \in B\};$$

where a and b denote the coordinate vectors of arbitrary points in A and B, and + denotes vector addition. If A and B represent polygons in R^2 or polyhedra in R^3, A $\oplus$ B can be generated by "sweeping" object A along the boundary of object B (or vice versa). This gives another equivalent definition of Minkowski sums, shown below, where Ba denotes the translation of object B by the vector a.

$$A \oplus B = \bigcup_{a \in A} B_a = \bigcup_{b \in B} A_b.$$

The linear interpolation between two objects A and B can be computed using Minkowski sums as $(1-t)A \oplus tB$, $t \in [0, 1]$.

4 Overview of algorithm

When PCC A and PCC B are both non-concave, I first split A by B and B by A to obtain the same number of arcs, and for each arc in A, there is a relevant arc in B which has the same normal pencil with it. And when given time t, for each pair of arcs *ArcA* and *ArcB*, I compute a new arc *ArcM* = LERP(*ArcA*, t, *ArcB*) as:

ArcM.A = LERP(*ArcA.A*, t, *ArcB.A*);

ArcM.B = LERP(*ArcA.B*, t, *ArcB.B*);

ArcM.C = LERP(*ArcA.C*, t, *ArcB.C*);

ArcM.cw = ArcA.A.cw;

ArcM.convex = true;

ArcM has the same normal pencil with A and B.

Then I use these new arcs to construct a new PCC.

5 Justification of my algorithm

Let's note *ArcA* as *L, ArcB* as *R, ArcM* as *M*, LERP(*L*, t, *R*) as *M2*. To improve *M* = *M2*, first improve that *M* has the same normal pencil with *M2*:

We have:

NormalPencil(*L*) = [N(*L.A*), N(*L.B*)], NormalPencil(*R*) = [N(*R.A*), N(*R.B*)],

and as definition:

NormalPencil(*L*) = NormalPencil(*R*) = NormalPencil(*M*);

as in *M2*, we have:

N(*M2.A*) = U(*M2.C*, *M2.A*) = U(LERP(*L.C*, t, *R.C*), LERP(*L.A*, t, *R.A*)),

N(*M2.B*) = U(*M2.C*, *M2.B*) = U(LERP(*L.C*, t, *R.C*), LERP(*L.B*, t, *R.B*)),

since N(*L.A*) = N(*R.A*), so line(*L.C*, *L.A*)// line(*R.C*, *R.A*), obviously:

line(*M2.C*, *M2.A*)// line(*L.C*, *L.A*)// line(*R.C*, *R.A*), same as line(*M2.C*, *M2.B*)// (*L.C*, *L.B*)// line(*R.C*, *R.B*),

so N(*M2.A*) = N(*L.A*) = N(*R.A*), N(*M2.B*) = N(*L.B*) = N(*R.B*), finally:

NormalPencil(*M2*) = NormalPencil(*L*) = NormalPencil(*R*) = NormalPencil(*M*).

After that, we should improve that *M* and *M2* have the same radium:

First:

Radium(*M2*) = d(*M2.C*, *M2.A*) = d(LERP(*L.C*, t, *R.C*), LERP(*L.A*, t, *R.A*)) = d(LERP(Radium(*L*), t, Radium(*R*))),

as definition of Minkowski Morph, when the morphing objects are both line segment:

line(*L.C*, *L.A*) and line(*R.C*, *R.A*)

we have:

line(*M.C*, *M.A*) = LERP(line(*L.C*, *L.A*), t, line(*R.C*, *R.A*)),

Then:

Radium(*M*) = d(LERP(Radium(*L*), t, Radium(*R*))) = Radium(*M2*).

So, *M* = *M2.*

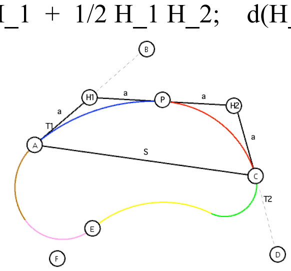
6 Details of my algorithm

First I use bi-arc method to compute PCC. For example, when given 4 points (A, B, C, D):

Step 1. Compute unit vector T_1= AB/(|AB|) and unit vector T_2= CD/(|CD|).

Step 2. Find the common end point P of the potential arc starting at A (the blue arc in the picture below) and the potential arc ending at C (the red arc in the picture below). Assume that H_1 is the tip of the hat corresponding to the blue arc and H_2 is the tip of the hat corresponding to the red arc. Connect H_1 and H_2, and P is supposed to be on the line going through H_(1)and H_2. Assume that H_1= A + aT_1 and H_2=C-bT_2, we can get d(H_1,H_2)=a+b. There are many possible choices of pair (a, b). For simplicity, we add one more constraint a=b such that we can solve for *a* directly. To sum up, we get the following equations:

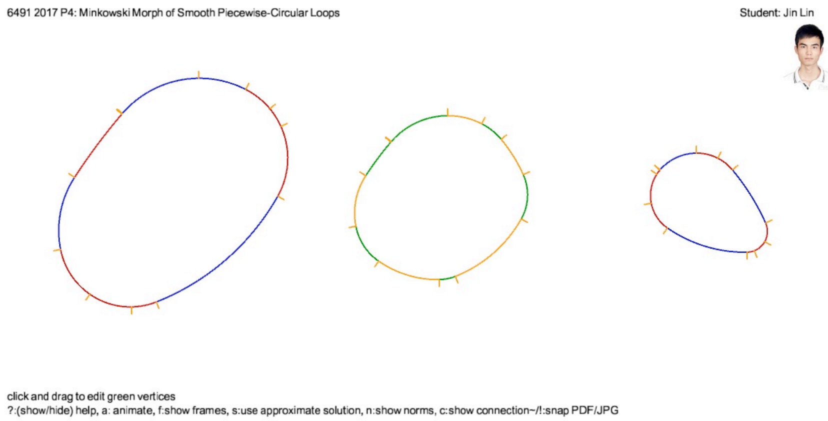
H_1= A + aT_1; H_2 = C – aT_2; P = H_1 + 1/2 H_1 H_2; d(H_1,H_2)=2a



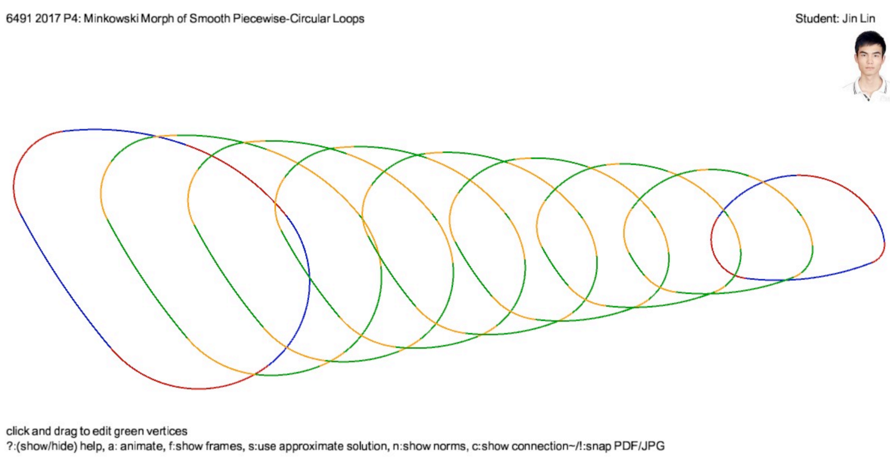
Step 3. Get the location of H_1, H_2 and P by solving the equations in Step 2. Then we get triangle (A, H_1, P) and triangle (P, H_2, C). Thus, we can draw the arcs controlled by these triangles.

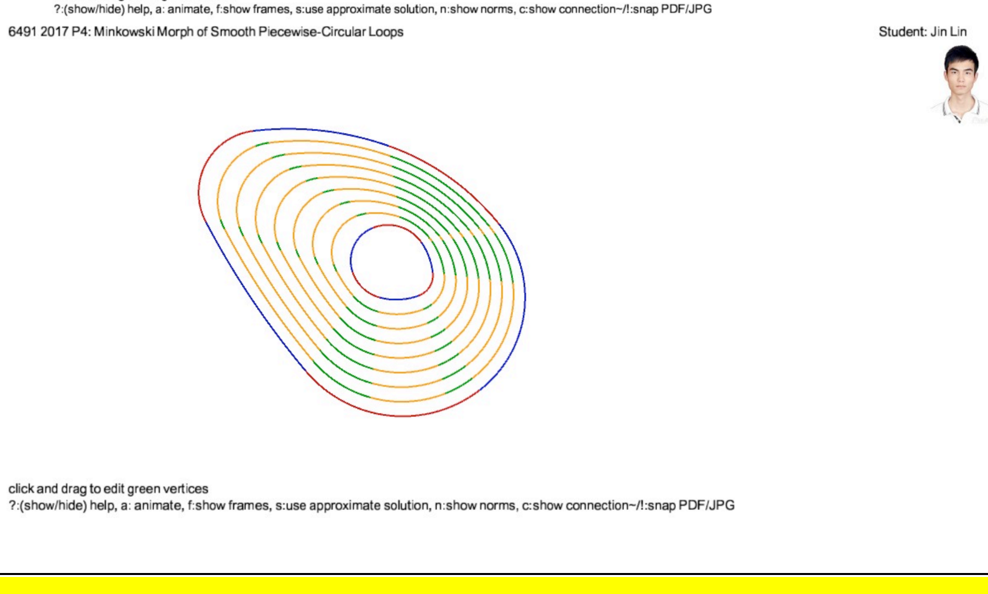
Step 4. Choose other point sets and repeat again.

Then we obtain 2 PCCs (non-concave). Since these PCCs are convex, we can sort each PCC's arcs based on their starting points' normal, and then split one PCC with the other (see picture below). After splitting them with each other, we have two PCCs with same number of arcs and each pair of arcs have the same normal pencil. When given time t, we can LERP each pair of arcs and connect them to construct a new PCC.



7 Stroboscopic superposition

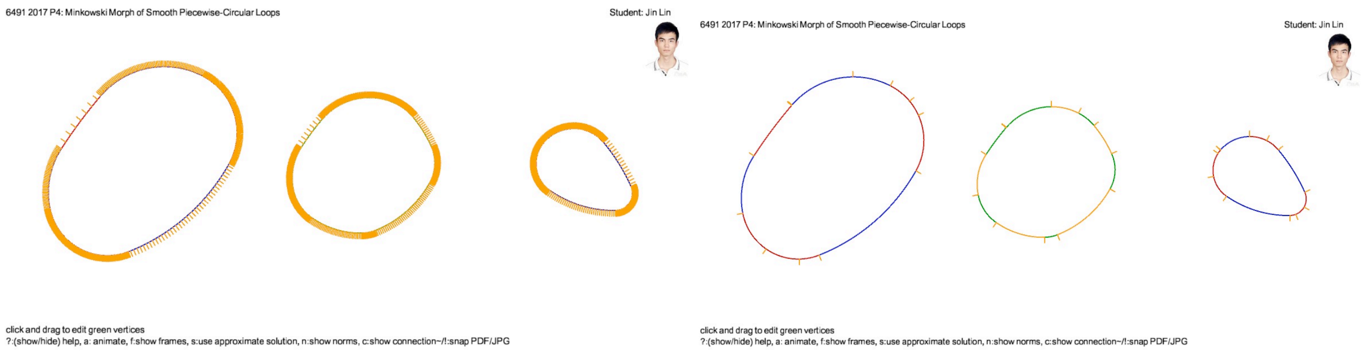




8 Extra credit extension: approximate solution

Before I applying “Splitting” method, I first use a kind of approximate solution by iterating each PCC over 360 orientations of normal. For each iteration, we can generate a normal value and find the relevant point on each PCC that has the same normal value, and then animate the LERP between these corresponding points. Comparing to “Splitting” method, it produces a similar result, but more slowly.

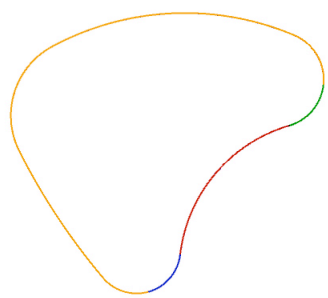
Approximate solution (left) VS Splitting (right)



9 Extra credit extension: non-convex SPCCs

When adding concave arc, things become far more complicate. For simplicity, I just realize one simple case: there is only 1 concave arc on PCC1 and no concave arc on PCC2.

I first compute the 2 normal value N1 and N2 of this concave arc’s starting and ending point, and then use these 2 normal to split PCC1 into 4 parts:



- A1: Orange part, which consists of convex arcs with normal pencil $[0, N1] \cup (N2, 2\pi]$;
- A2: Green part, which consists of convex arcs with normal pencil [N1, N2];
- A3: Red part, which consists of 1 concave arc with normal pencil [N1, N2];
- A4: Blue part, which consists of convex arcs with normal pencil [N1, N2].

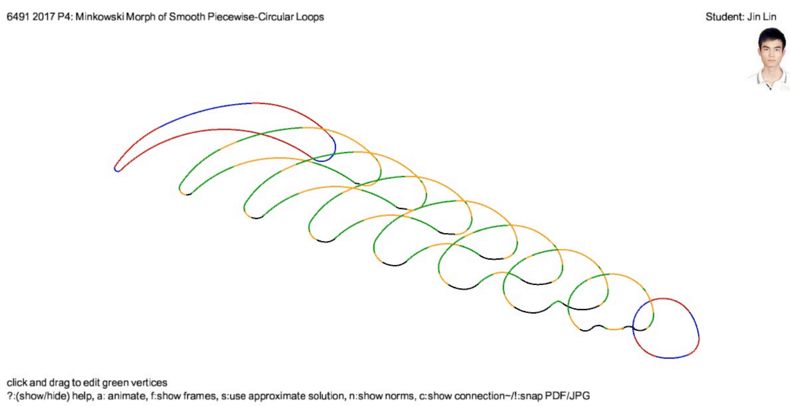
And split PCC2 into 4 parts:

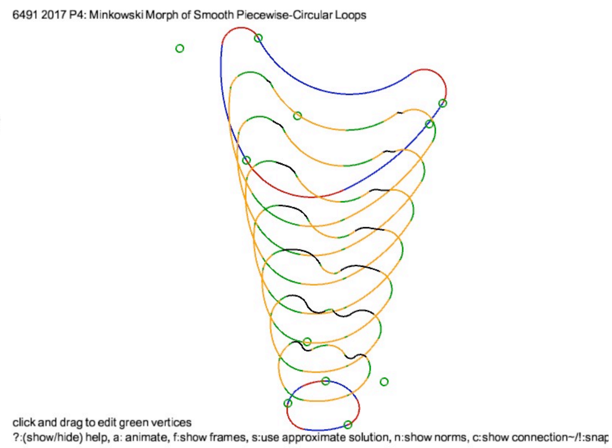
- B1: Consists of convex arcs with normal pencil $[0, N1] \cup (N2, 2\pi)$;
- B2: A point with normal value N1;
- B3: Consists of convex arcs with normal pencil [N1, N2].
- B4: A point with normal value N2;

And then applying Minkowski Morph to each pair of parts: A1-B1, A2-B2, A3-B3, A4-B4.

Since part A2 and part A4 morphs to 1 point separately, there will be some gaps between each morphing part, so I create some extra bi-arcs to fill these gaps (black arcs, see picture below):

Result:


6491 2017 P4: Minkowski Morph of Smooth Piecewise-Circular Loops
Student: Jin Lin
click and drag to edit green vertices
?:(show/hide) help, a: animate, f:show frames, s:use approximate solution, n:show norms, c:show connection~/!:snap PDF/JPG


6491 2017 P4: Minkowski Morph of Smooth Piecewise-Circular Loops
click and drag to edit green vertices
?:(show/hide) help, a: animate, f:show frames, s:use approximate solution, n:show norms, c:show connection~/!:snap