21

Techniques for Formation Movement Using Steering Circles

Stephen Bjore

21.1 Introduction

Moving formations around open terrain is fairly easy. However, it becomes more difficult to generate a path such that the formation ends at a specific point in a specified orientation, given the limitation that formations can only turn so quickly. A solution to this problem is presented in this chapter, and is an extension of the idea of using steering circles from Chris Jurney's GDC presentation [Jurney et al. 07]. The solution can be broken into two parts:

1. Generate the path to follow.
2. Navigate the formation along the path.

It's worth noting here that the first part of generating the path isn't limited to formations. It can be used for individual characters, vehicles, or any other moving object for which a steering circle can be defined. The second part is specific to formations and describes two different techniques for moving the formation along the path.

21.2 Generate the Path

Using two steering circles, one based on the current position of the formation and one based on the target position, we can calculate the path the formation needs to take.

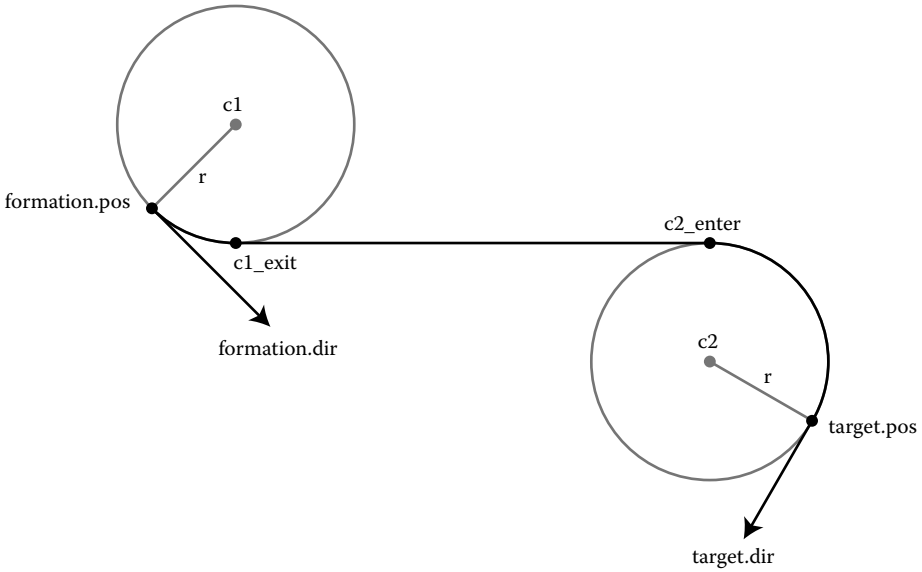

Figure 21.1

The final path starts at *formation.pos,* facing *formation.dir,* and ends at *target.pos,* facing *target.dir. c1* and *c2* show the centers of the steering circles. *c1_exit* is where the path breaks away from the starting circle and *c2_enter* is where the path joins the ending steering circle.

Figure 21.1 shows the most important information, as well as an example of what our final path could look like.

When we first begin generating the path, we start with five pieces of information: the current position of the formation (*formation.pos*), the current orientation (*formation.dir*), the target position (*target.pos*), the target orientation (*target.dir*), and the radius of the steering circles (or the turn radius of the formation, *r*). Based on this data, we need to calculate four additional values:

- *c1*, the center point of the starting steering circle.
- *c1_exit*, the point where the formation will break away from the starting circle.
- *c2*, the center point of the ending steering circle.
- *c2_enter*, the point where the formation will join the ending circle.

21.2.1 Calculating c1 and c2

The first step towards calculating the path is to generate the steering circles that will be used at the start and end points. To do this, we need to calculate the vector *target.pos–formation.pos,* which we will refer to as *dirVec*.

Next, we will calculate the center point of starting circle, *c1*. For this, we need the perpendicular vector of *formation.dir* in the same direction as *dirVec*. We can calculate this by taking the dot product of both perpendiculars of *formation.dir* with *dirVec*. We will use the perpendicular with the positive result, labeling it *formation.perp*. Scale *formation.perp* to have length equal to *r*, and then add it to *formation.pos* to get *c1*. The center point for

the ending circle, *c2*, is calculated in the same way, using *–dirVec* (instead of *dirVec*) and the perpendicular vectors of *target.dir*. The vector with the positive dot product will be referred to as *target.perp*.

The only exception is when the distance between *c1* and *c2* is less than *2r* (i.e., the steering circles are overlapping). The solution in this case is to invert both *formation.perp* and *target.perp*. This will cause the formation to steer in the opposite direction, thereby giving it enough space to turn. For example, if we originally used the right-hand perpendicular of *formation.dir*, we will use the left-hand perpendicular instead, essentially flipping the steering circle to the other side of *formation.dir*.

21.2.2 Calculating c1_exit and c2_enter

The goal of this section, finding *c1_exit* and *c2_enter*, has two different cases that we need to consider. In order to determine which case we have, we first need to look at whether *position.perp* and *target.perp* are the left or right perpendiculars of *position.dir* and *target.dir*. For brevity, we will say *formation.perp* is equal to "Left" if it is the left-hand perpendicular of *formation.dir*; otherwise it is "Right." Likewise, *target.perp* is equal to "Left" if it is the left-hand perpendicular of *target.dir*; otherwise it is "Right."

The first case is for when *formation.perp* and *target.perp* fall on opposite sides (e.g., *formation.perp* equals *Right* and *target.perp* equals *Left)*. In this instance, our goal is to calculate the angles where the points *c1_exit* and *c2_enter* are on the circumference of the two circles, relative to the x-axis (these angles are *a3* and *b3* in Figure 21.2). Once we have those angles, we can then calculate the two points.

The second case is when *formation.perp* and *target.perp* are on the same side. This case doesn't require us to calculate any angles, but instead relies on the observation that the important angles involved are all 90 degrees.

21.2.2.1 Calculation if Formation.Perp and Target.Perp Are on Opposite Sides

In the case where *formation.perp* is not on the same side as *target.perp*, our goal is to calculate the angles *a3* and *b3*. We will get into the details momentarily, but it should be noted that the calculation for *a3* and *b3* will change slightly, depending on which sides *formation.perp* and *target.perp* are on, which is why there are two diagrams in Figure 21.2.

Before we start the calculations, we will make several observations. First, the line from *c1* to *c1_exit* and the line from *c2* to c2_*enter* are both perpendicular to the line between *c1_exit* and *c2_enter*. Second, the lines *c1 to c2* and *c1_exit to c2_enter* intersect each other at the midpoint of both lines. Third, we know the radius of the steering circles, *r*. And fourth, we are able to calculate the distance between *c1* and *c2,* which is labeled *d*. Looking at Figure 21.2, we can see that we now have two right-hand triangles, and that we know two sides of the triangles (one side is *r*, the other is *½*d)*. This means that we can calculate the angle *a1*: `a1 = acos(r/(1/2 * d))`. We can also calculate *a2* by finding the angle of the vector *c2-c1* relative to the x-axis.

For *a3,* the calculation we need to use will depend on the values of *formation.perp* and *target.perp*. If *formation.perp* is Right, then we can refer to Diagram A, and *a3* is calculated by adding *a1* to *a2*. Else, if *formation.perp* is Left, we refer to Diagram B, and *a3* can be calculated by subtracting *a1* from *a2*.

The calculation for *c2_enter* is very similar to the calculation for *c1_exit*. The angle b*1* is calculated using the exact same equation and values that we used to find *a1*. The angle *b2*

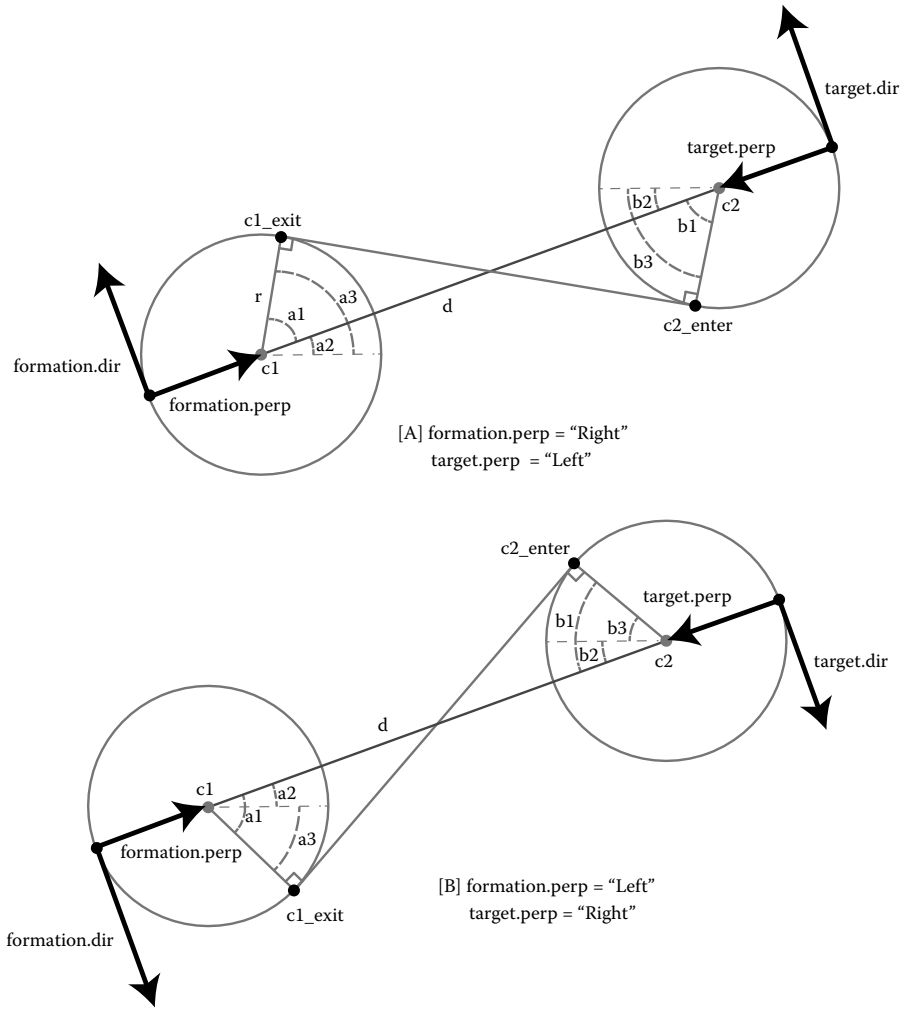

Figure 21.2

This diagram shows everything necessary to calculate the angles *a3* and *b3* when *formation.perp* and *target.perp* are on opposite sides. [A] shows the angles to be calculated when *formation.perp* is Right and *target.perp* is Left. [B] shows the angles for when *formation.perp* is Left and *target.perp* is Right.

is the angle of the vector *c1-c2* relative to the x-axis (unlike *a2*, which is the angle of *c2-c1* relative to the x-axis). The calculation for *b3* is also the same as the one we used for *a3*, and is simply *b2-b1* or *b2+b1*, depending on the values of *formation.perp* and *target.perp*.

Finally, now that we've calculated *a3* and *b3*, we can generate the points on the circles, *c1_exit* and *c2_enter*:

- *c1_exit(x,y)* = (c1.x + r * cos(a3), c1.y + r * sin(a3))
- *c2_enter(x,y)* = (c2.x + r * sin(b3), c2.y + r * sin(b3))

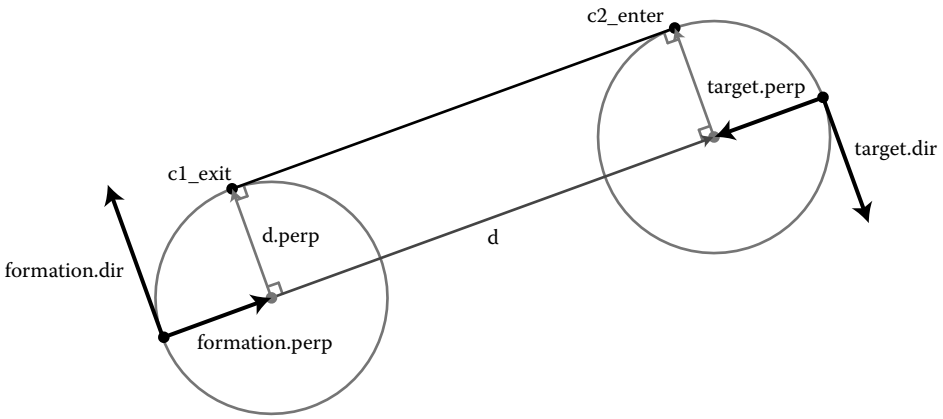

Figure 21.3

This shows what needs to be calculated when *formation.perp* and *target.perp* fall on the same side; in this case, they are both "Right."

21.2.2.2 Calculation if Formation.Perp and Target.Perp Are on the Same Side

In the case where *formation.perp* and *target.perp* both fall on the same side (meaning that they are both Right or Left), the calculation is a bit simpler. Here, we start by calculating the vector *c2-c1,* which we called *d*. As seen in Figure 21.3, if *formation.perp* is equal to Right, then we will use the left-hand perpendicular of *d*, labeled *d.perp*. Similarly, if *formation.perp* is equal to Left, then *d.perp* will be the right-hand perpendicular of *d*. In either case, once we have *d.perp*, add it to *c1* and *c2* to get *c1_exit* and *c2_enter*.

21.2.3 Generate the Points Along the Path

Finally, we can generate the points along the path for the formation to navigate by. The points will start at *formation.pos*, and move around the circle *c1* to *c1_exit*, move on to *c2_enter*, and finally around the circle *c2* until we arrive at *target.pos*.

The direction of travel around the circles is determined by *formation.perp* and *target.perp*. When they are equal to "Right," then we will generate points on the corresponding circle going around in the clockwise direction, and when they are "Left," we will generate points going around in the counter-clockwise direction.

21.3 Navigate the Formation

Moving a formation around in a way that looks reasonable requires the positions within the formation to be fluid. The following examples will keep the first row of the formation static, and the rows behind will follow in a couple of different ways. Here, we will look at two styles: the first style involves moving each unit within the formation towards the unit ahead of it, and the second style requires each unit to preserve its row by staying next to the units to its left and right.

Note that the points within the formation described here are intended to be pathfinding targets, not necessarily the actual locations of the units within the formation. This flexibility could allow units to go off and do other things, such as dealing with attacking

enemies, gathering nearby resources, or navigate around smaller obstacles. Once the unit has completed whatever subtask it had, it can then resume pathfinding to its target position within the formation.

21.3.1 "Column" Formation

This technique involves moving each unit within the formation towards the unit in front of it, while maintaining a set distance. While this will result in a fairly fluid look, and maintain a connection between each unit in a column, it does not preserve the rows.

The first step is to update the position and direction of the formation, based on the velocity of the formation and the next point in the path. To keep the units of the first row in a straight line, their positions are calculated such that the line they form is perpendicular to the formation's updated direction, and is centered at the formation's position. Starting on the second row, calculate the direction between each target, and the target ahead of it in the same column. We then move the target in that direction until it's touching the unit ahead of it, repeating this process for the units in all of remaining rows in the formation.

21.3.2 "Band" Formation

For this movement style, the formation preserves the rows as it steers around corners. The first row is calculated in the same manner that was used for the *Column Formation* style.

Next, we need to determine whether the second row is turning left or right, based on the direction that the first row moved in. To do this, first calculate the direction from any unit in the second row to the unit in the same column of the first row, and then take the right-hand perpendicular, which we will call *rPerpendicular*. Next, take the dot product of the direction that the first row moved in with *rPerpendicular*. If that result is positive, the second row will be turning right, or else it will be turning left.

For the moment, let's assume that the second row is turning left. The next step will be to move the leftmost unit in the second row towards the unit ahead of it in the same column until they are touching. Next, starting with the second unit in the row, set each unit's position to be touching the unit to its left, moving in the direction of *rPerpendicular*.

If the row is turning right, there are only two minor differences. The main one is that we will want to calculate the position of the rightmost unit first, moving it towards the unit in front of it until they are touching. The other difference is that you will start with the second unit from the right, and will set each unit's position to be touching the unit to its right, moving in the direction of *-1*rPerpendicular*.

In either case, once we have the positions for all of the units in the second row, we can repeat this process for the remaining rows in the formation.

21.4 The Demo

A demo provided on the book's website (http://www.gameaipro.com) was created to be used as a proof-of-concept for the ideas presented in this paper. It was written in HTML5 and Javascript, with the intention of being as portable as possible. Nearly all of the logic for generating the path can be found within "main.js," while the logic for drawing, updating, and moving the formation and the units within the formation can be found in "formation.js."

21.5 Conclusion

This article has shown that steering a formation to end in a specific place and direction, with the use of some simple linear algebra, is not difficult. By calculating two steering circles based on the formation's current position and the destination position, it's possible to generate a path for the formation to follow that will ensure that the formation will never need to make turns sharper than it is capable of.

Further development of this concept could include the ability for the formation to take obstacles into account, on both a large and small scale. For smaller obstacles, the intention is that the formation can largely ignore them. This is because it is intended that the pathfinding targets within the formation will deal with finding a way around anything small. Large-scale obstacles would need to be dealt with separately, but could potentially be handled by taking into account the size of the entire formation, and then perform pathfinding for the formation as a whole.

References

[Jurney et al. 07] C. Jurney and S. Hubick. "Dealing with destruction: AI from the trenches of company of heroes." *Game Developers Conference, 2007*. Available online (https://store.cmpgame.com/product.php?cat=24&id=2089).