SubtractRect doesn't always give you the exact difference

devblogs.microsoft.com/oldnewthing/20130918-00

September 18, 2013

Raymond Chen

The `SubtractRect` function takes a source rectangle and subtracts out the portion which intersects a second rectangle, returning the result in a third rectangle. But wait a second, the result of subtracting one rectangle from another need not be another rectangle. It might be an L-shape, or it might be a rectangle with a rectangular hole. How does this map back to a rectangle?

The documentation for `SubtractRect` says that the function performs the subtraction when they "intersect completely in either the x- or y-direction." But I prefer to think of it as the alternate formulation offered in the documentation: "In other words, the resulting rectangle is the bounding box of the geometric difference."

I was reminded of this subject when I saw some code that tried to do rectangle manipulation like this:

```
// Clip rcA to be completely inside rcB.
RECT rcSub;
// rcSub = the part of rcA that stick out beyond rcB
if (SubtractRect(&rcSub, &rcA, &rcB)) {
    // Remove that part from rcA
    SubtractRect(&rcA, &rcA, &rcSub);
}
```

If the rectangle `rcA` extends beyond `rcB` in more than one direction, then the geometric difference will not be rectangular, and the result of `SubtractRect` will be expanded to the bounding box of the difference, which means that it will return `rcA` again. And then the second line will subtract it all out, leaving the rectangle empty.

Oops.

What they really wanted was

```
// Clip rcA to be completely inside rcB.
IntersectRect(&rcA, &rcA, &rcB);
```

Raymond Chen

Follow