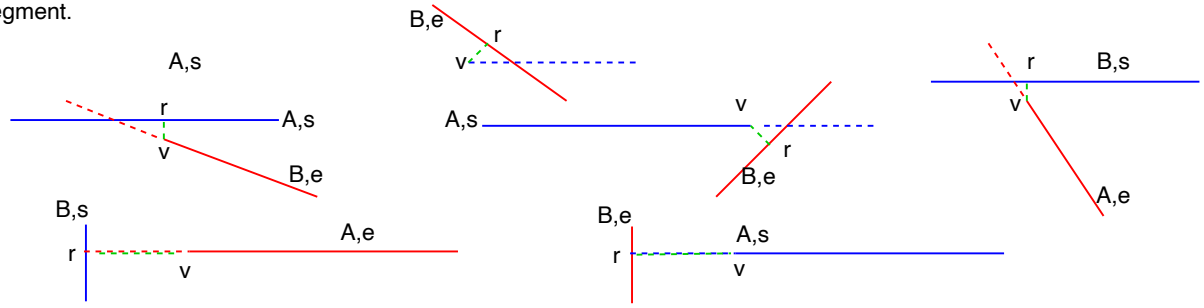
s is segment 1 (blue), **e** is segment 2 (red)

What we want to prove is:
{1} EXISTS (u: point_2d):
point_on_segment?(u, s) AND point_on_segment?(u, e)
{2} EXISTS (v: (segment_endpoint?(s))), (r: point_2d):
point_on_segment?(r, e) AND norm(r - v) <= norm(p - q)
{3} EXISTS (v: (segment_endpoint?(e))), (r: point_2d):
point_on_segment?(r, s) AND norm(r - v) <= norm(p - q)

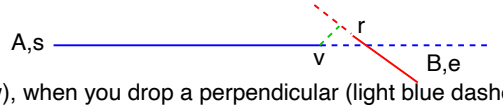
If there is an intersection, we satisfy the first consequent, so QED (case 0 above)
If not, then then we want to extend the segments until they intersect. This leads to three additional cases:

1. Only 1 line segment has to be extended to intersect the other.
2. Both line segments have to be extended to intersect the other.
3. The line segments are parallel and so extending them will do no good.

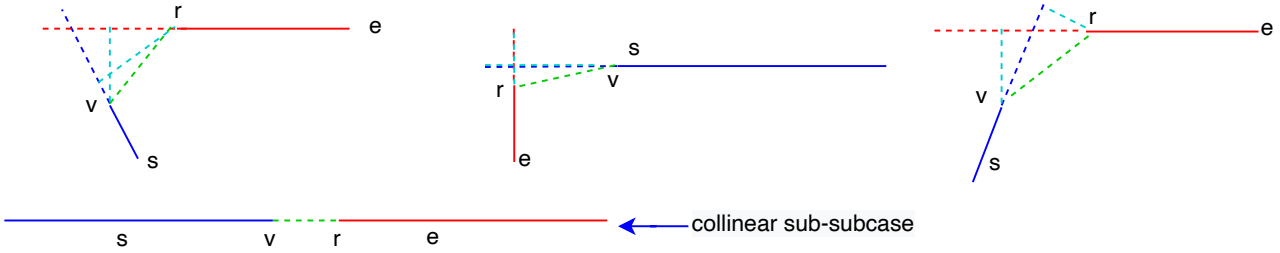
For case 1 (below), we'll call the line segment that has to be extended A, and the other segment B. The end point from which A has to be extended will be point v in consequent 2 or 3 above. A segment from that endpoint perpendicular to B is then drawn (green dashed line) so that it intersects B. There are two subcases. For case 1a, the dropped perpendicular intersects B. Where it intersects will be point r in that same consequent. This forms a right triangle, and moving either end point of the perpendicular segment will result in a longer segment.



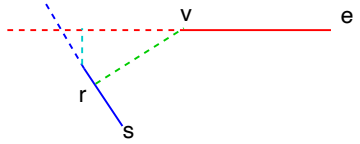
For case 1b (below), the dropped perpendicular does not intersect B. The endpoint of B closest to the dropped perpendicular is the nearest point to the endpoint of A from which the perpendicular was dropped.



For case 2, there are two subcases. For case 2a (below), when you drop a perpendicular (light blue dashed line) to the end points where each line segment is extended, those perpendiculars hit the extended portion of the other line segment (unless the line segments are collinear) rather than the line segment itself. In this subcase, the closest points will be the two end points (light green dashed line).



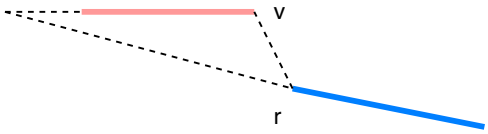
For case 2b (below), when you drop a perpendicular from one of the end points where a line segment is extended to the other line segment, that parpendicular crosses the other line segment. This can only happen when the angle between the line segments is acute, and it is not possible for both end points to have this property as a subcase of case 2.

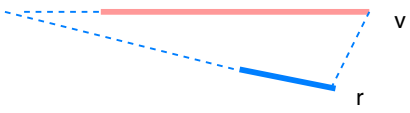


For case 3, there are two subcases. For case 3a (below), there is some overlap. We pick an endpoint that is involved in the overlap (there must be at least 2), and then drop a perpendicular segment to the other segment. We then substitute into either consequent 2 or 3, depending on which endpoint is selected.

For case 3b (below), there is no overlap. We pick whichever pair of endpoints is closest to each other, and substitue into consequent 2 (we could equally pick consequent 3) accordingly.

Additional subcases of 2a are possible, with different endpoints being nearest.



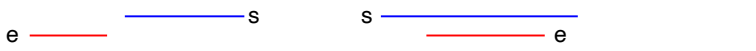


parallel_or_extending_crosses

parallel?(s)(e) **OR** collinear?(s, e) **OR** extending_s_crosses_e(s, e) **OR** extending_s_crosses_e(e, s)

parallel_seg_with_no_xing_imp_other_seg_has_0_or_2_xing

find_perp_ray_crossing(s, e, s`p1) = None
find_perp_ray_crossing(s, e, s`p2) = None

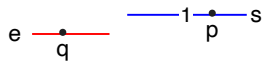


find_perp_ray_crossing(e, s, e`p1) = None
find_perp_ray_crossing(e, s, e`p2) = None

OR

find_perp_ray_crossing(e, s, e`p1) /= None
find_perp_ray_crossing(e, s, e`p2) /= None

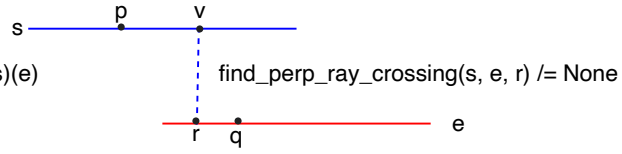
parallel_nonoverlapping_segs_closest_at_endpts.[1-4]



parallel?(s)(e)

NOT overlapping?(s, e)

parallel_overlapping_segs_closest_at_perps



parallel?(s)(e)

find_perp_ray_crossing(s, e, r) /= None

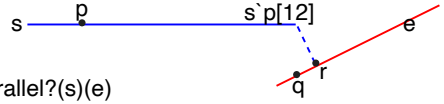
norm(val(find_perp_ray_crossing(s, e, r)) - r) <= norm(p - q)

nonoverlapping_collinear_closest_at_endpoints.[1-4]

collinear?(s, e)

dropped_perp_after_extension_intersect.[12]

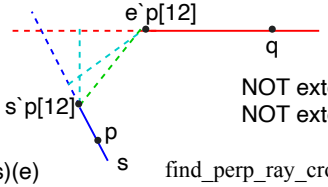
extending_s_crosses_e(s, e)



NOT parallel?(s)(e)
NOT collinear?(s, e)

find_perp_ray_crossing(s, e, s`p[12]) = r /= None

dropped_perp_with_no_extension_no_intersect.[12].[12]



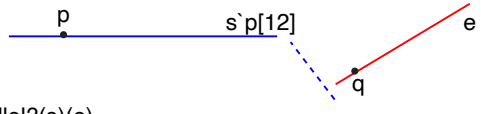
NOT extending_s_crosses_e(s, e)
NOT extending_s_crosses_e(e, s)

NOT parallel?(s)(e)
NOT collinear?(s, e)

find_perp_ray_crossing(s, e, s`p[12]) = None
find_perp_ray_crossing(e, s, e`p[12]) = None

dropped_perp_after_extension_no_intersect_endpts_closest.[12]

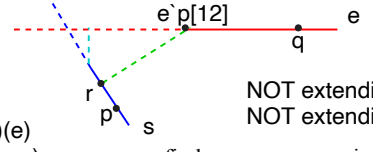
extending_s_crosses_e(s, e)



NOT parallel?(s)(e)
NOT collinear?(s, e)

find_perp_ray_crossing(s, e, s`p[12]) = None

dropped_perp_from_e_with_no_extension_intersect.[12]



NOT extending_s_crosses_e(s, e)
NOT extending_s_crosses_e(e, s)

NOT parallel?(s)(e)
NOT collinear?(s, e)

find_perp_ray_crossing(e, s, e`p[12]) = r /= None

e`p1`x * e`p1`x + e`p1`y * e`p1`y + e`p2`x * e`p2`x + e`p2`y * e`p2`y - 2 * (e`p1`x * e`p2`x) - 2 * (e`p1`y * e`p2`y)
=> sq(e`p2`x - e`p1`x) + sq(e`p2`y - e`p1`y)
=> len(e)

e`p1`x*e`p1`x + e`p1`y*e`p1`y + e`p2`x*s`p1`x + e`p2`y*s`p1`y - e`p1`x*e`p2`x - e`p1`x*s`p1`x - e`p1`y*e`p2`y - e`p1`y*s`p1`y
- e`p1`x*s`p2`x*t - e`p1`y*s`p2`y*t - e`p2`x*s`p1`x*t - e`p2`y*s`p1`y*t + e`p1`x*s`p1`x*t + e`p1`y*s`p1`y*t + e`p2`x*s`p2`x*t + e`p2`y*s`p2`y*t
=> (s`p1`x - e`p1`x*)*(e`p2`x - e`p1`x) + (s`p1`y - e`p1`y)*(e`p2`y - e`p1`y) + (s`p2`x - s`p1`x)*t*(e`p2`x - e`p1`x) + (s`p2`y - s`p1`y)*t*(e`p2`y - e`p1`y)
=> (s`p1`x*(1 - t) + t*s`p2`x - e`p1`x)*(e`p2`x - e`p1`x) + (s`p1`y*(1 - t) + t*s`p2`y - e`p1`y)*(e`p2`y - e`p1`y)

e`p1`x * pv`y + e`p1`y * s`p1`x + e`p2`x * s`p1`y + e`p2`y * pv`x - e`p1`x * s`p1`y - e`p1`y * pv`x - e`p2`x * pv`y - e`p2`y * s`p1`x - e`p1`x * s`p2`y * t
- e`p1`y * s`p1`x * t - e`p2`x * s`p1`y * t - e`p2`y * s`p2`x * t + e`p1`x * s`p1`y * t + e`p1`y * s`p2`x * t + e`p2`x * s`p2`y * t + e`p2`y * s`p1`x * t = 0
=>(e`p1`x)(pv`y - ((1-t)*s`p1`y + t*s`p2`y)) - (e`p1`y)(pv`x - ((1-t)*s`p1`x - t*s`p2`x)) - (e`p2`x)(pv`y - ((1-t)*s`p1`y + t*s`p2`y)) + (e`p2`y)(pv`x - ((1-t)*s`p1`x - t*s`p2`x))
=>(e`p1`x - e`p2`x)*(pv`y - ((1-t)*s`p1`y + t*s`p2`y)) - (e`p1`y - e`p2`y)*(pv`x - ((1-t)*s`p1`x +ex t*s`p2`x))

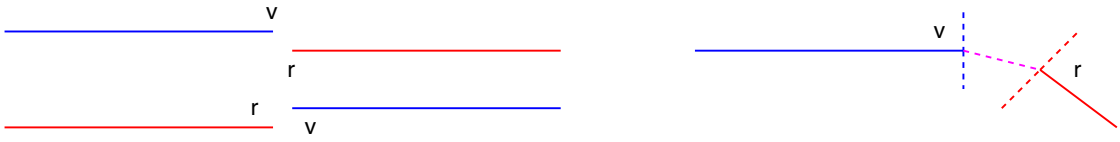
s is segment 1 (blue), **e** is segment 2 (red)

What we want to prove is:
{1} EXISTS (u: point_2d):
point_on_segment?(u, s) AND point_on_segment?(u, e)
{2} EXISTS (v: (segment_endpoint?(s))), (r: point_2d):
point_on_segment?(r, e) AND norm(r - v) <= norm(p - q)
{3} EXISTS (v: (segment_endpoint?(e))), (r: point_2d):
point_on_segment?(r, s) AND norm(r - v) <= norm(p - q)

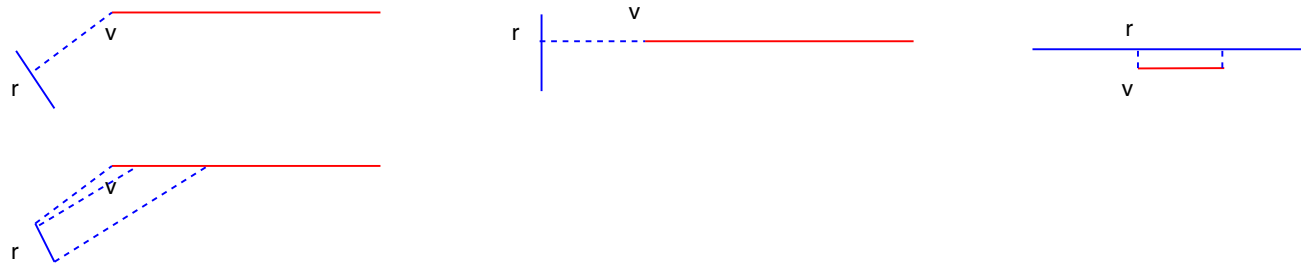
If there is an intersection, we satisfy the first consequent, so QED
If not, then for each endpoint of **s**, we want to draw lines perpendiculars to **e** from them, and see where (or if) those perpendiculars cross **e**. Similarly, for each endpoint of **e**, we want to draw lines perpendiculars to **s**.
We now have two cases:

1. No perpendiculars cross
2. At least one perpendicular crosses

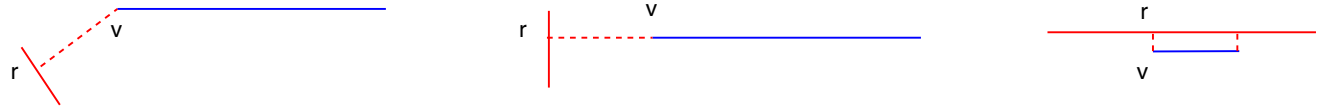
For case 1, where no perpendicular crosses, then we pairwise consider the two endpoints of **s** and the two endpoints of **e**, to see which endpoint of **s** is closest to which endpoint of **e**. We then instantiate consequent #2 with v = the endpoint of **s**, and r = the endpoint of **e**.
For case 2, we find the shortest of those perpendiculars to the corresponding line segment. If the shortest corresponds to an endpoint of **s** and a perpendicular to **e**, then we instantiate consequent #2 with v = the endpoint of **s**, and r = where the perpendicular crosses **e**. If the shortest corresponds to an endpoint of **e** and a perpendicular to **s**, then we instantitate consequent #3 with v = the endpoint of **e**, and r = where the perpendicular crosses **s**.
For the proof, we're looking at how to instantiate our consequents, with a focus on the 2nd and 3rd consequent. We first divide it up into the cases where there is no perpendicular crossing from either of the end points of **s**, and then further divide each of those where there is no perpendicular crossing from either of the end points of **e**. We'll consider these as four subcases, 2.1-2.4.
Case 2.1 is where there is no perpendicular crossing from any of the endpoints to the other segment.



For case 2.1, we find which end point pair is closest to each other, and insert that into consequent 2.
Case 2.2 is where there is perpendicular crossing from an end point of **e**, but there is not a perpendicular crossing from an end point of **s**:



For case 2.2, we find the end point of **e** that has a perpendicular crossing to **s** (or the closest to **s** if both cross) and insert that into consequent 3.
Case 2.3 is where there is perpendicular crossing from an end point of **s**, but there is not a perpendicular crossing from an end point of **e**:



For case 2.3, we find the end point of **s** that has a perpendicular crossing to **e** (or the closest to **e** if both cross) and insert that into consequent 2.
Case 2.4 is where there both **s** and **e** have perpendicular crossings to each other.

For case 2.4, we find the closest end point with a perpendicular crossing and if that end point is for **s**, we use consequent 2, and if that end point is for **e**, we use consequent 3.