1. **ALGORITHM**: *spatialIntersectionLine2DLine2D*
2. **INPUT**: *Line2D* objects *lineLhs* and *lineRhs*
3. **OUTPUT**: *Line2D* object *intersectionLine*
4. **BEGIN**
5. *Line2D* *intersectionLine := NULL*;
6. vector<seg*2D>* *intersectionLines := NULL*;
7. **if** *lineLhs = NULL* **or** *lineRhs = NULL* **then**
8. **return** *intersectionLine*;
9. **endif**
10. **else**
11. PlaneSweep *twoLinesPlaneSweep*(*lineLhs, lineRhs*)*;*
12. Initialize *previousHistory := nil;*
13. **while** twoLinesPlaneSweep.getObject() != none **and** twoLinesPlaneSweep.getStatus() = endOfNone
14. *twoLinesPlaneSweep.*selectFirst();
15. **if** *twoLinesPlaneSweep.*getObject() *= both* **and** *previousHistory = both* **then**
16. *intersectionLines.*push\_back(*twoLinesPlaneSweep.*getEvent());
17. **endif**
18. **if** *twoLinesPlaneSweep.*getObject() *= both* **then**
19. *previousHistory = both;*
20. **endif**
21. **else**
22. *previousHistory = nil;*
23. **endelse**
24. *twoLinesPlaneSweep.*selectNext();
25. **endwhile**
26. **endelse**
27. *intersectionLine = Line2D(intersectionLines);*
28. **return** *intersectionLine*;
29. **END** *spatialIntersectionLine2DLine2D*

The *spatialIntersectionLine2DLine2D* algorithm takes in two *Line2D* objects *lineLhs* and *lineRhs* and computes the spatial intersection between them and returns a *Line2D* object *intersectionLine* containing this intersection.

In line 5 and 6, we initialize the empty lineIntersection object and the lineIntersections vector (which is used to create the lineIntersection object).

In line 7, we first check with a simple if statement whether either of the two objects are NULL (or empty line objects). If so, there is obviously they do not have an intersection. Then we return the (empty/initialized to NULL) *intersectionLine* object. Otherwise, we proceed to lines 10-26.

Firstly, on line 11, we create the required PlaneSweep object “*twoLinesPlaneSweep*” and pass to its constructor the two line objects.

Then in line 12, we initialize a variable previousHistory with the value ‘nil’. This either holds the value ‘both’ or ‘nil’ depending on the value of ‘object’ encountered previously in the while loop (line 13-25).

Next, line 14, we call the selectFirst function, which would in turn update the value of *object* with *none, both, first* or *second ,* and the value of *status* with *endOfNone, endOfFirst, endOfSecond* or *endOfBoth.*

Only if the value of both *object* and *previousHistory* is *both* would the line 16 execute. Obviously this would not be true for the first execution (since *previousHistory* isinitializedwith *nil*).

What this means (if line 16 executes), is that the current pointers for both the Line2D objects under consideration have the same line segments (seg2D). If so, add this segment to the *intersectionLines vector* being calculated.

The selectNext function call updates the respective pointer(s) of the object(s) according to the value in *object.* This is captured by line 24.

We do this as long as we don’t reach the end of either of the objects, which is captured by line 13.

After the complete traversal of the either/both of the two objects, we return the calculated *intersectionLine* object (line 18) containing the required intersection lines.