**//**We can write a small Geo-Library to put those helper methods which is

//**GeometryHelper.** I add utility code as initial to deal with floating-point

//issues.

**public** **class** GeometryHelper

{

const **double** EquityTolerance = 0.000000001d;

**private** **static** **bool** IsEqual(**double** d1, **double** d2)

{

**return** Math.Abs(d1-d2) <= EquityTolerance;

}

….

}

//And Before we start, it is better to write our own data types

**public** **class** ConvexPolygon2D

{

**public** Point2D[] Corners;

**public** ConvexPolygon2D(Point2D[] corners)

{

Corners = corners;

}

}

**public** **class** Point2D

{

**public** **double** X;

**public** **double** Y;

**public** Point2D(**double** x, **double** y)

{

X = x;

Y = y;

}

}

//So, for given points (x1, y1) and (x2, y2), we can calculate A, B, C values:

**//A = y2-y1**

**//B = x1-x2**

**//C = A\*x1+B\*y1**

//After we calculate A, B, C values for two line segments **intersection**

**//point can be calculated as follow**;

**float** det = A1\*B2 - A2\*B1

**if**(det == 0){

//Lines are parallel

} **else**{

**float** x = (B2\*C1 - B1\*C2)/det

**float** y = (A1\*C2 - A2\*C1)/det

}

Let’s Write the code for the function;

//math logic from http://www.wyrmtale.com/blog/2013/115/2d-line-intersection-in-c

**public** **virtual** Point2D GetIntersectionPoint(Point2D l1p1, Point2D l1p2, Point2D l2p1, Point2D l2p2)

{

**double** A1 = l1p2.Y - l1p1.Y;

**double** B1 = l1p1.X - l1p2.X;

**double** C1 = A1 \* l1p1.X + B1 \* l1p1.Y;

**double** A2 = l2p2.Y - l2p1.Y;

**double** B2 = l2p1.X - l2p2.X;

**double** C2 = A2 \* l2p1.X + B2 \* l2p1.Y;

//lines are parallel

**double** det = A1 \* B2 - A2 \* B1;

**if** (IsEqual(det, 0d))

{

**return** **null**; //parallel lines

}

**else**

{

**double** x = (B2 \* C1 - B1 \* C2) / det;

**double** y = (A1 \* C2 - A2 \* C1) / det;

**bool** online1 = ((Math.Min(l1p1.X, l1p2.X) < x || IsEqual(Math.Min(l1p1.X, l1p2.X), x))

&& (Math.Max(l1p1.X, l1p2.X) > x || IsEqual(Math.Max(l1p1.X, l1p2.X), x))

&& (Math.Min(l1p1.Y, l1p2.Y) < y || IsEqual(Math.Min(l1p1.Y, l1p2.Y), y))

&& (Math.Max(l1p1.Y, l1p2.Y) > y || IsEqual(Math.Max(l1p1.Y, l1p2.Y), y))

);

**bool** online2 = ((Math.Min(l2p1.X, l2p2.X) < x || IsEqual(Math.Min(l2p1.X, l2p2.X), x))

&& (Math.Max(l2p1.X, l2p2.X) > x || IsEqual(Math.Max(l2p1.X, l2p2.X), x))

&& (Math.Min(l2p1.Y, l2p2.Y) < y || IsEqual(Math.Min(l2p1.Y, l2p2.Y), y))

&& (Math.Max(l2p1.Y, l2p2.Y) > y || IsEqual(Math.Max(l2p1.Y, l2p2.Y), y))

);

**if** (online1 && online2)

**return** new Point2D(x, y);

}

**return** **null**; //intersection is at out of at least one segment.

}

**//Check if a given point is inside a convex polygon**

// taken from https://wrf.ecse.rpi.edu//Research/Short\_Notes/pnpoly.html

**public** **bool** IsPointInsidePoly(Point2D test, ConvexPolygon2D poly)

{

**int** i;

**int** j;

**bool** result = **false**;

**for** (i = 0, j = poly.Corners.Length - 1; i < poly.Corners.Length; j = i++)

{

**if** ((poly.Corners[i].Y > test.Y) != (poly.Corners[j].Y > test.Y) &&

(test.X < (poly.Corners[j].X - poly.Corners[i].X) \* (test.Y - poly.Corners[i].Y) / (poly.Corners[j].Y - poly.Corners[i].Y) + poly.Corners[i].X))

{

result = !result;

}

}

**return** result;

}

**//Finding Intersection Points of a line segment and given convex polygon**

**public** **virtual** Point2D[] GetIntersectionPoints(Point2D l1p1, Point2D l1p2, ConvexPolygon2D poly)

{

List<Point2D> intersectionPoints = new List<Point2D>();

**for** (**int** i = 0; i < poly.Corners.Length; i++)

{

**int** next = (i + 1 == poly.Corners.Length) ? 0 : i + 1;

Point2D ip = GetIntersectionPoint(l1p1, l1p2, poly.Corners[i], poly.Corners[next]);

**if** (ip != **null**) intersectionPoints.Add(ip);

}

**return** intersectionPoints.ToArray();

}

Some edge cases, such as **two overlapping corners** or **intersection on a corner** can cause some **duplicates corner added to the polygon**. We can easily get rid of these with such small utility function:

**private** **void** AddPoints(List<Point2D> pool, Point2D[] newpoints)

{

foreach (Point2D np in newpoints)

{

**bool** found = **false**;

foreach (Point2D p in pool)

{

**if** (IsEqual(p.X, np.X) && IsEqual(p.Y, np.Y))

{

found = **true**;

**break**;

}

}

**if** (!found) pool.Add(np);

}

}

**//We are ready to write the main algorithm**

**public** ConvexPolygon2D GetIntersectionOfPolygons(ConvexPolygon2D poly1, ConvexPolygon2D poly2)

{

List<Point2D> clippedCorners = new List<Point2D>();

//Add the corners of poly1 which are inside poly2

**for** (**int** i = 0; i < poly1.Corners.Length; i++)

{

**if** (IsPointInsidePoly(poly1.Corners[i], poly2))

AddPoints(clippedCorners, new Point2D[] { poly1.Corners[i] });

}

//Add the corners of poly2 which are inside poly1

**for** (**int** i = 0; i < poly2.Corners.Length; i++)

{

**if** (IsPointInsidePoly(poly2.Corners[i],poly1))

AddPoints(clippedCorners, new Point2D[]{ poly2.Corners[i]});

}

//Add the intersection points

**for** (**int** i = 0, next = 1; i < poly1.Corners.Length; i++, next = (i + 1 == poly1.Corners.Length) ? 0 : i + 1)

{

AddPoints(clippedCorners, GetIntersectionPoints(poly1.Corners[i], poly1.Corners[next], poly2));

}

**return** new ConvexPolygon2D(OrderClockwise( clippedCorners.ToArray()));

}