
polybooljs

Boolean operations on polygons (union, intersection, difference, xor).

NOTE: I am no longer maintaining this, but I have created a TypeScript version, available here: @velip-so/polybool.

Features

1. Clips polygons for all boolean operations
2. Removes unnecessary vertices
3. Handles segments that are coincident (overlap perfectly, share vertices, one inside the other, etc)
4. Uses formulas that take floating point irregularities into account (via configurable epsilon)
5. Provides an API for constructing efficient sequences of operations
6. Support for GeoJSON "Polygon" and "MultiPolygon" types (experimental)

Resources

- Demo + Animation
- Companion Tutorial
- Based somewhat on the F. Martinez (2008) algorithm: Paper, Code

Ports

Other kind souls have ported this library:

- Java port by @the3deers
- Java port by @Menecats
- .NET port by @idormenco
- Flutter/Dart port by @mohammedX6
- Python port by @KaivnD
- Please make a ticket if you'd like to be added here :-)

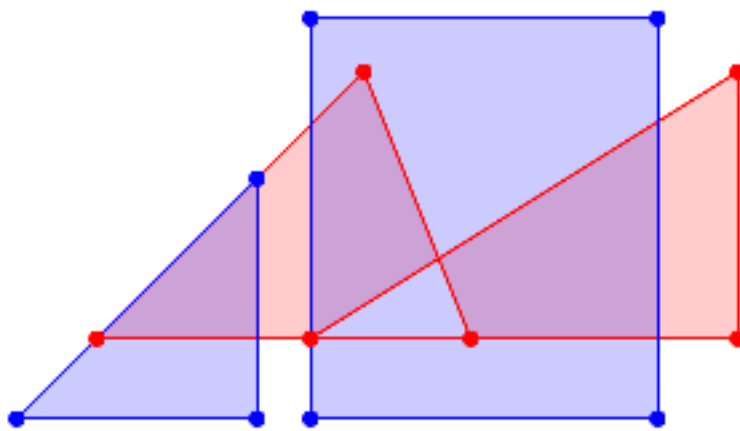
Installing

```
npm install polybooljs
```

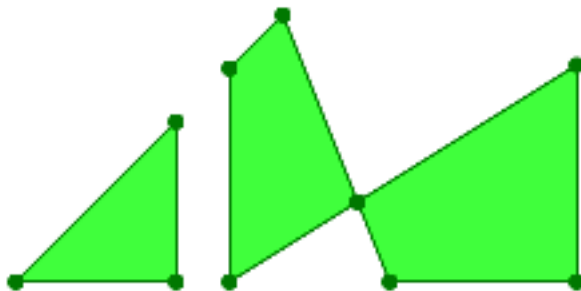
Or, for the browser, look in the `dist/` directory for a single file build. When included on a page, it will expose the global `PolyBool`.

Example

```
1 var PolyBool = require('polybooljs');
2 PolyBool.intersect({
3   regions: [
4     [[50,50], [150,150], [190,50]],
5     [[130,50], [290,150], [290,50]]
6   ],
7   inverted: false
8 }, {
9   regions: [
10    [[110,20], [110,110], [20,20]],
11    [[130,170], [130,20], [260,20], [260,170]]
12  ],
13  inverted: false
14 });
15 ==> {
16   regions: [
17     [[50,50], [110,50], [110,110]],
18     [[178,80], [130,50], [130,130], [150,150]],
19     [[178,80], [190,50], [260,50], [260,131.25]]
20   ],
21   inverted: false
22 }
```



Intersect



Basic Usage

```
1 var poly = PolyBool.union      (poly1, poly2);
2 var poly = PolyBool.intersect  (poly1, poly2);
3 var poly = PolyBool.difference (poly1, poly2); // poly1 - poly2
4 var poly = PolyBool.differenceRev(poly1, poly2); // poly2 - poly1
5 var poly = PolyBool.xor        (poly1, poly2);
```

Where `poly1`, `poly2`, and the return value are Polygon objects, in the format of:

```
1 // polygon format
2 {
3   regions: [ // list of regions
4     // each region is a list of points
```

```
5     [[50,50], [150,150], [190,50]],
6     [[130,50], [290,150], [290,50]]
7 ],
8 inverted: false // is this polygon inverted?
9 }
```

GeoJSON (experimental)

There are also functions for converting between the native polygon format and GeoJSON.

Note: These functions are currently **experimental**, and I'm hoping users can provide feedback. Please comment in this issue on GitHub – including letting me know if it's working as expected. I don't use GeoJSON, but I thought I would take a crack at conversion functions.

Use the following functions:

```
1 var geojson = PolyBool.polygonToGeoJSON(poly);
2 var poly    = PolyBool.polygonFromGeoJSON(geojson);
```

Only "Polygon" and "MultiPolygon" types are supported.

Core API

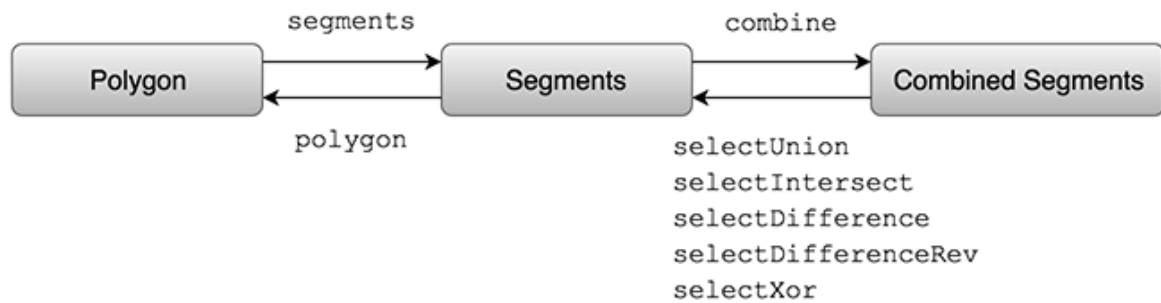
```
1 var segments = PolyBool.segments(polygon);
2 var combined = PolyBool.combine(segments1, segments2);
3 var segments = PolyBool.selectUnion(combined);
4 var segments = PolyBool.selectIntersect(combined);
5 var segments = PolyBool.selectDifference(combined);
6 var segments = PolyBool.selectDifferenceRev(combined);
7 var segments = PolyBool.selectXor(combined);
8 var polygon  = PolyBool.polygon(segments);
```

Depending on your needs, it might be more efficient to construct your own sequence of operations using the lower-level API. Note that `PolyBool.union`, `PolyBool.intersect`, etc, are just thin wrappers for convenience.

There are three types of objects you will encounter in the core API:

1. Polygons (discussed above, this is a list of regions and an `inverted` flag)
2. Segments
3. Combined Segments

The basic flow chart of the API is:



You start by converting Polygons to Segments using `PolyBool.segments(poly)`.

You convert Segments to Combined Segments using `PolyBool.combine(seg1, seg2)`.

You select the resulting Segments from the Combined Segments using one of the selection operators `PolyBool.selectUnion(combined)`, `PolyBool.selectIntersect(combined)`, etc. These selection functions return Segments.

Once you're done, you convert the Segments back to Polygons using `PolyBool.polygon(segments)`.

Each transition is costly, so you want to navigate wisely. The selection transition is the least costly.

Advanced Example 1

Suppose you wanted to union a list of polygons together. The naive way to do it would be:

```
1 // works but not efficient
2 var result = polygons[0];
3 for (var i = 1; i < polygons.length; i++)
4   result = PolyBool.union(result, polygons[i]);
5 return result;
```

Instead, it's more efficient to use the core API directly, like this:

```
1 // works AND efficient
2 var segments = PolyBool.segments(polygons[0]);
3 for (var i = 1; i < polygons.length; i++){
4   var seg2 = PolyBool.segments(polygons[i]);
5   var comb = PolyBool.combine(segments, seg2);
6   segments = PolyBool.selectUnion(comb);
7 }
8 return PolyBool.polygon(segments);
```

Advanced Example 2

Suppose you want to calculate all operations on two polygons. The naive way to do it would be:

```
1 // works but not efficient
2 return {
3   union      : PolyBool.union      (poly1, poly2),
4   intersect  : PolyBool.intersect  (poly1, poly2),
5   difference : PolyBool.difference (poly1, poly2),
6   differenceRev: PolyBool.differenceRev(poly1, poly2),
7   xor        : PolyBool.xor        (poly1, poly2)
8 };
```

Instead, it's more efficient to use the core API directly, like this:

```
1 // works AND efficient
2 var seg1 = PolyBool.segments(poly1);
3 var seg2 = PolyBool.segments(poly2);
4 var comb = PolyBool.combine(seg1, seg2);
5 return {
6   union      : PolyBool.polygon(PolyBool.selectUnion      (comb)),
7   intersect  : PolyBool.polygon(PolyBool.selectIntersect  (comb)),
8   difference : PolyBool.polygon(PolyBool.selectDifference (comb)),
9   differenceRev: PolyBool.polygon(PolyBool.selectDifferenceRev(comb)),
10  xor        : PolyBool.polygon(PolyBool.selectXor        (comb))
11 };
```

Advanced Example 3

As an added bonus, just going from Polygon to Segments and back performs simplification on the polygon.

Suppose you have garbage polygon data and just want to clean it up. The naive way to do it would be:

```
1 // union the polygon with nothing in order to clean up the data
2 // works but not efficient
3 var cleaned = PolyBool.union(polygon, { regions: [], inverted: false })
4 ;
```

Instead, skip the combination and selection phase:

```
1 // works AND efficient
2 var cleaned = PolyBool.polygon(PolyBool.segments(polygon));
```

Epsilon

Due to the beauty of floating point reality, floating point calculations are not exactly perfect. This is a problem when trying to detect whether lines are on top of each other, or if vertices are exactly the same.

Normally you would expect this to work:

```
1 if (A === B)
2   /* A and B are equal */;
3 else
4   /* A and B are not equal */;
```

But for inexact floating point math, instead we use:

```
1 if (Math.abs(A - B) < epsilon)
2   /* A and B are equal */;
3 else
4   /* A and B are not equal */;
```

You can set the epsilon value using:

```
PolyBool.epsilon(newEpsilonValue);
```

Or, if you just want to get the current value:

```
var currentEpsilon = PolyBool.epsilon();
```

The default epsilon value is 0.0000000001.

If your polygons are really really large or really really tiny, then you will probably have to come up with your own epsilon value – otherwise, the default should be fine.

If `PolyBool` detects that your epsilon is too small or too large, it will throw an error:

```
1 PolyBool: Zero-length segment detected; your epsilon is probably too
   small or too large
```

Build Log

The library also has an option for tracking execution of the internal algorithms. This is useful for debugging or creating the animation on the demo page.

By default, the logging is disabled. But you can enable or reset it via:

```
var buildLog = PolyBool.buildLog(true);
```

The return value is an empty list that will have log entries added to it as more API calls are made.

You can inspect the log by looking in the values:

```
1 buildLog.forEach(function(logEntry){
2   console.log(logEntry.type, logEntry.data);
3 });
```

Don't rely on the build log functionality to be consistent across releases.

You can disable the build log via:

```
PolyBool.buildLog(false);
```

You can get the current list (or **false** if disabled) via:

```
var currentLog = PolyBool.buildLog();
```