

Basic Geometry 2D

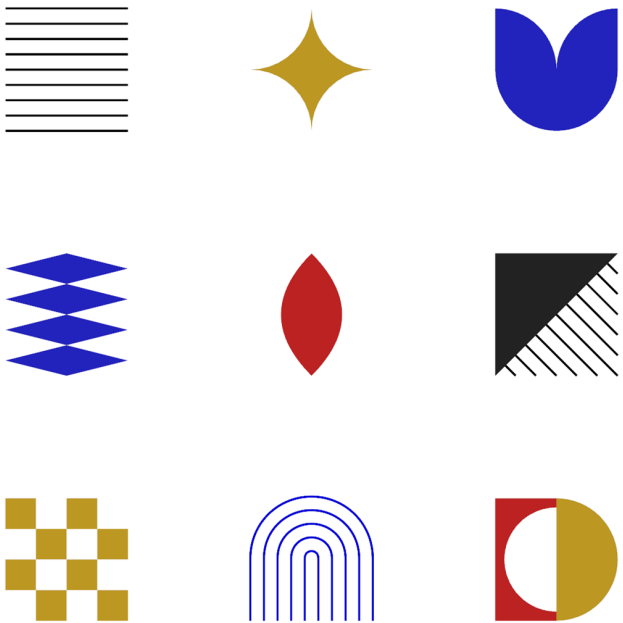





Content

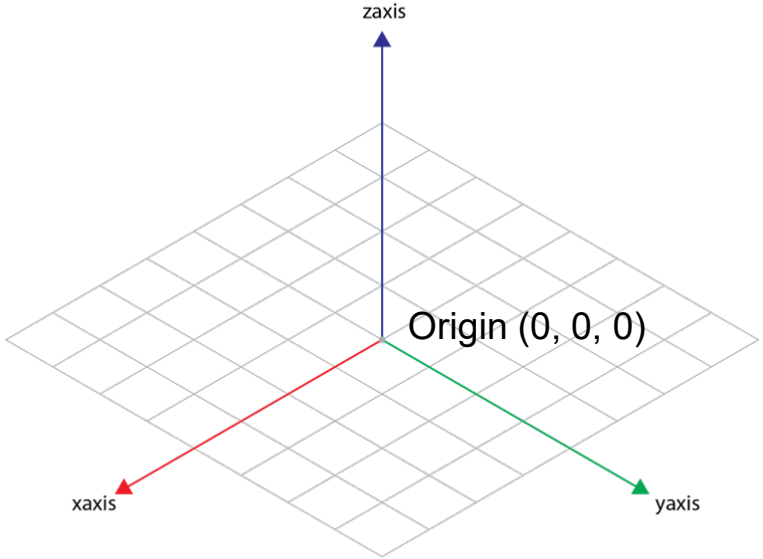
1. Points
2. Point Clouds
3. Vectors
4. Planes
5. Lines / Polylines
6. Circles / Arcs
7. Curves (Bezier, Splines, B-Splines, NURBS)

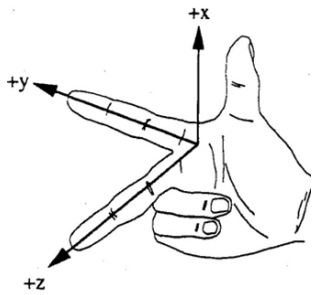
What is 2D Geometry?



With **2D geometry** we refer to shapes and objects that locally exist in **1D or 2D, but these shapes are embedded in a 3D space.**

World Coordinate System (WCF)

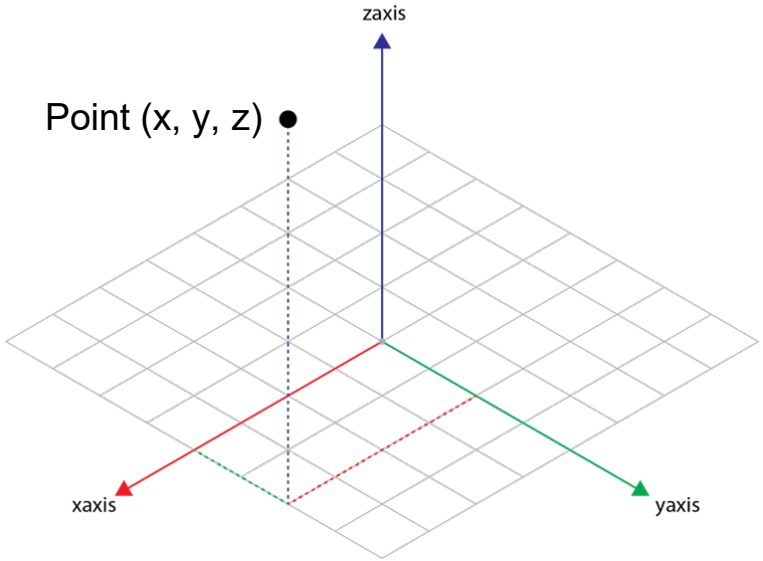




Right-hand rule

1. Point

Point(x: float, y: float, z: float)

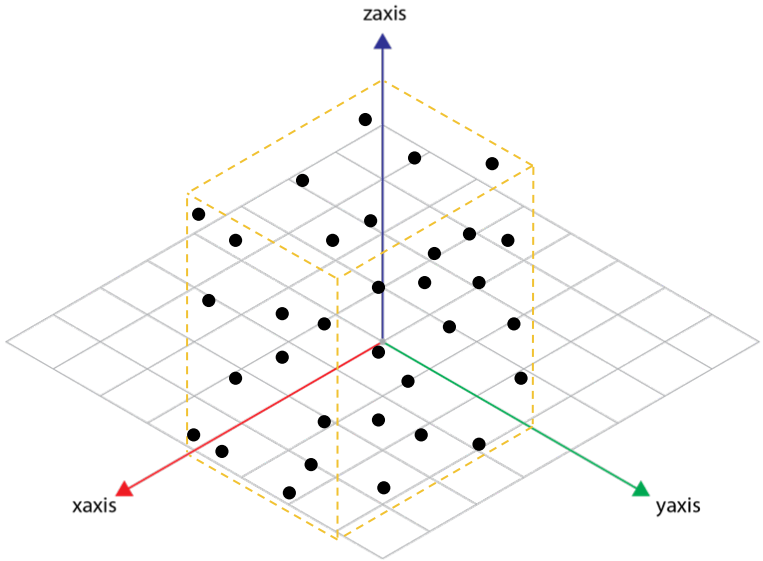


$$p = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- location

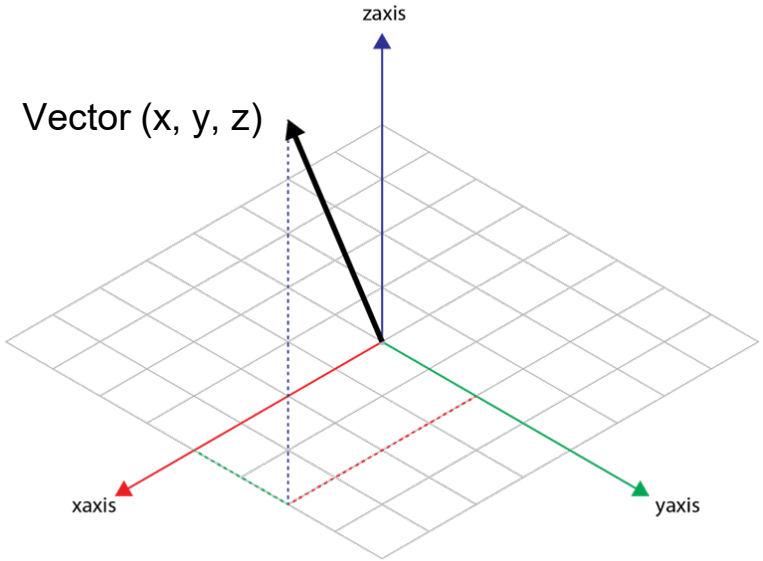
2. Point cloud

class **Rhino.Geometry**.**Pointcloud**(point: Point, xaxis: Vector, yaxis: Vector)



3. Vector

class **Rhino.Geometry**.**Vector3d**(x: float, y: float, z: float)

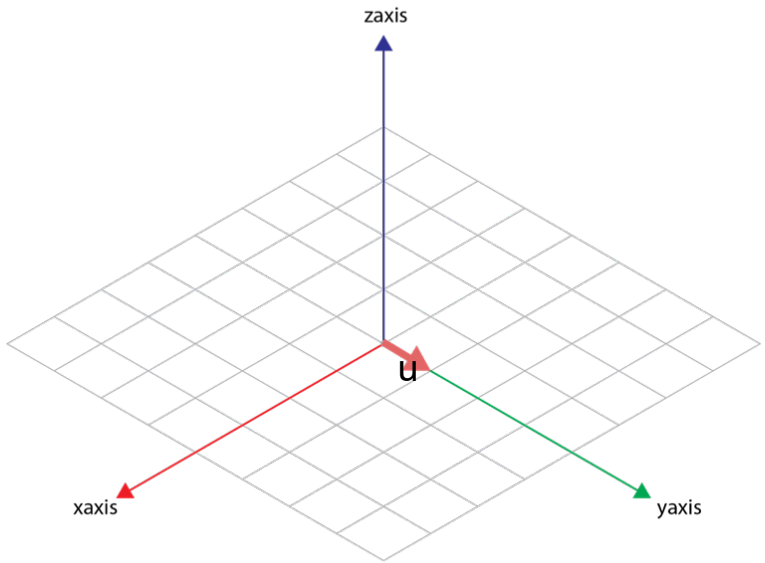


- magnitude ||v||
- direction ∠(v, v')

$$v = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

3. Vector

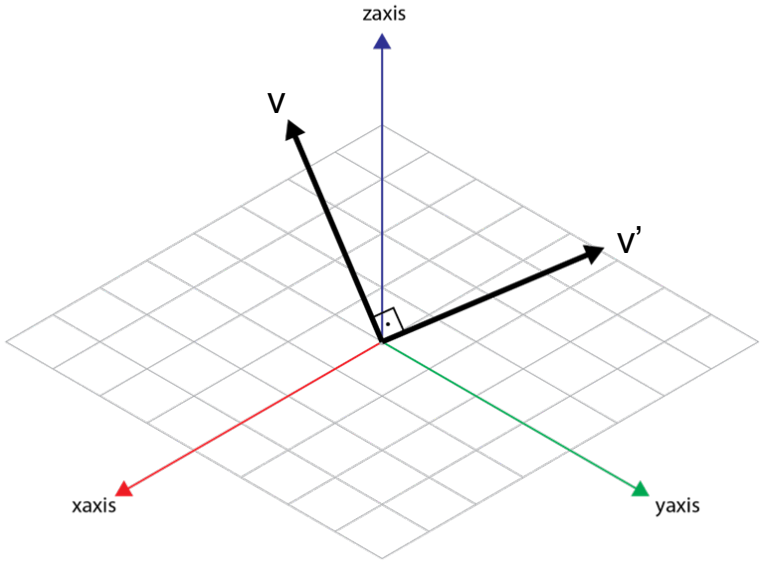
class **Rhino.Geometry**.**Vector3d**(x: float, y: float, z: float)



Unit vector is when ||u|| = 1

3. Vector

class **Rhino.Geometry**.**Vector3d**(x: float, y: float, z: float)

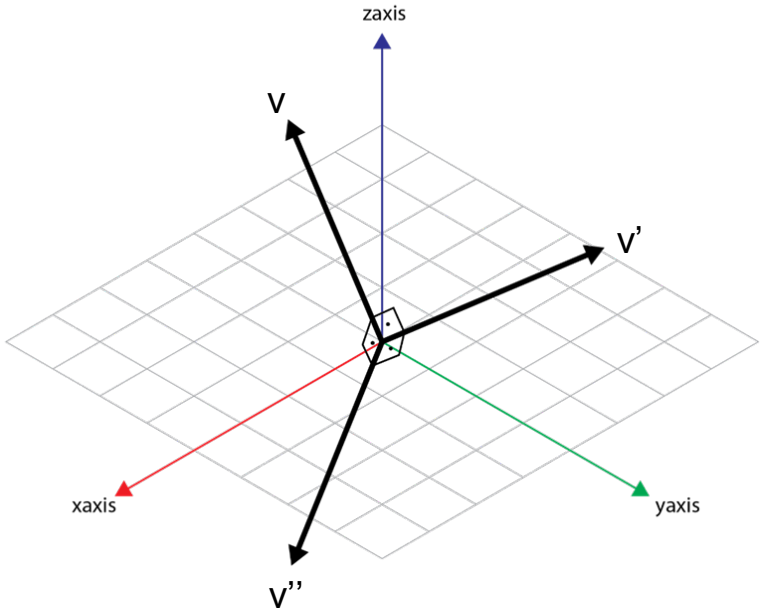


Orthogonal vectors are when ∠(v, v') = 90°

3. Vector

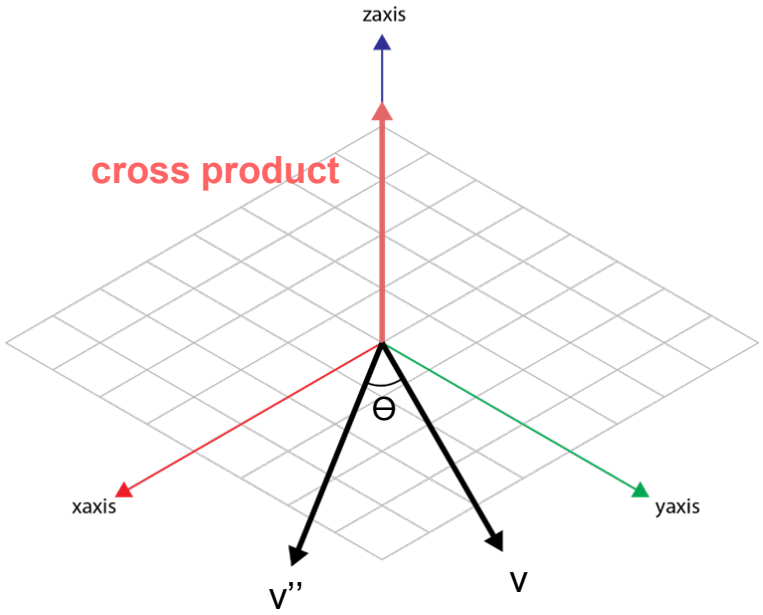
class **Rhino.Geometry**.**Vector3d**(x: float, y: float, z: float)

Orthonormal vectors are when ∠(v, v') = 90°, ∠(v, v'') = 90°, ∠(v', v'') = 90°



3. Vector

class **Rhino.Geometry**.**Vector3d**(x: float, y: float, z: float)

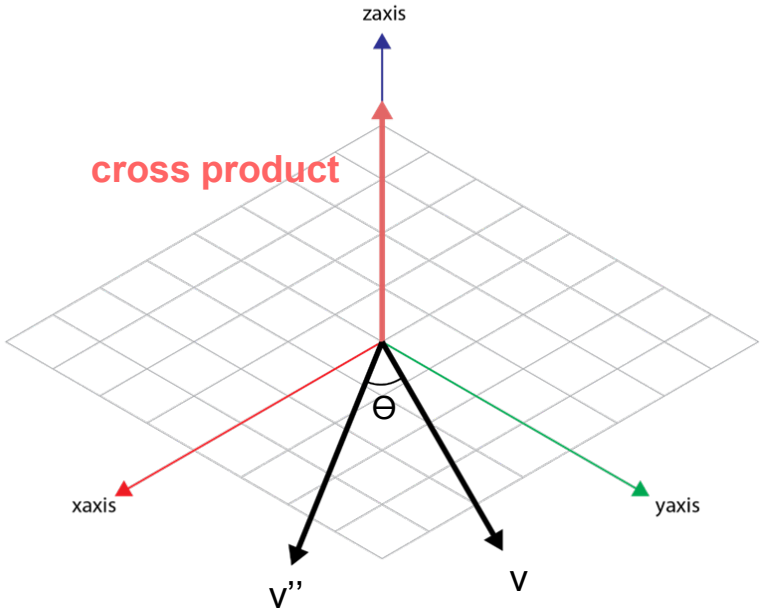


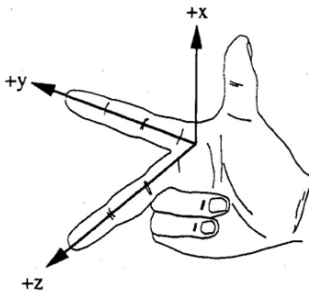
Cross product is a vector orthogonal to both given vectors.

cross product = v x v''

3. Vector

class **Rhino.Geometry**.**Vector3d**(x: float, y: float, z: float)



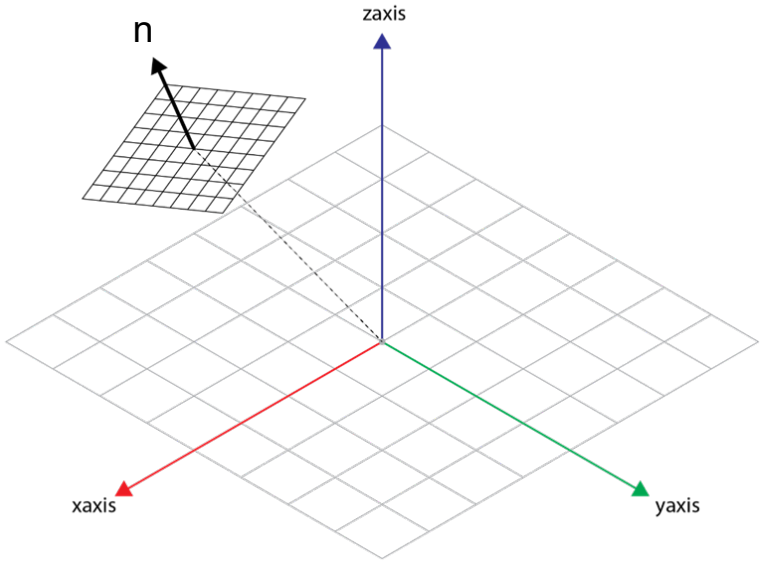


v'' x v ≠ v x v''

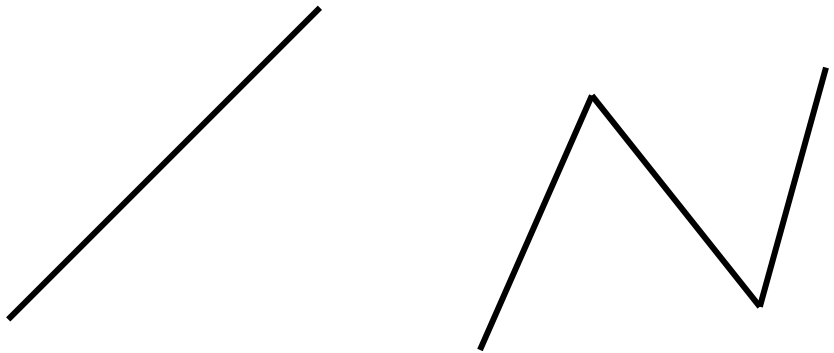
It is **anticommutative** and according to right-hand formula.

4. Plane

class **Rhino.Geometry.Plane**(origin: Point3d, normal: Vector3d)

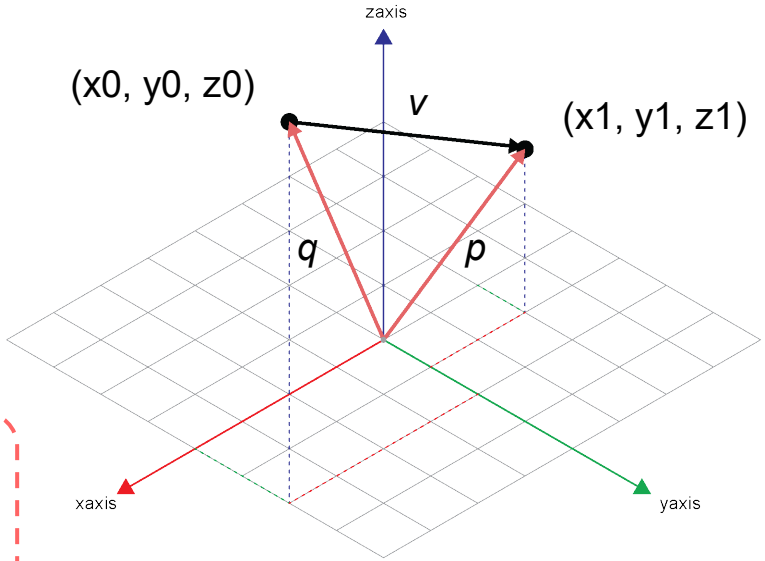


5. Line / Polyline



5. Line from six numbers

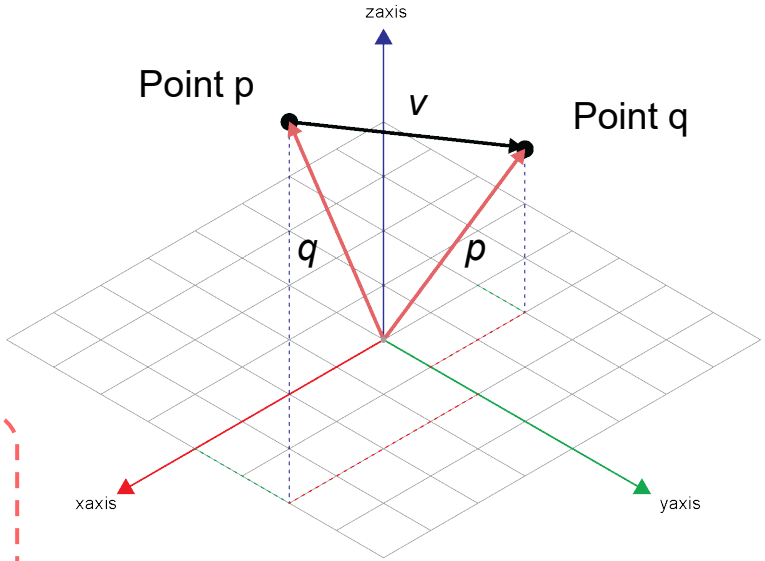
class **Rhino.Geometry**.**Line**(float *x0*, float *y0*, float *z0*, float *x1*, float *y2*, float *z3)*



Defined by **the coordinates** of the start and the end points. The **direction vector of the line** is **v = q − p**.

5. Line from two points

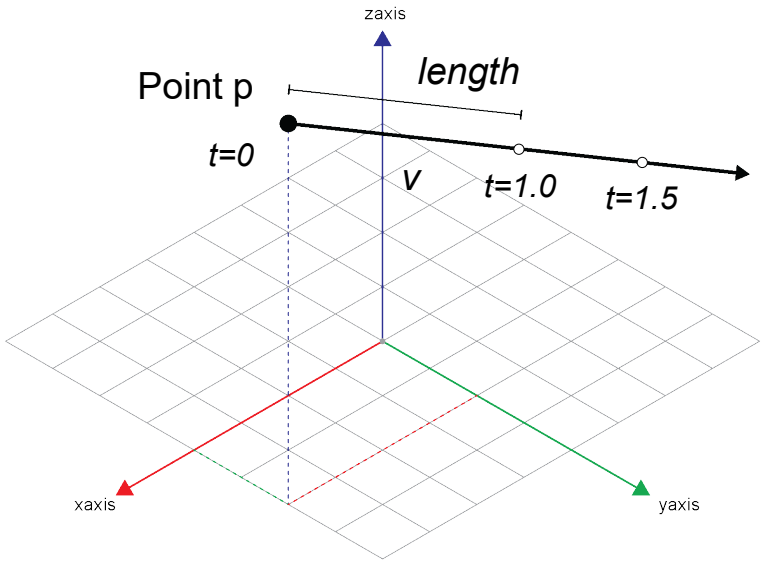
class **Rhino.Geometry**.**Line**(Point3d *p*, Point3d *q*)



Defined by points ***p*** and ***q***. The **direction vector of the line** is ***v* = *q* − *p***.

5. Line from point + direction + length

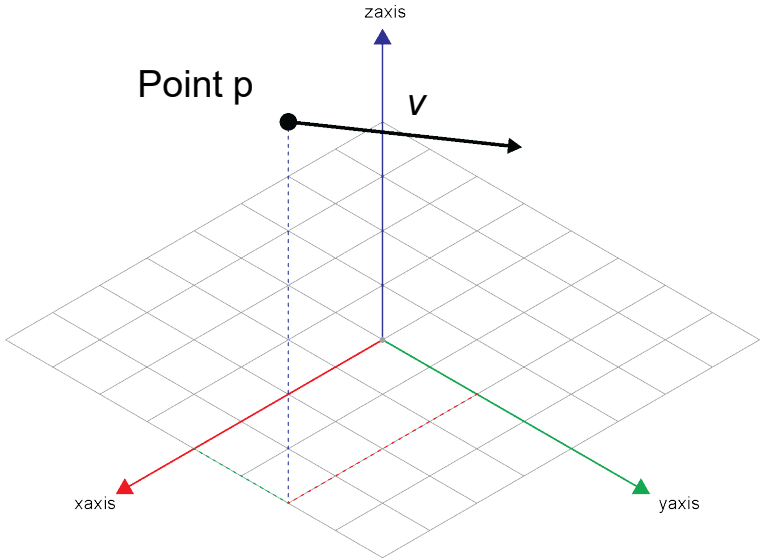
class **Rhino.Geometry**.**Line**(Point3d *p*, Vector3d *v*, float *length*)



Defined by a **start point *p*, direction vector *v*** and ***length*. Parametric representation: x = p + t·v**

5. Line from point + direction

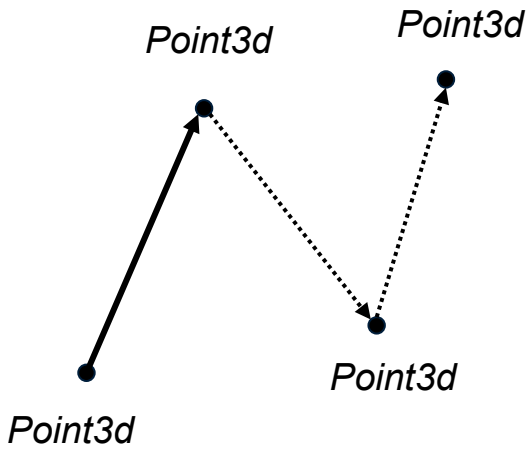
class **Rhino.Geometry**.**Line**(Point3d *p*, Vector3d *v*)



Defined by a **start point *p* and a span vector *v*.**

5. Polyline

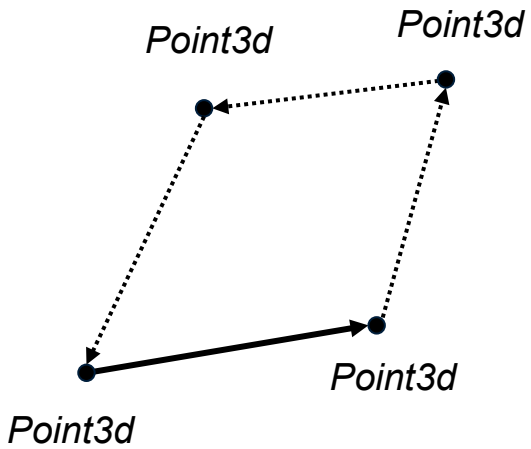
class **Rhino.Geometry**.**Polyline**(list[Point3d])



A **polyline** is a sequence of straight line segments.

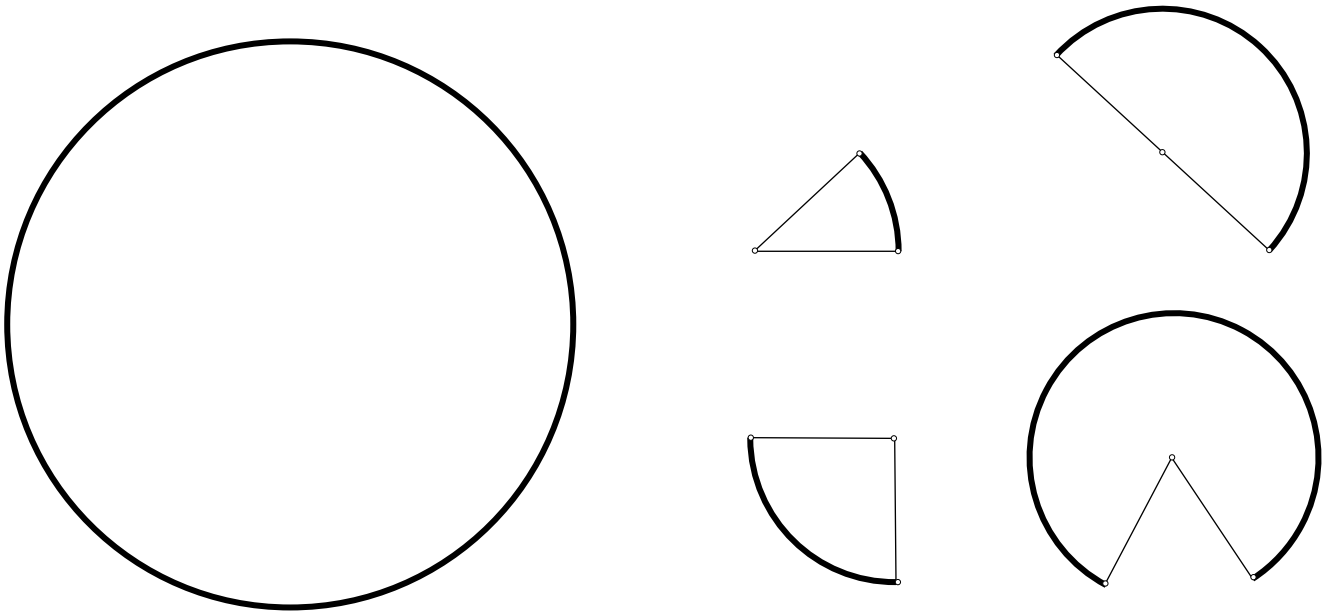
5. Polyline

class **Rhino.Geometry**.**Polyline**(list[Point3d])



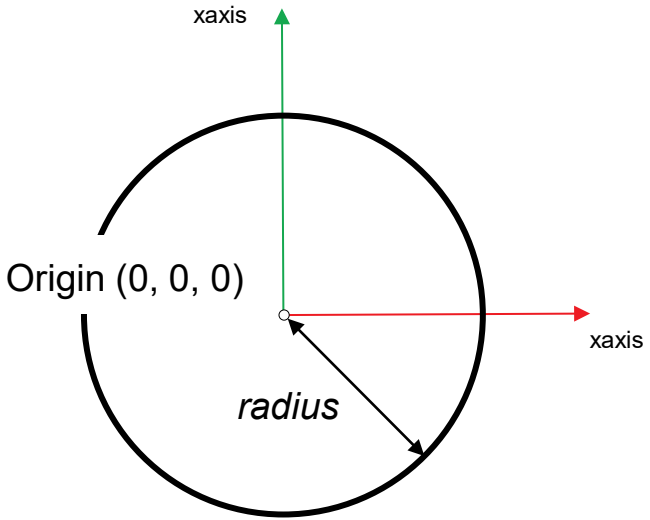
If the **first and last points coincide**; the polyline is **a closed polygon.**

6. Circles / Arcs



6. Circle

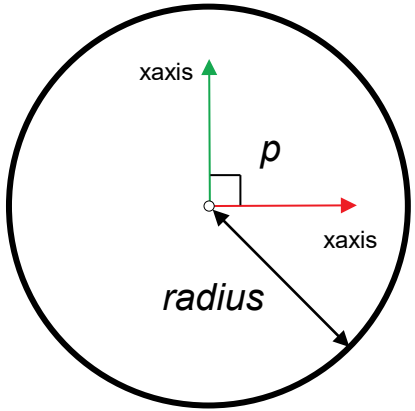
class **Rhino.Geometry**.**Circle**(float *radius*)



Defined by a **radius.**

6. Circle

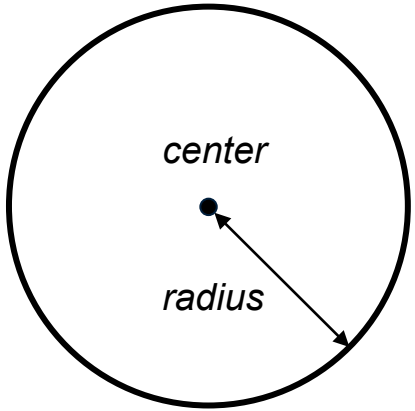
class **Rhino.Geometry**.**Circle**(Plane *p*, float *radius*)



Defined by **a plane** and **a radius.**

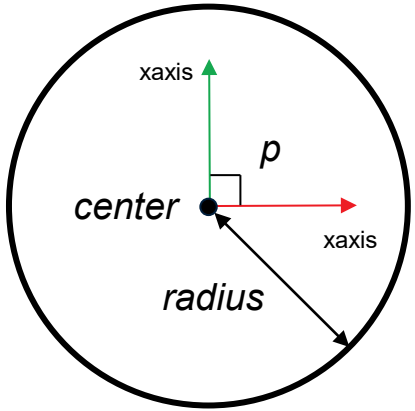
6. Circle

class **Rhino.Geometry**.**Circle**(Point3d *center*, float *radius*)



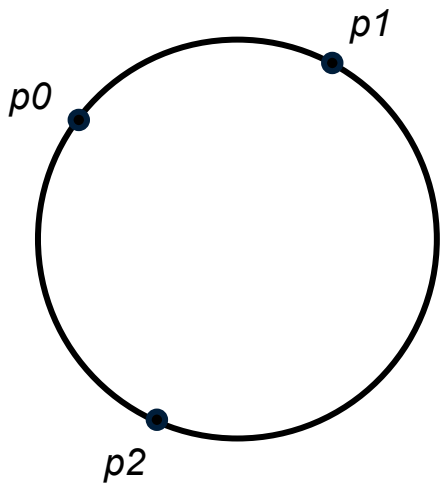
6. Circle

class **Rhino.Geometry**.**Circle**(Plane *p*, Point3d *center*, float *radius*)



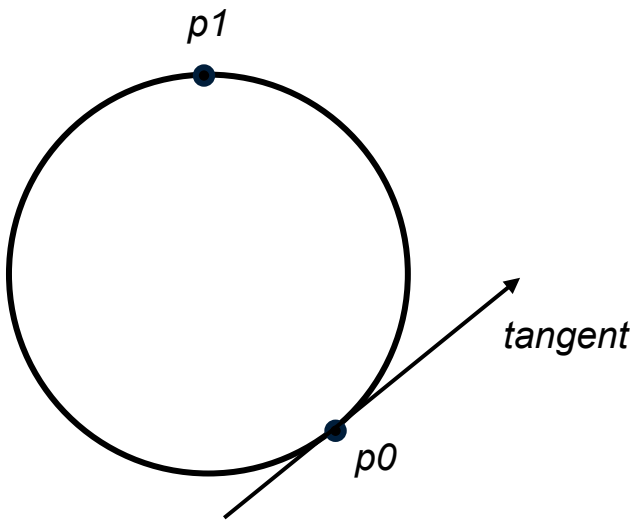
6. Circle

class **Rhino.Geometry**.**Circle**(Point3d *p0*, Point3d *p1,* Point3d *p2*)

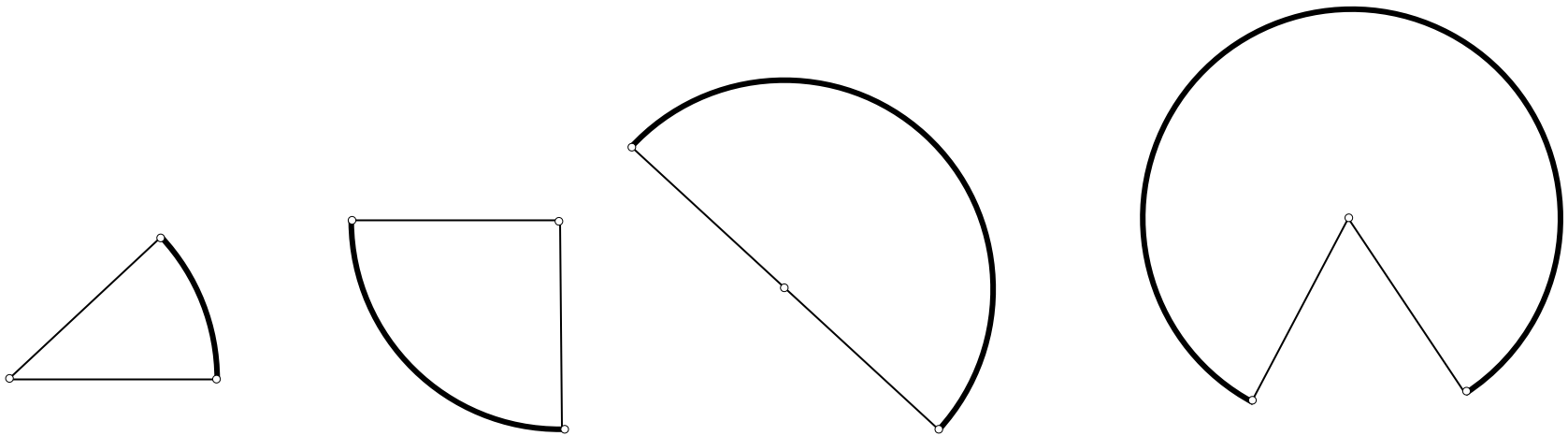


6. Circle

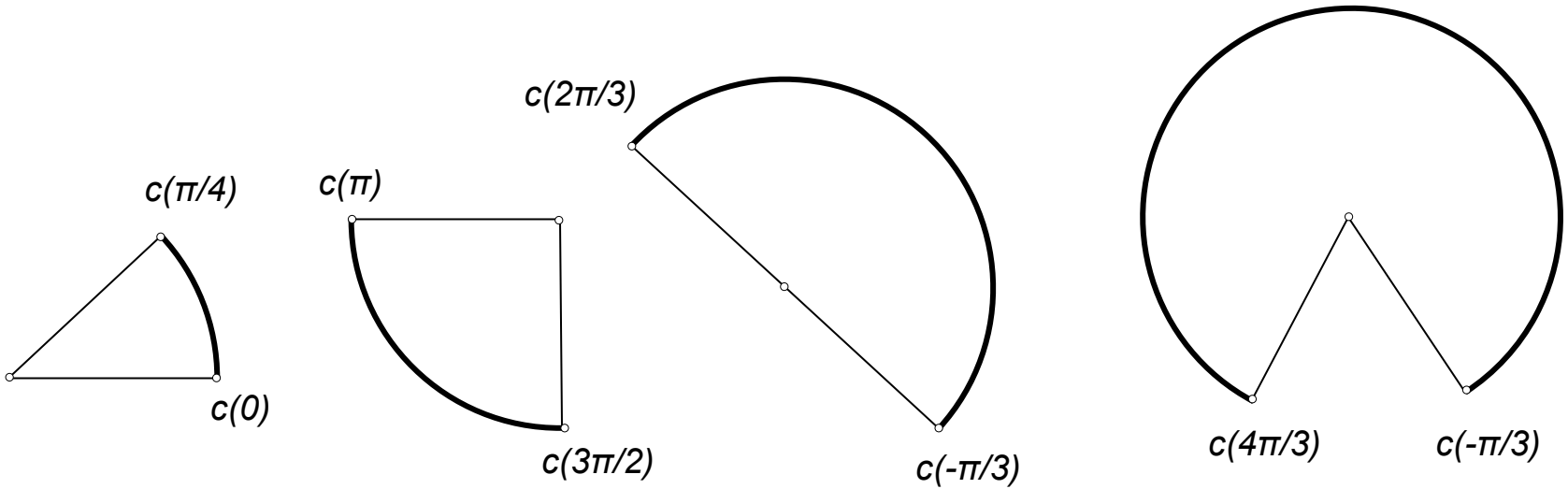
class **Rhino.Geometry**.**Circle**(Point3d *p0*, Vector3d *tangent,* Point3d *p1*)



6. Arcs

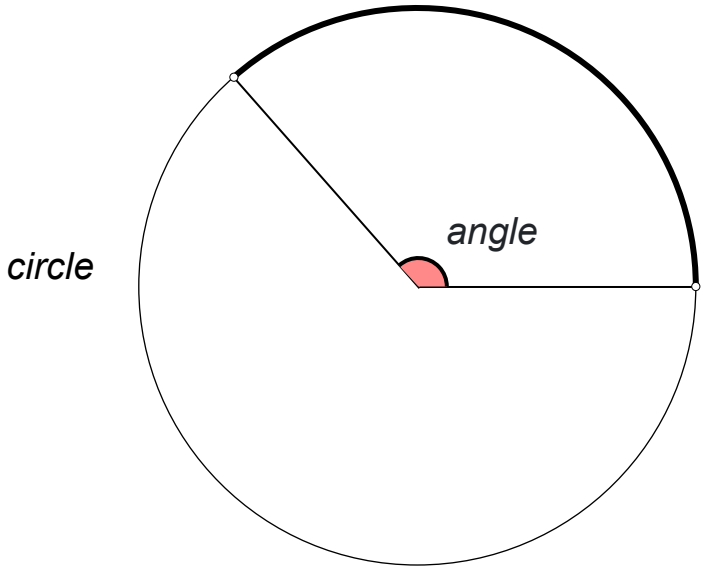


6. Arcs



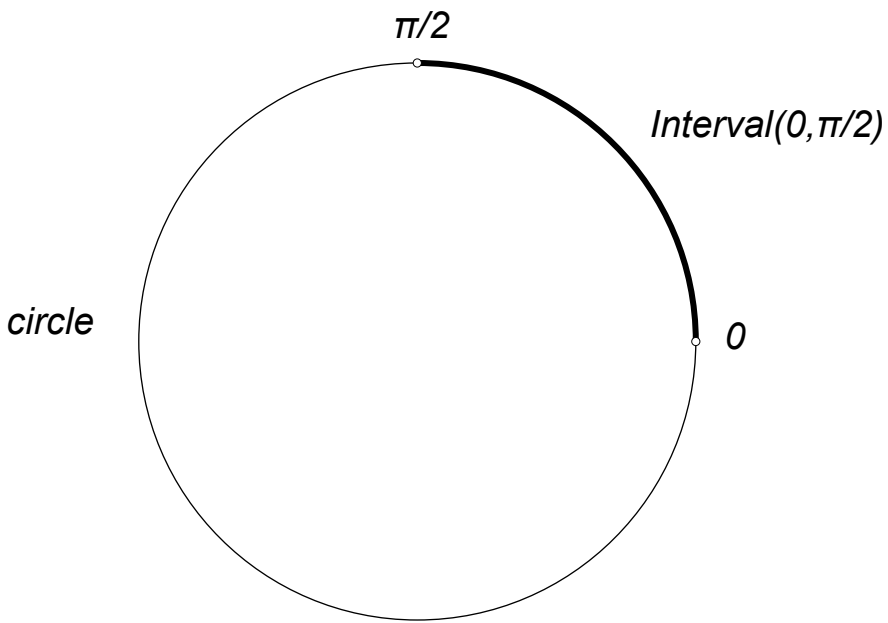
6. Arcs

class **Rhino.Geometry**.**Arc**(Circle *circle*, float *angle*)



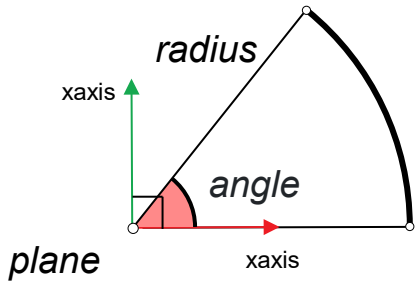
6. Arcs

class **Rhino.Geometry**.**Arc**(Circle *circle*, Interval *angleIntervalRadians*)



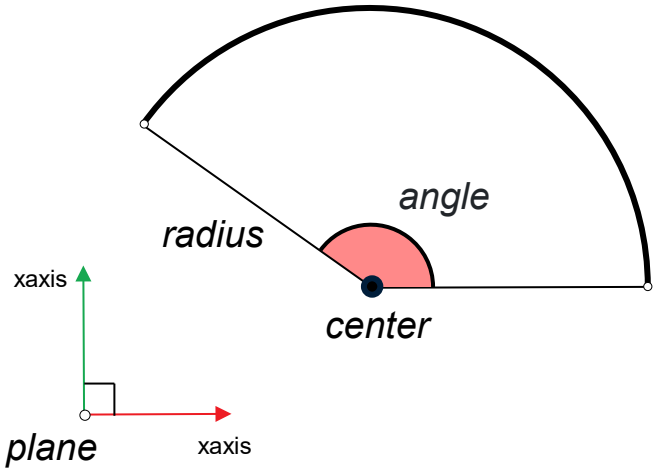
6. Arcs

class **Rhino.Geometry**.**Arc**(Plane *plane*, float *radius*, float *angle*)



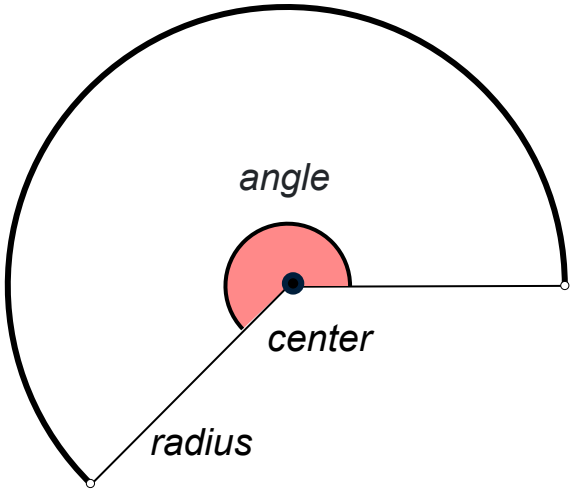
6. Arcs

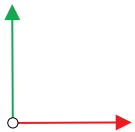
class **Rhino.Geometry**.**Arc**(Plane *plane*, Point3d *center*, float *radius*, float *angle*)



6. Arcs

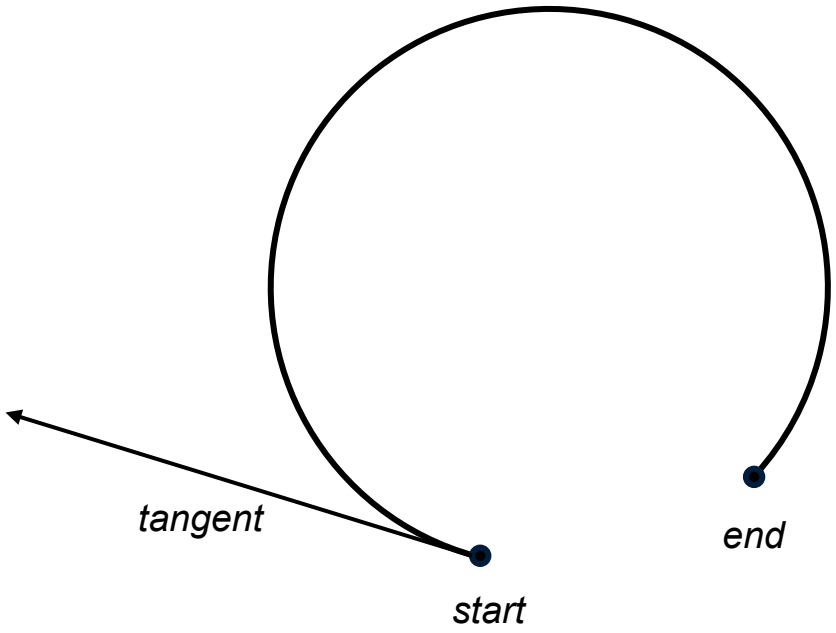
class **Rhino.Geometry**.**Arc**(Point3d *center*, float *radius*, float *angle*)





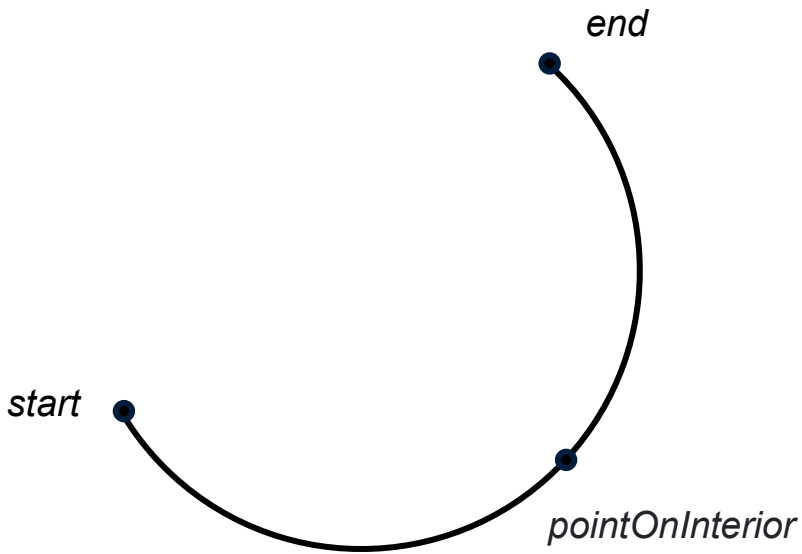
6. Arcs

class **Rhino.Geometry**.**Arc**(Point3d *start*, Vector3d *tangent*, Point3d *end*)



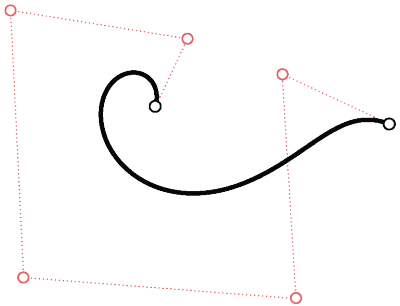
6. Arcs

class **Rhino.Geometry**.**Arc**(Point3d *start*, Point3d *pointOnInterior*, Point3d *end*)

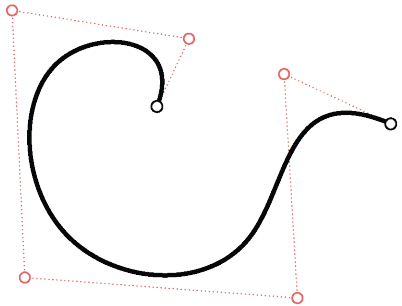


6. Curves

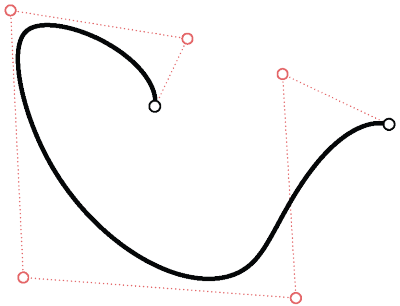
Bézier curve



B-spline curve



NURBS curve

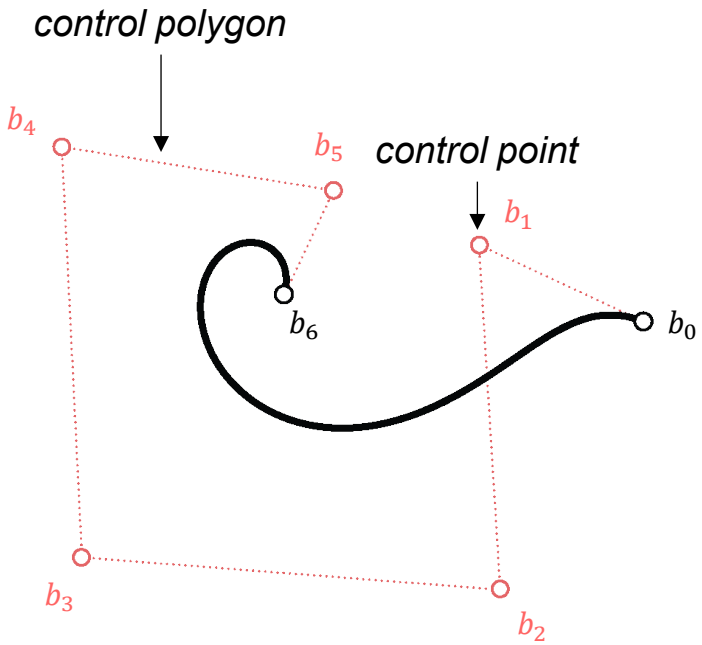


6. Curves

	control points	*degree*	*weights*
Bézier curve	✓		
B-spline	✓	✓	
NURBS	✓	✓	✓

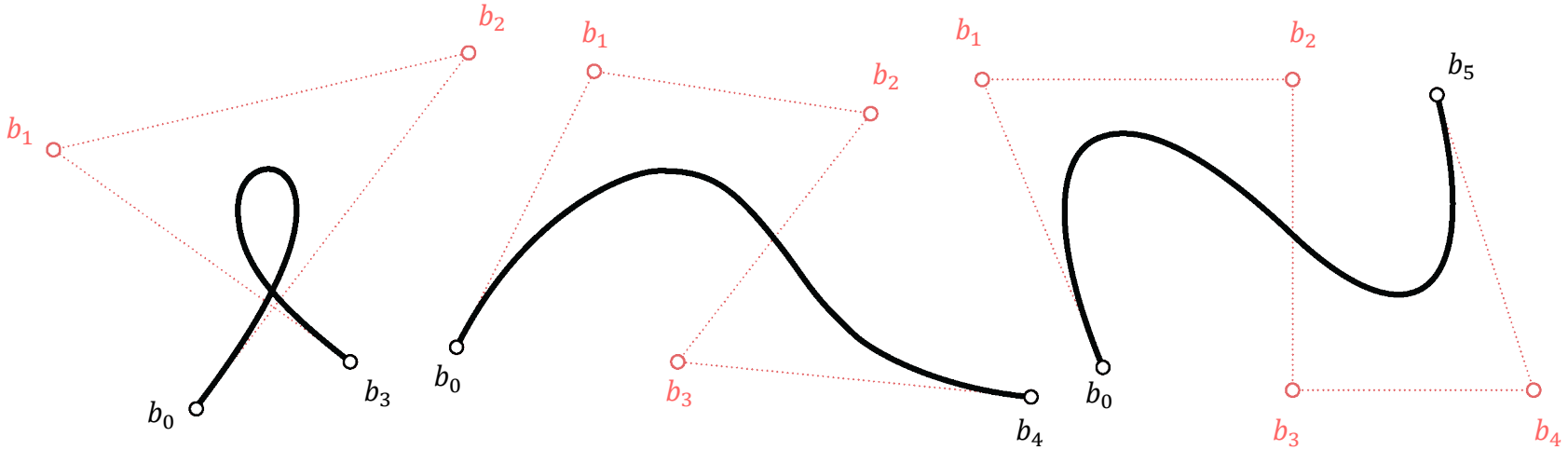
6. Bézier curve

class **Rhino.Geometry**.**BezierCurve**(List <Point3d> *controlPoints*)



A ***Bézier curve*** is completely defined by control polygons.

6. Bézier curve



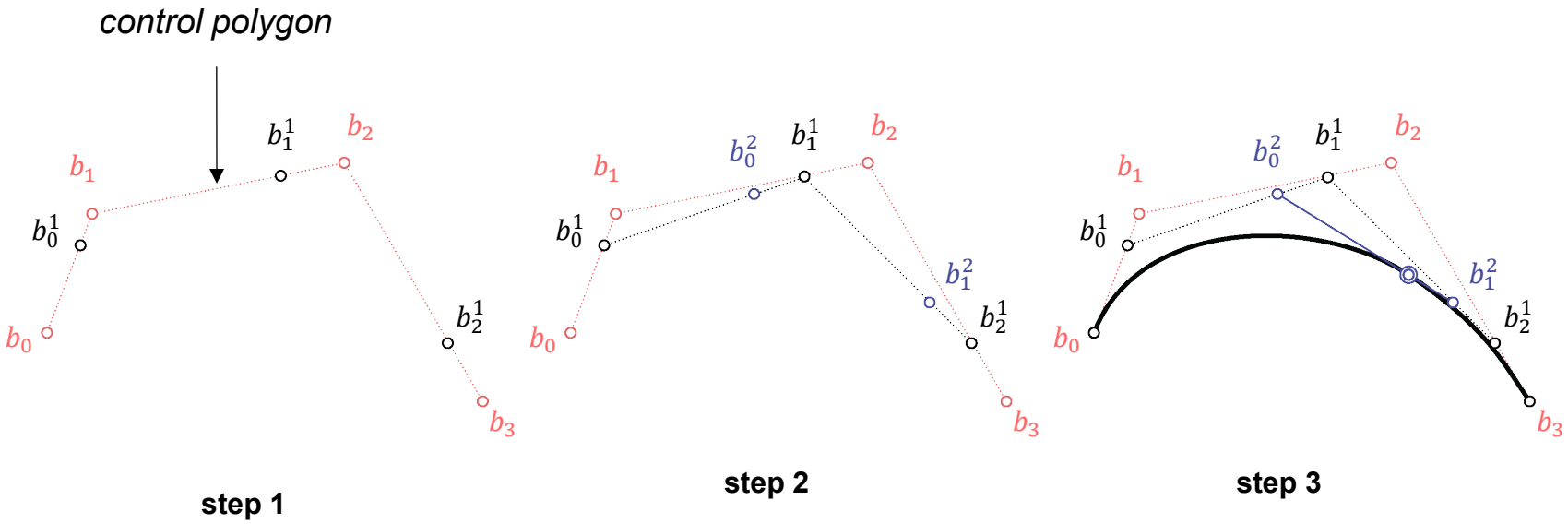
cubic Bézier curve with a loop

Bézier curve of degree 4

Bézier curve of degree 5

A Bézier curve with ***n + 1*** control points is of degree ***n.***

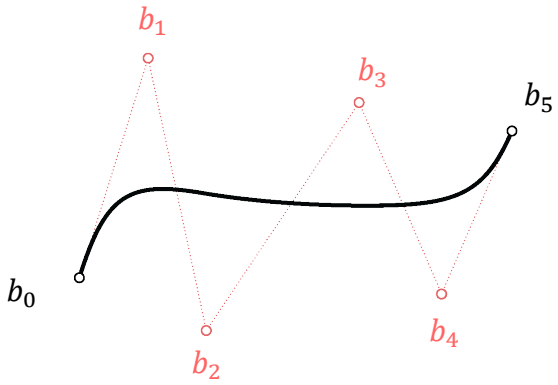
6. Bézier curve

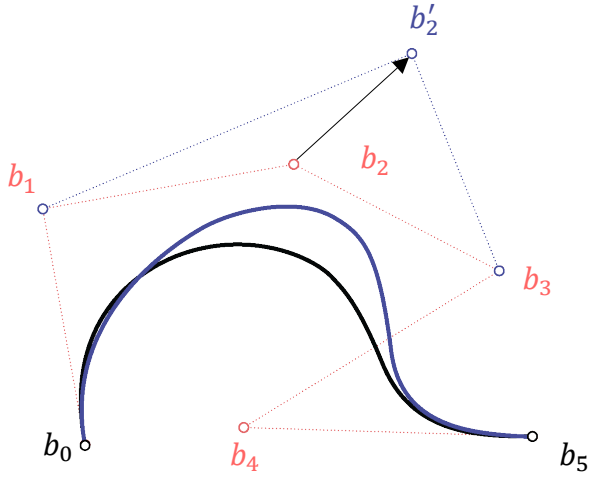


Algorithm of de Casteljau

6. Bézier curve

Limitations:
- **no local control**
- does not pass through most points
- unstable/expensive to calculate



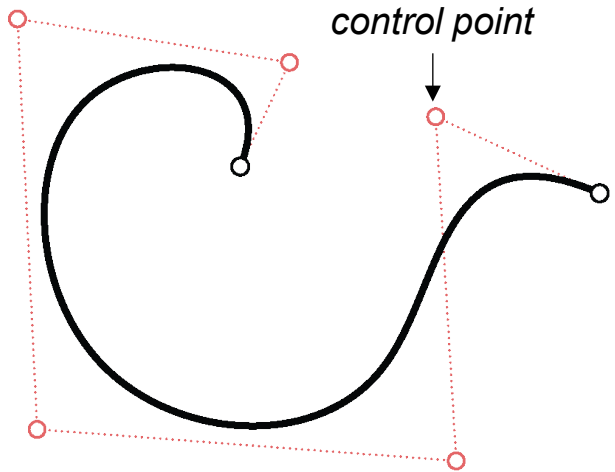


6. B-spline curve

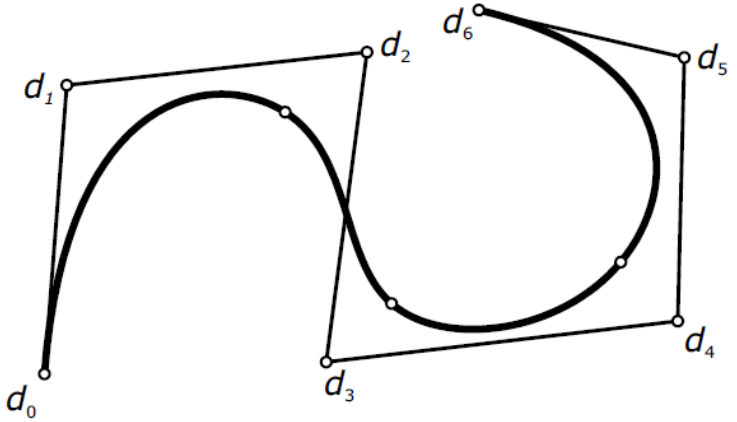
class **Rhino.Geometry**.**NurbsCurve**.Create(Bool *periodic*, int *degree*, list <Point3d> *pts*)

A ***B-spline curve*** is defined by:

- ***m + 1*** control points,
- the curve ***degree n***
- and ***knot vector.***

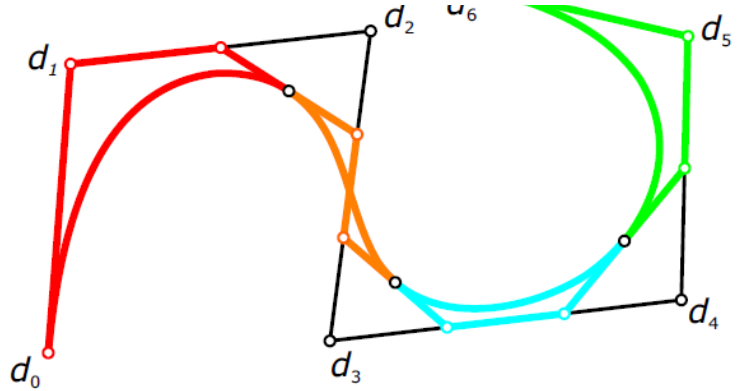


6. B-spline curve

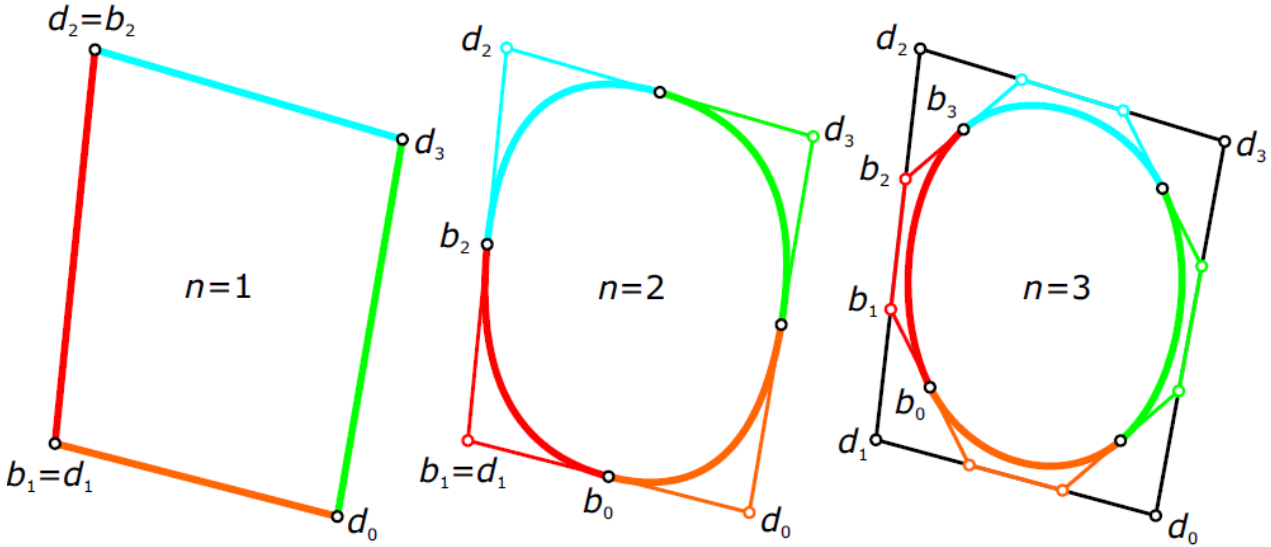


A ***B-spline curve*** is defined by:

- ***m + 1*** control points,
- and the curve ***degree n.***



6. B-spline curve

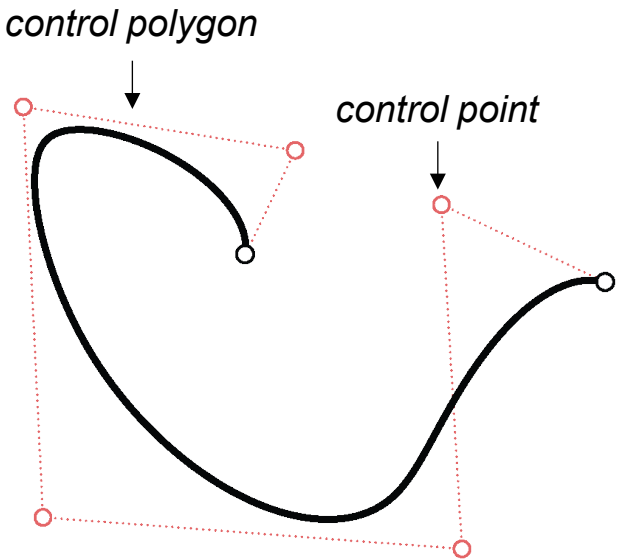


6. NURBS curve

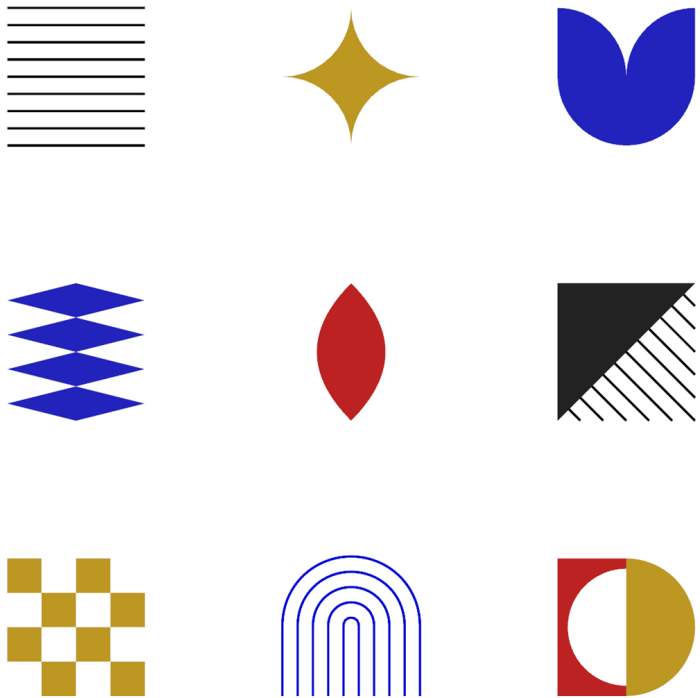
class **Rhino.Geometry**.**NurbsCurve**.Create(Bool *periodic*, int *degree*, list <Point3d> *pts*)

class **Rhino.Geometry**.**NurbsCurve**.Points.SetWeight(int *index*, float *weight*)

A **NURBS curve** is a **B-spline with weights**.



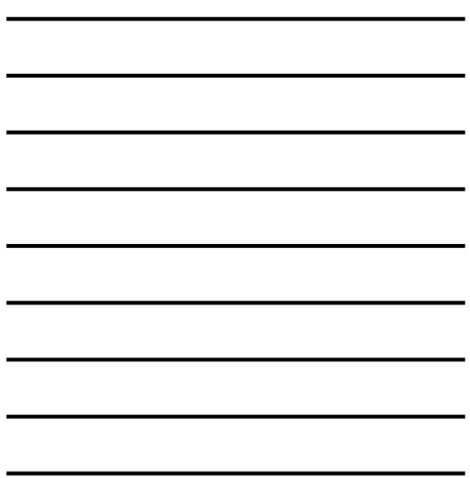
Examples



1. Parallel lines

Learning objectives

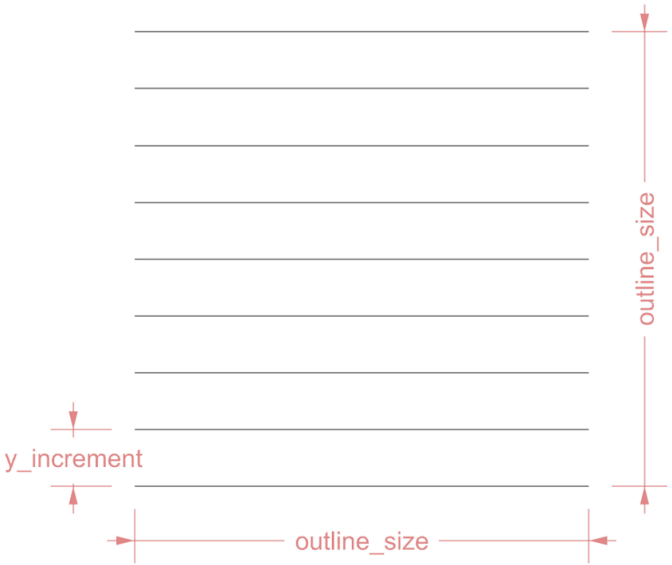
1. Constructing **points**
2. Constructing **lines**
3. Using **for-loops** to generate coordinates for the points



1. Parallel lines

Learning objectives

1. Constructing **points**
2. Constructing **lines**
3. Using **for-loops** to generate coordinates for the points

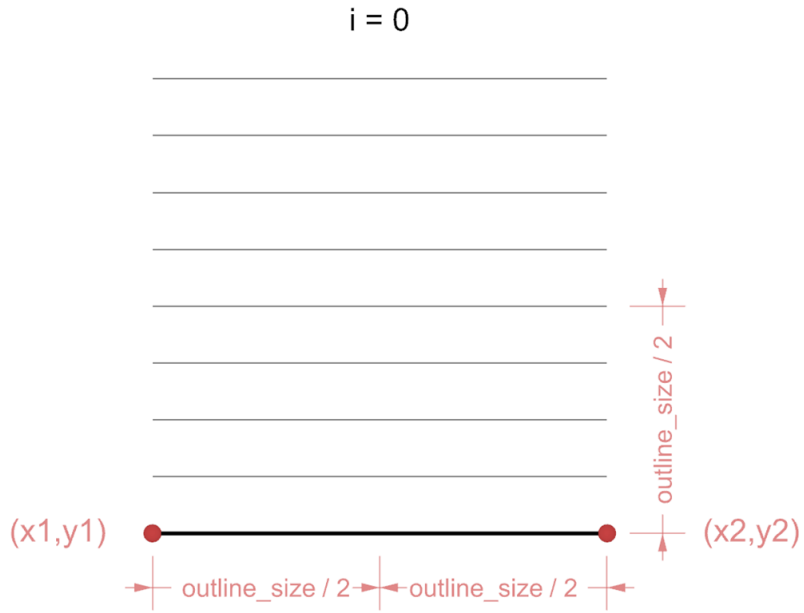


y_increment = outline_size / num

1. Parallel lines

Learning objectives

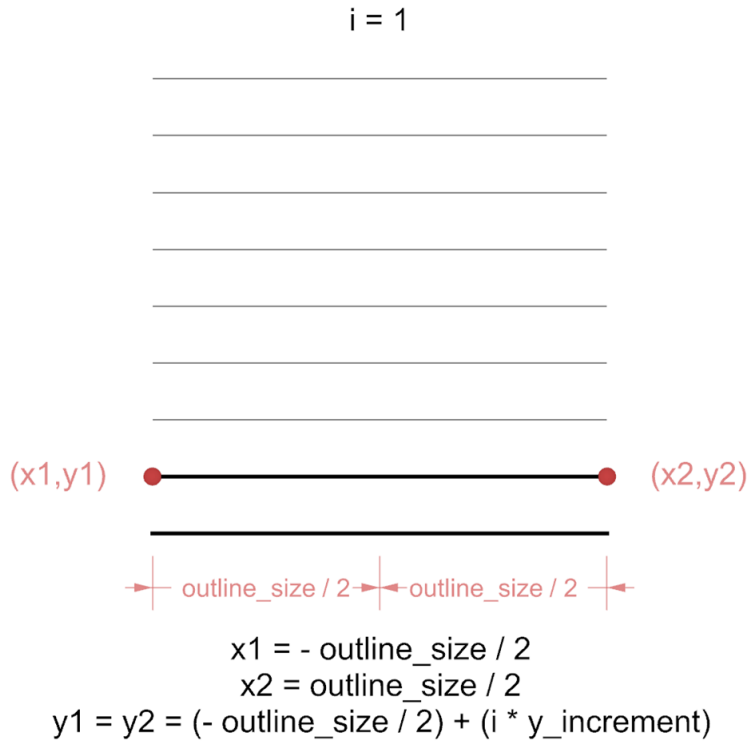
1. Constructing **points**
2. Constructing **lines**
3. Using **for-loops** to generate coordinates for the points



1. Parallel lines

Learning objectives

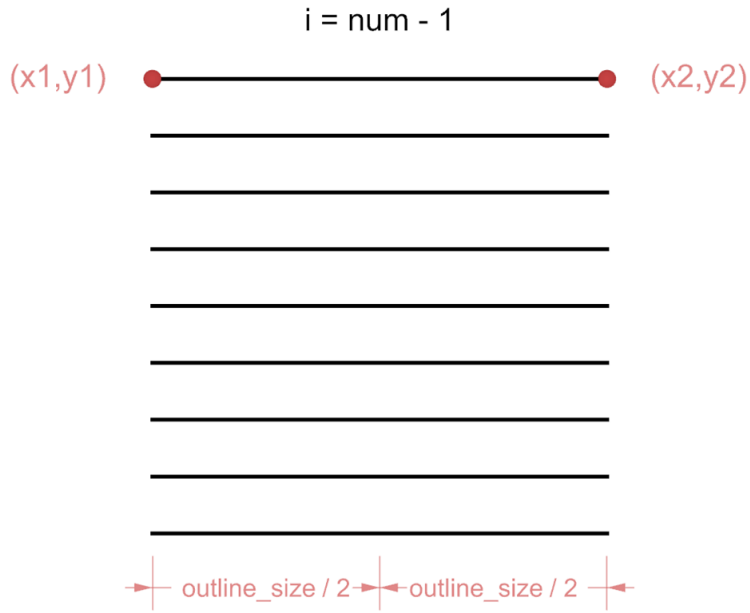
1. Constructing **points**
2. Constructing **lines**
3. Using **for-loops** to generate coordinates for the points



1. Parallel lines

Learning objectives

1. Constructing **points**
2. Constructing **lines**
3. Using **for-loops** to generate coordinates for the points

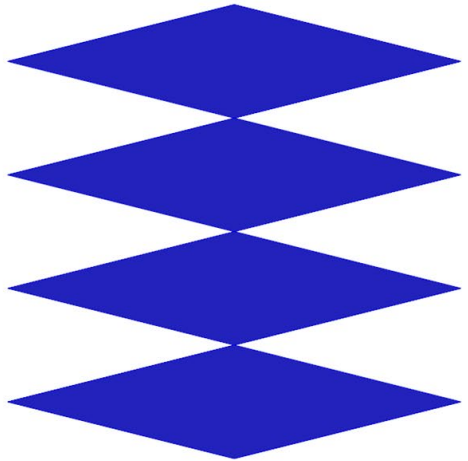


x1 = - outline_size / 2
x2 = outline_size / 2
y1 = y2 = (- outline_size / 2) + (i * y_increment)

2. Stacked rhombuses

Learning objectives

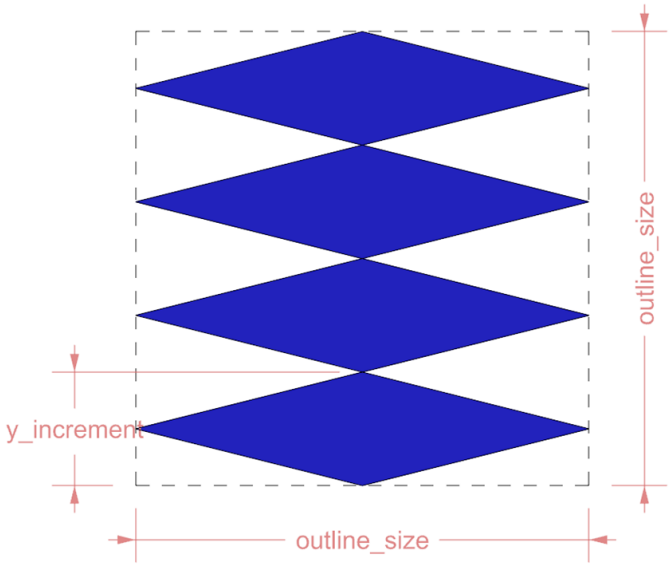
1. Constructing **Polylines**
2. Converting polylines to planar **Breps**



2. Stacked rhombuses

Learning objectives

1. Constructing **Polylines**
2. Converting polylines to planar **Breps**



y_increment = outline_size / num

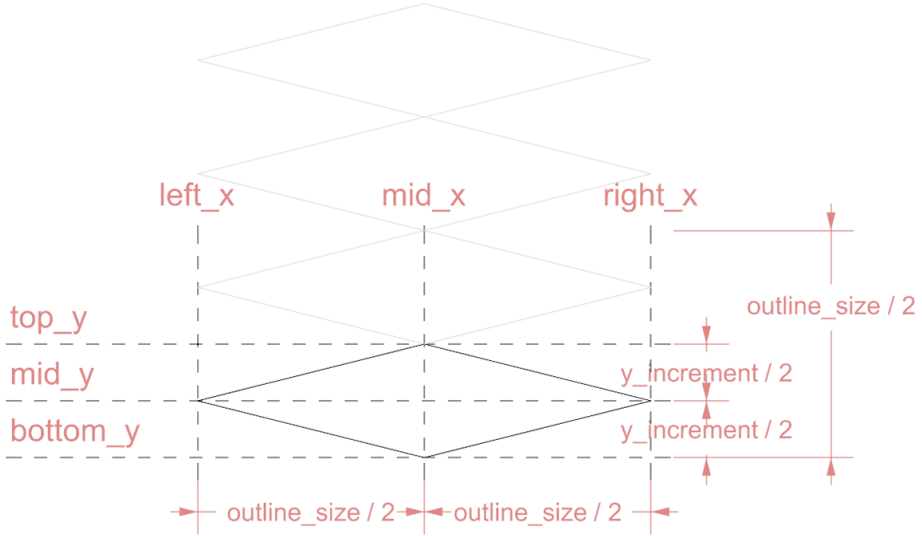
2. Stacked rhombuses

Learning objectives

1. Constructing **Polylines**
2. Converting polylines to planar **Breps**

i = 0

bottom_y = (-outline_size/2) + (i * y_increment)
mid_y = (-outline_size/2) + ((i+1) * (y_increment/2))
top_y = (-outline_size/2) + ((i+1) * y_increment)

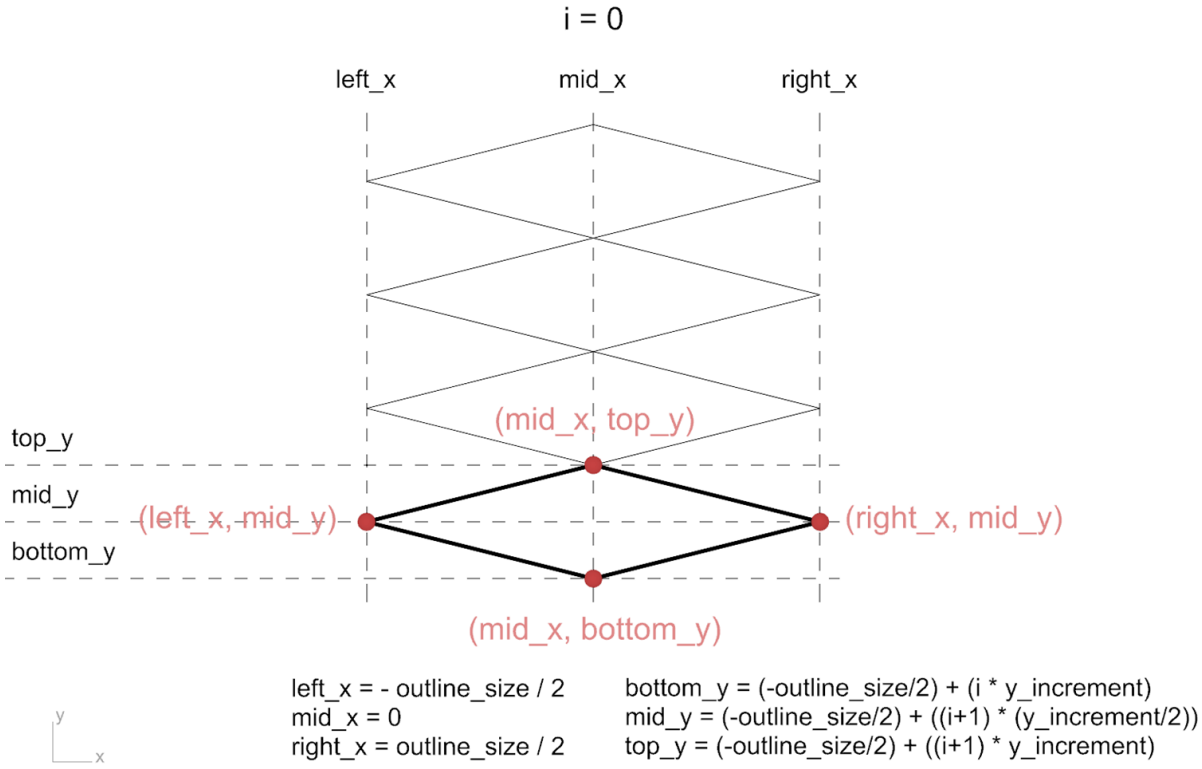


left_x = - outline_size / 2
mid_x = 0
right_x = outline_size / 2

2. Stacked rhombuses

Learning objectives

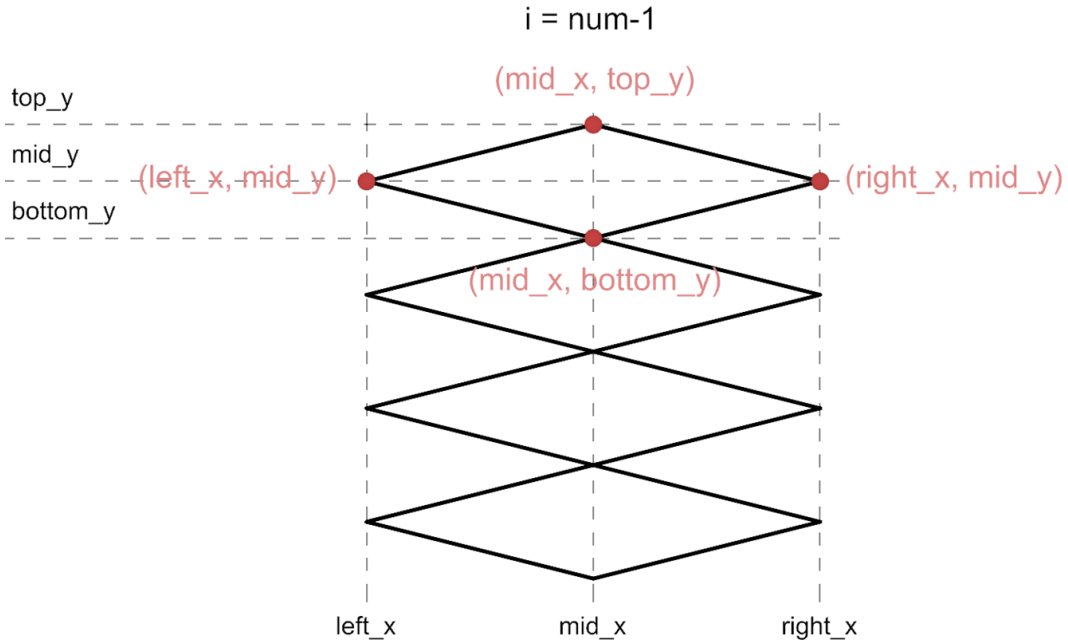
1. Constructing **Polylines**
2. Converting polylines to planar **Breps**



2. Stacked rhombuses

Learning objectives

1. Constructing **Polylines**
2. Converting polylines to planar **Breps**



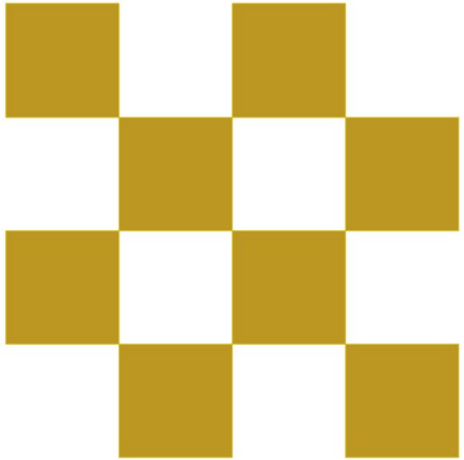
left_x = - outline_size / 2
mid_x = 0
right_x = outline_size / 2

bottom_y = (-outline_size/2) + (i * y_increment)
mid_y = (-outline_size/2) + ((i+1) * (y_increment/2))
top_y = (-outline_size/2) + ((i+1) * y_increment)

3. Chequered squares

Learning objectives

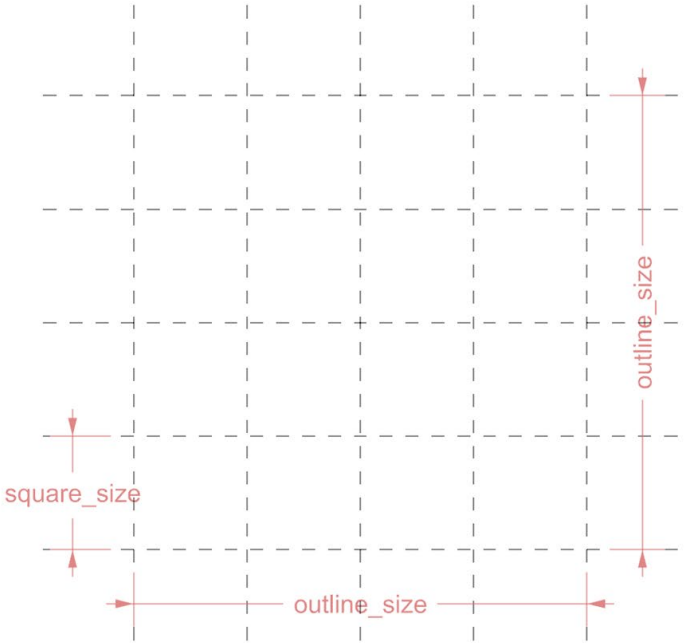
1. Using **nested for-loops** and **if-condition**
2. Constructing **Rectangle**



3. Chequered squares

Learning objectives

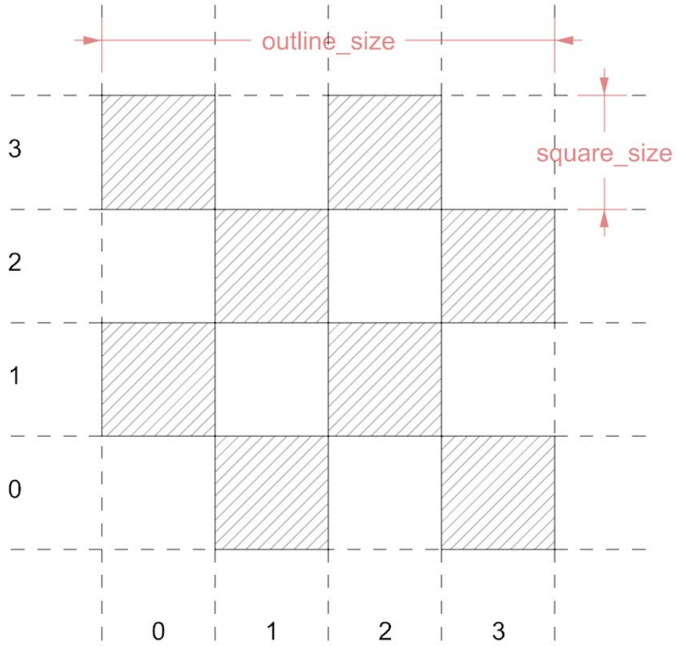
1. Using **nested for-loops** and **if-condition**
2. Constructing **Rectangle**



3. Chequered squares

Learning objectives

1. Using **nested for-loops** and **if-condition**
2. Constructing **Rectangle**



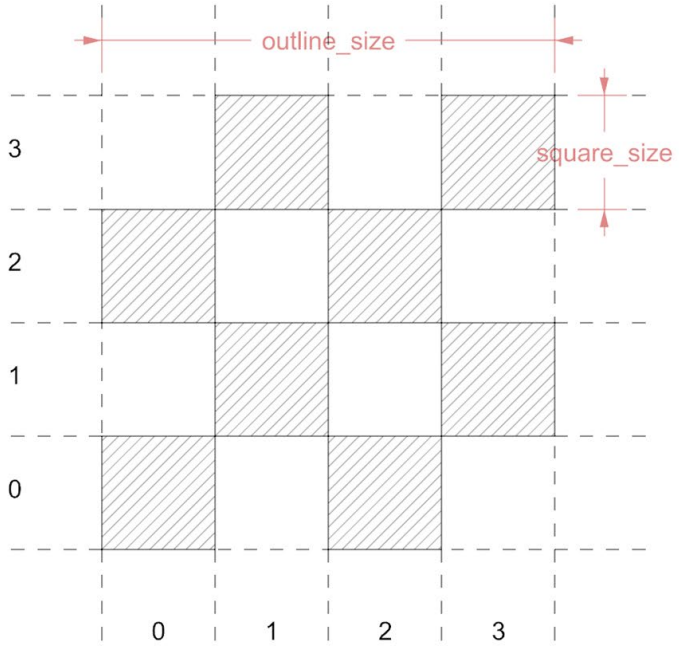
if (i%2==0 and j%2!=0) or (i%2!=0 and j%2==0)

Option 1: Either i or j should be divisible by 2 but not both

3. Chequered squares

Learning objectives

1. Using **nested for-loops** and **if-condition**
2. Constructing **Rectangle**



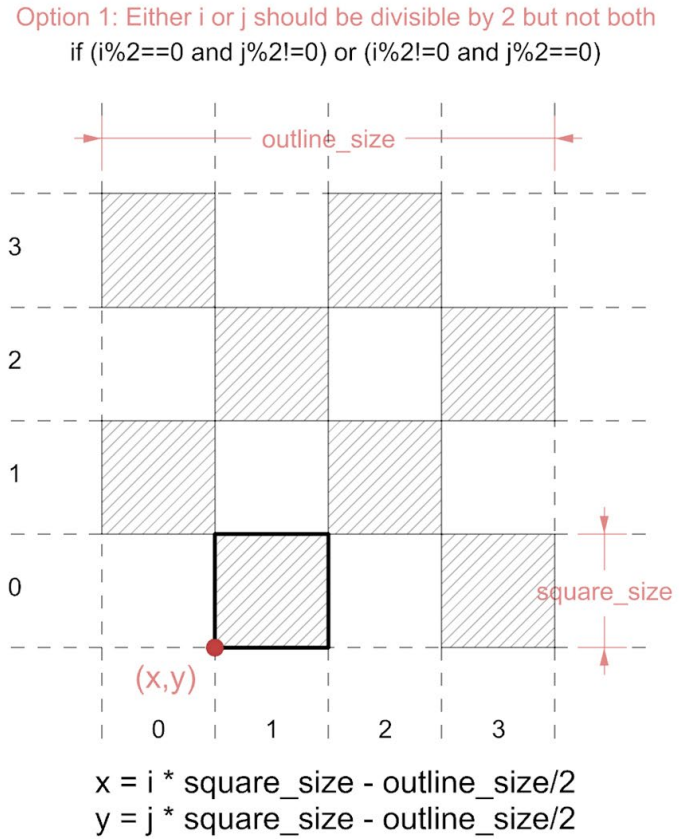
if (i%2==0 and j%2==0) or (i%2!=0 and j%2!=0)

Option 2: Both i and j should be divisible by 2 or not

3. Chequered squares

Learning objectives

1. Using **nested for-loops** and **if-condition**
2. Constructing **Rectangle**



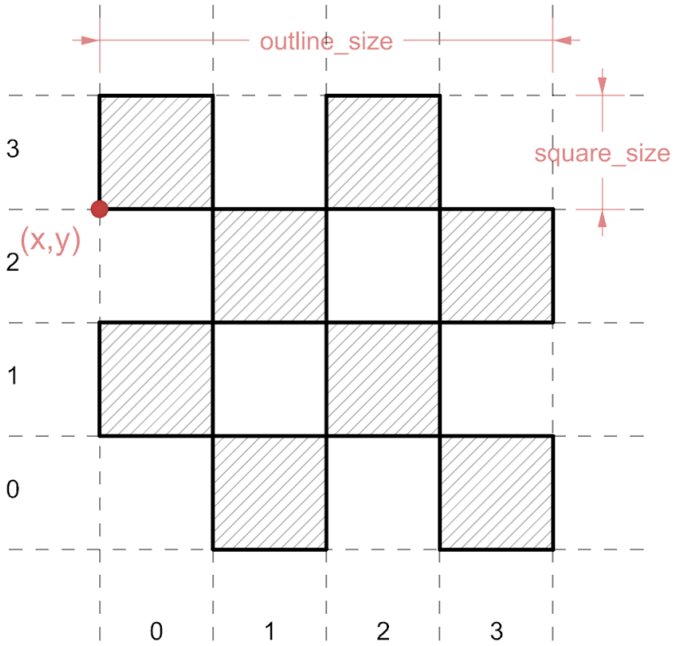
3. Chequered squares

Learning objectives

1. Using **nested for-loops** and **if-condition**
2. Constructing **Rectangle**

Option 1: Either i or j should be divisible by 2 but not both

if (i%2==0 and j%2!=0) or (i%2!=0 and j%2==0)



x = i * square_size - outline_size/2

y = j * square_size - outline_size/2

4. Star

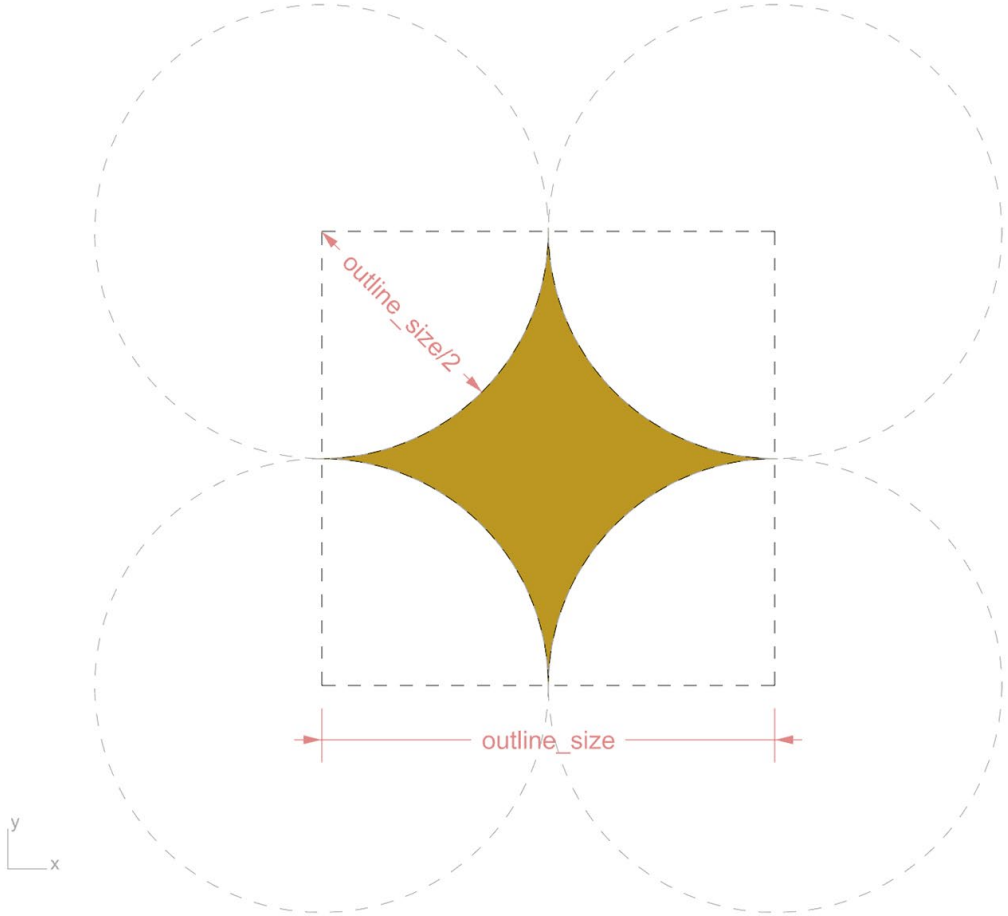
Learning objectives

1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**

4. Star

Learning objectives

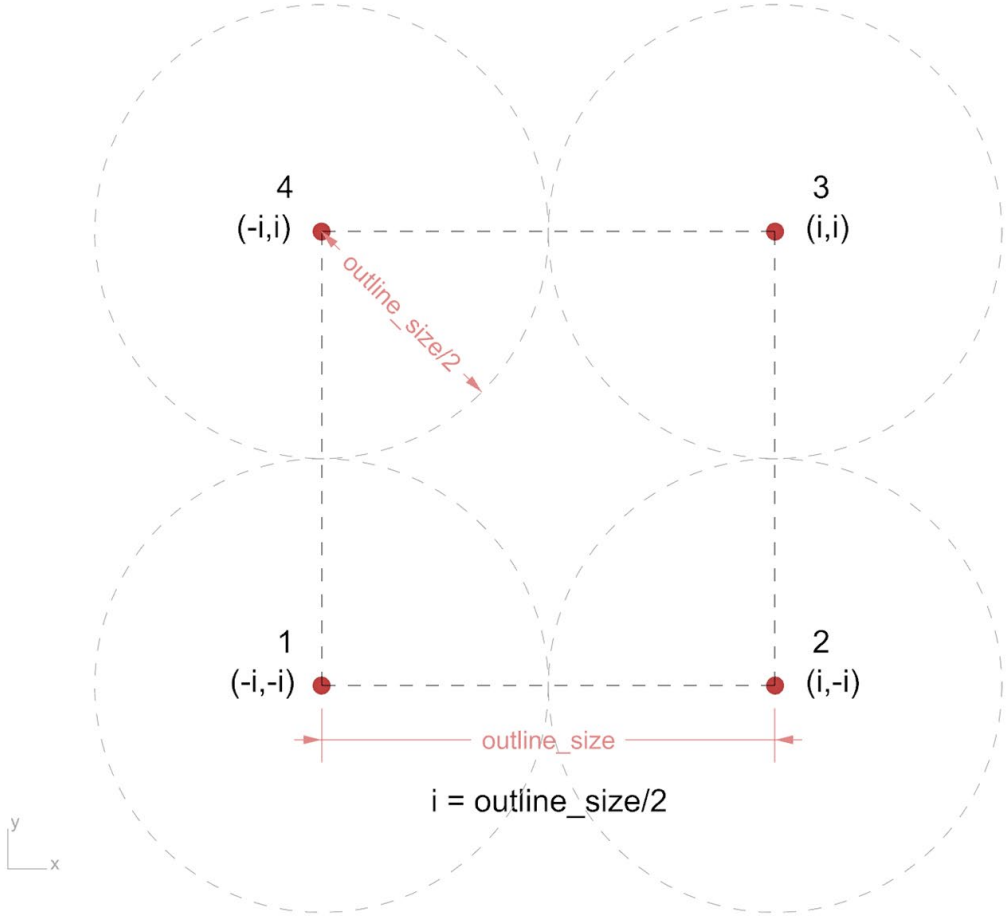
1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**



4. Star

Learning objectives

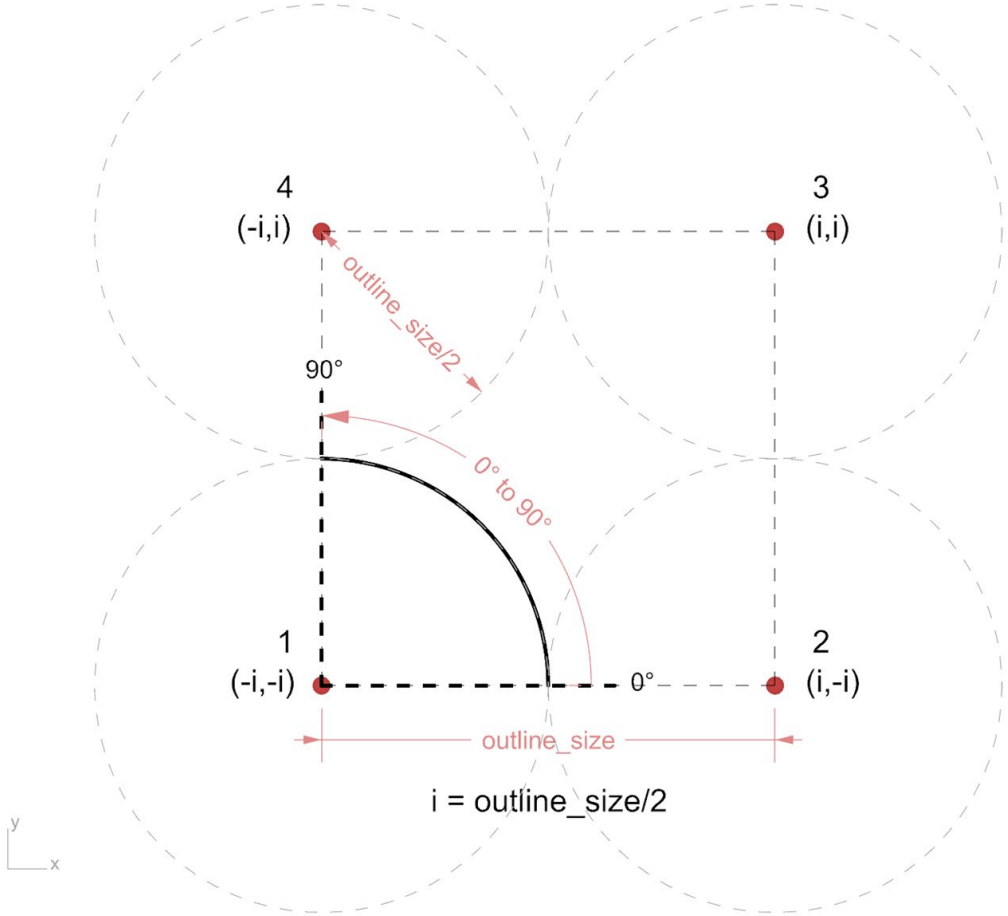
1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**



4. Star

Learning objectives

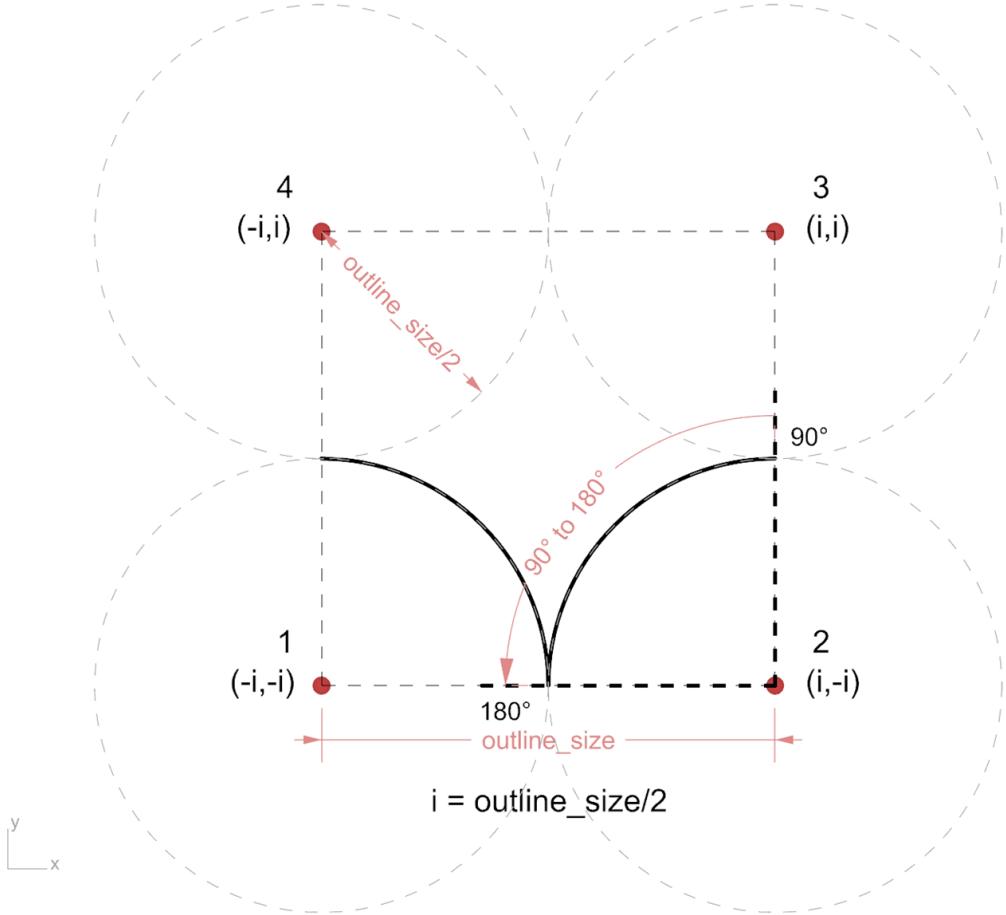
1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**



4. Star

Learning objectives

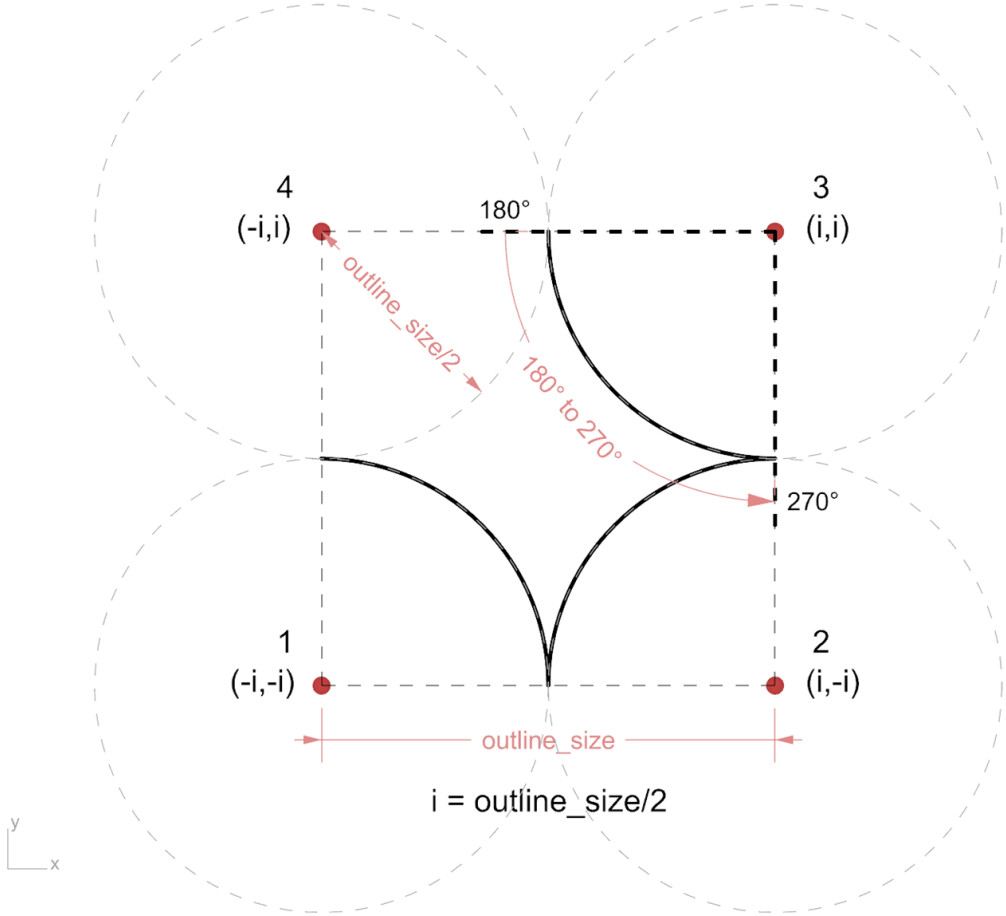
1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**



4. Star

Learning objectives

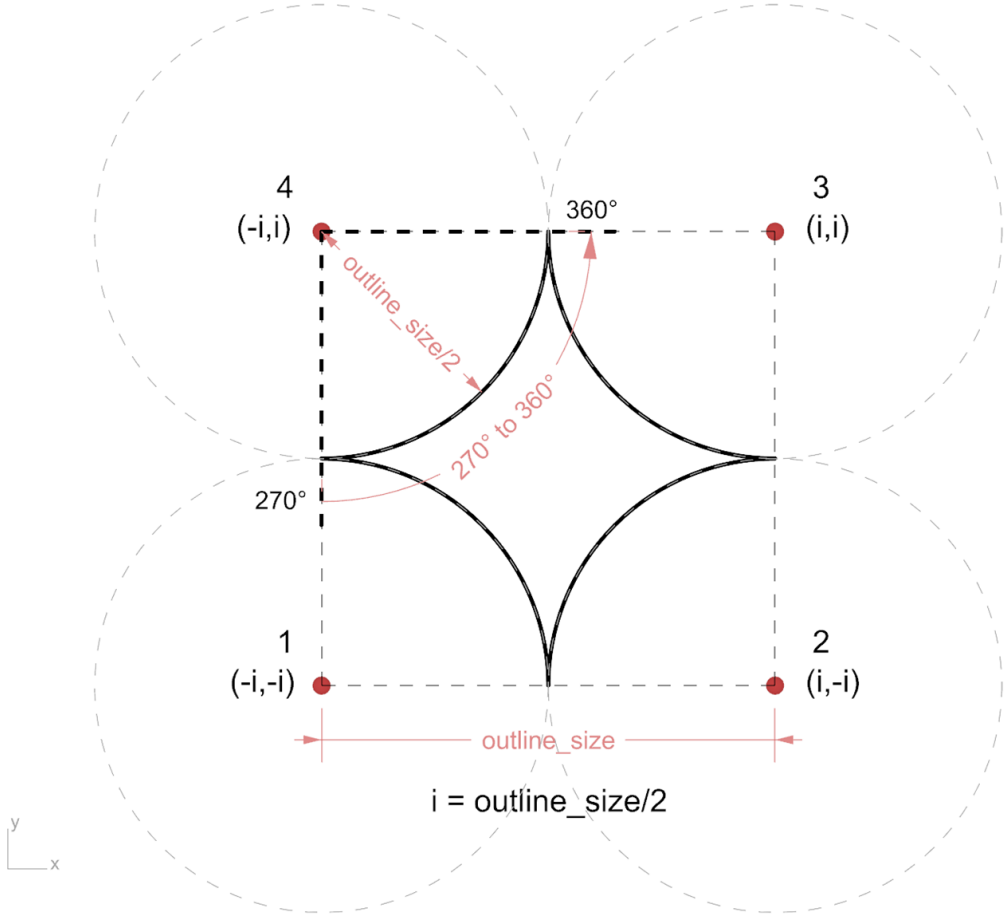
1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**



4. Star

Learning objectives

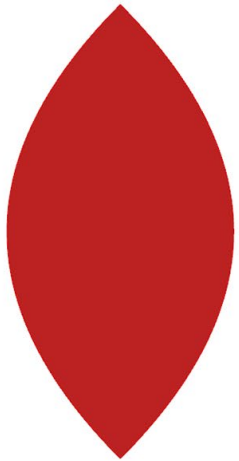
1. Constructing **Circles**
2. Constructing **Arcs** from circles and **angle domains**
3. Creating planar **Breps with multiple curves**



5. Eye shape

Learning objectives

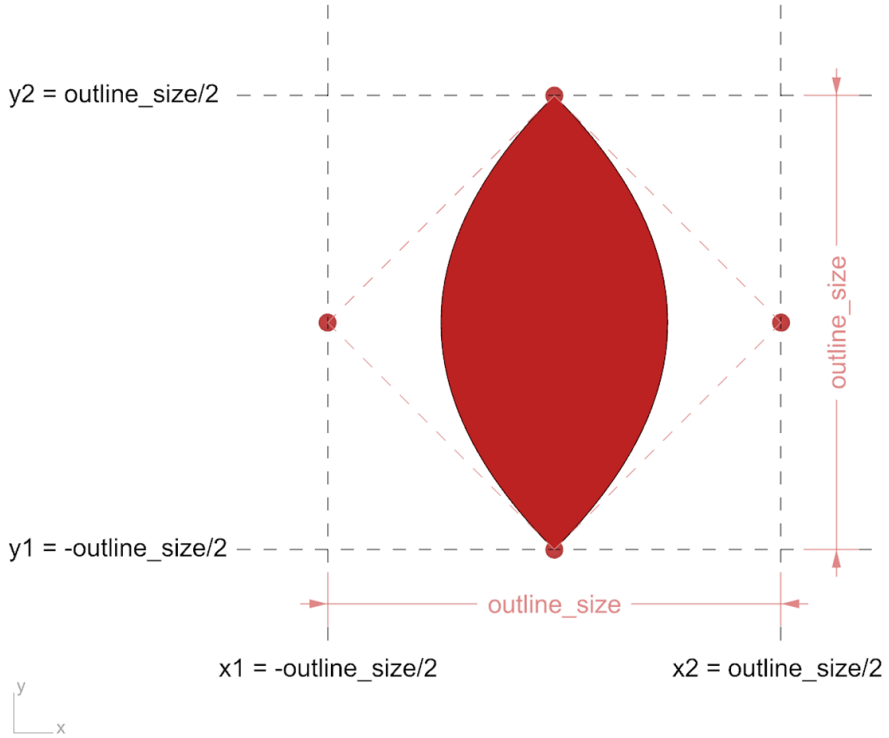
1. Constructing **Curves** with **Control points**
2. Constructing Breps with multiple planar curves.



5. Eye shape

Learning objectives

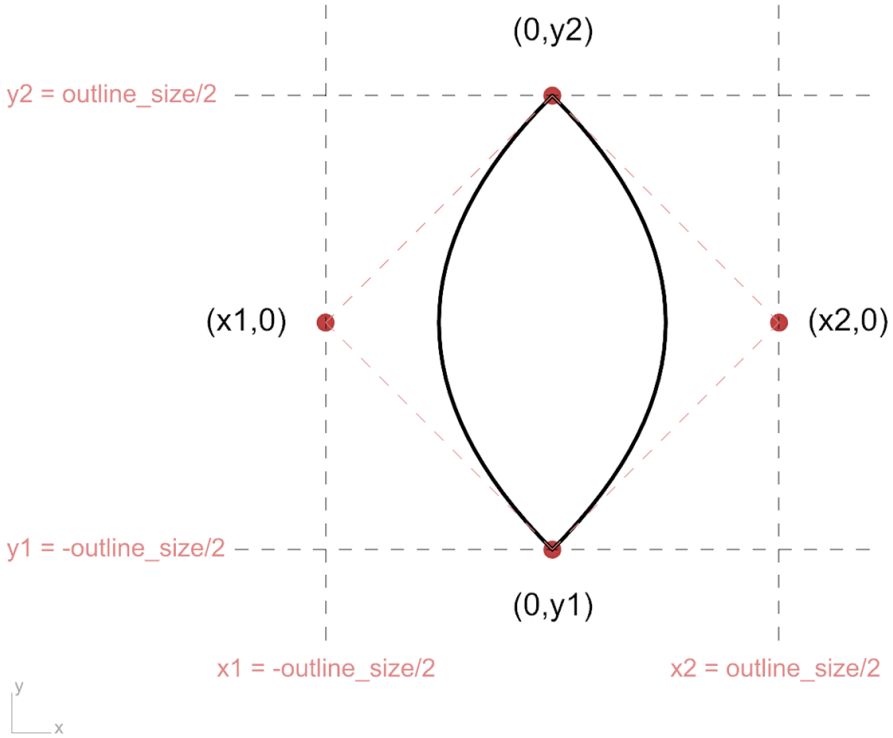
1. Constructing **Curves** with **Control points**
2. Constructing Breps with multiple planar curves.



5. Eye shape

Learning objectives

1. Constructing **Curves** with **Control points**
2. Constructing Breps with multiple planar curves.

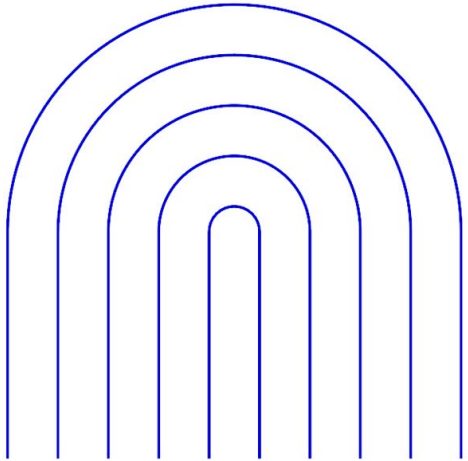


6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves

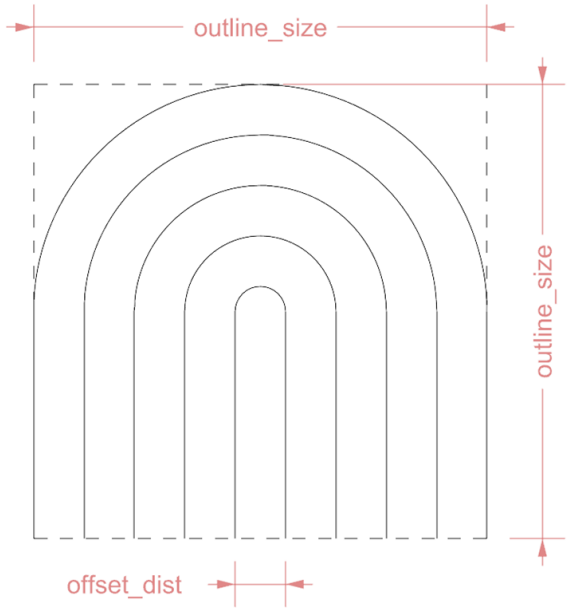


6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves



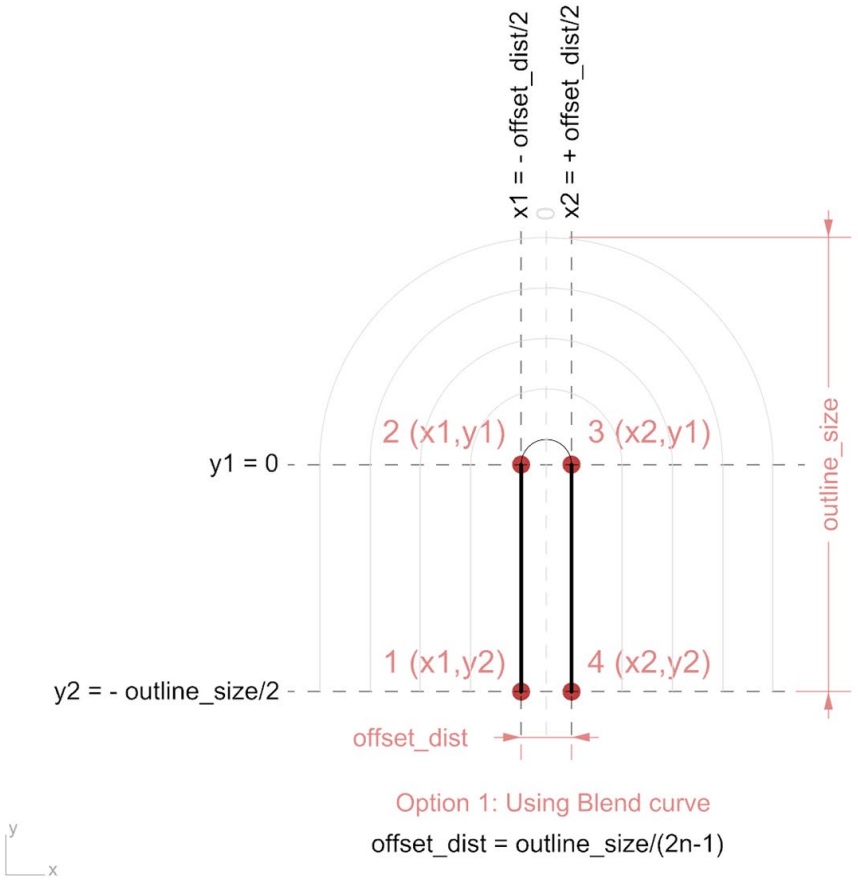
offset_dist = outline_size/(2n-1)

6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves

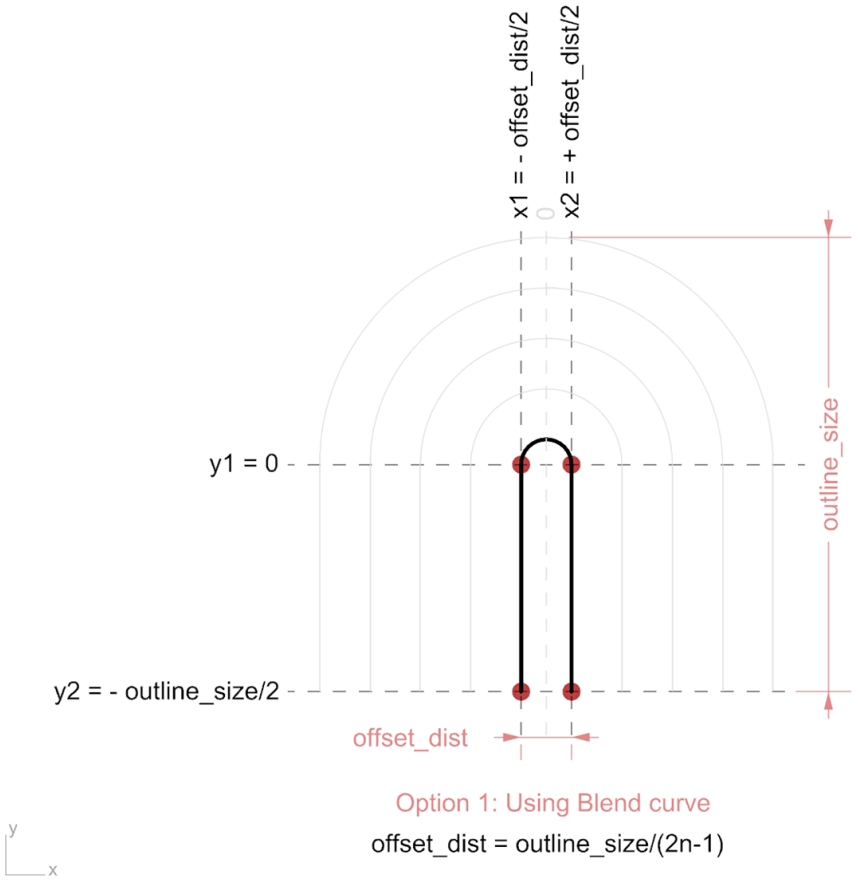


6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves

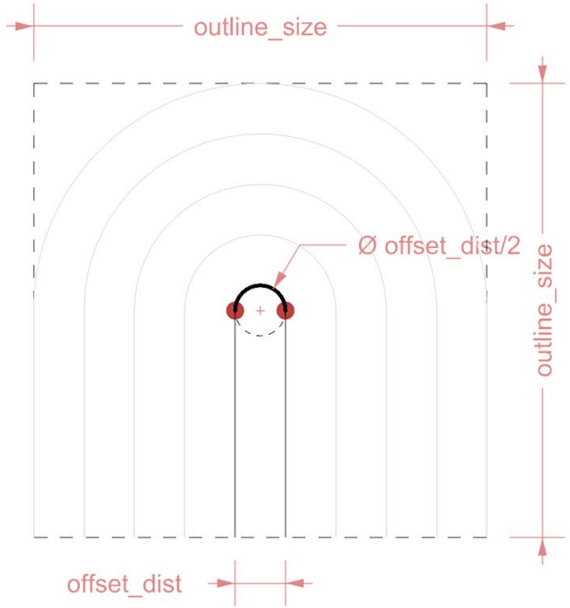


6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves



Option 2: Using Extend curve

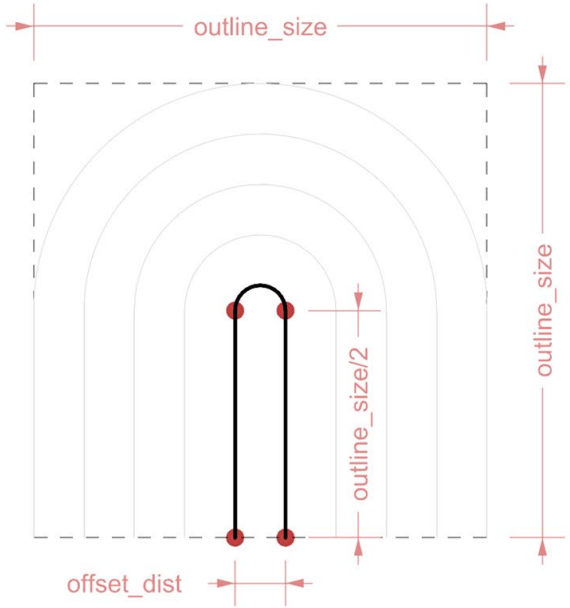
offset_dist = outline_size/(2n-1)

6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves



Option 2: Using Extend curve

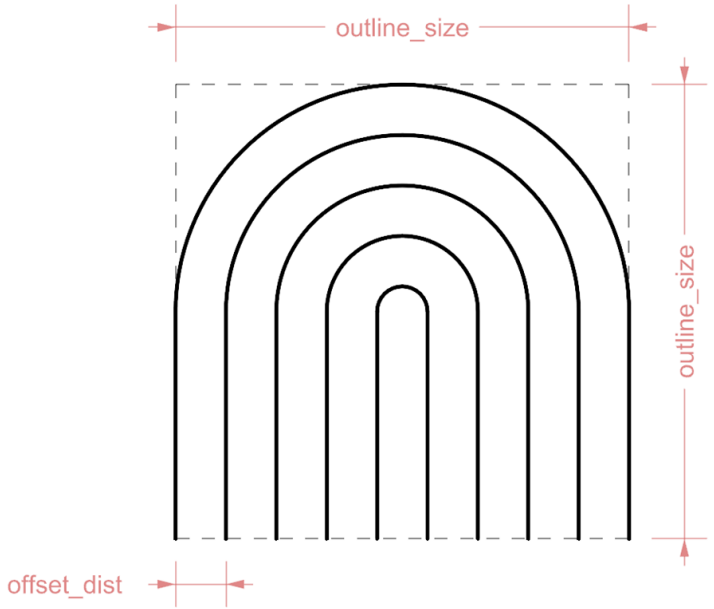
offset_dist = outline_size/(2n-1)

6. Concentric arches

Learning objectives

Understanding Curve operations like:

1. **Blend** curves
2. **Join** curves
3. **Offset** curves
4. **Extend** curves

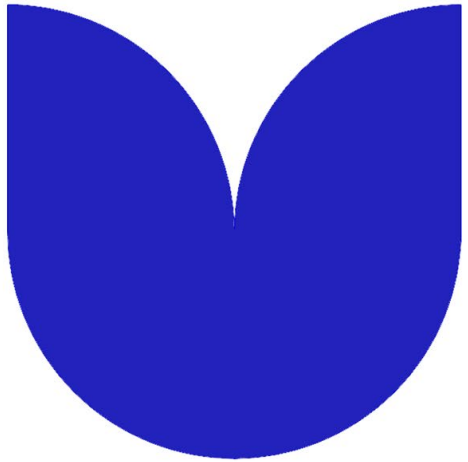


7. Tulip shape

Learning objectives

Understanding Curve operations like:

1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**

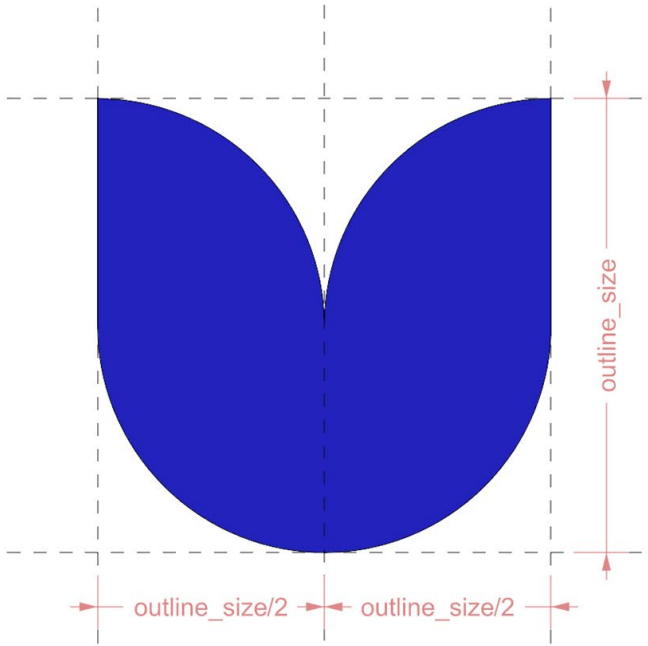


7. Tulip shape

Learning objectives

Understanding Curve operations like:

1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**

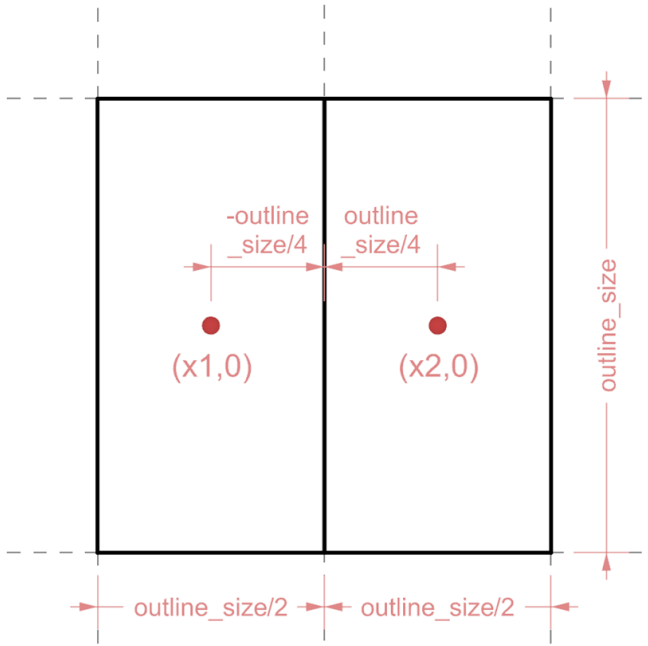


7. Tulip shape

Learning objectives

Understanding Curve operations like:

1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**



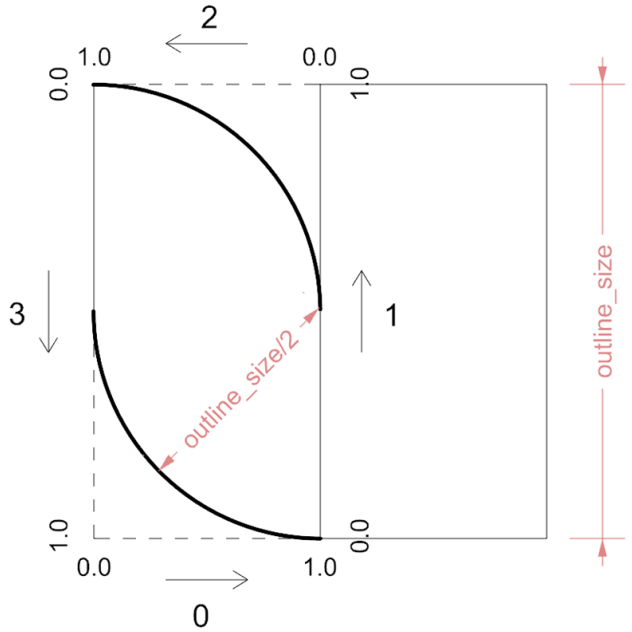
x_domain = -outline_size/4 to outline_size/4
y_domain = -outline_size/2 to outline_size/2

7. Tulip shape

Learning objectives

Understanding Curve operations like:

1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**

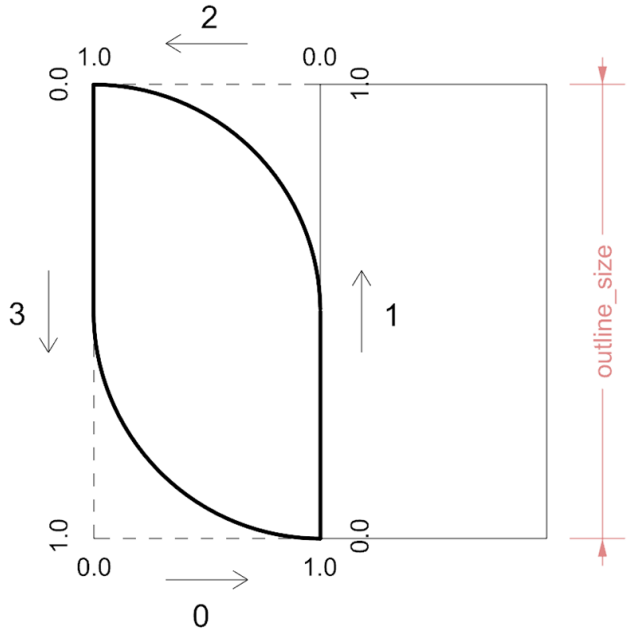


7. Tulip shape

Learning objectives

Understanding Curve operations like:

1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**

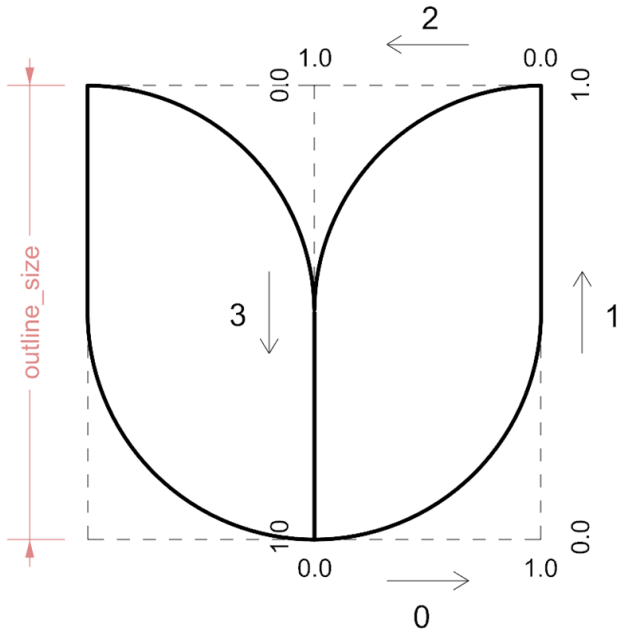


7. Tulip shape

Learning objectives

Understanding Curve operations like:

1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**

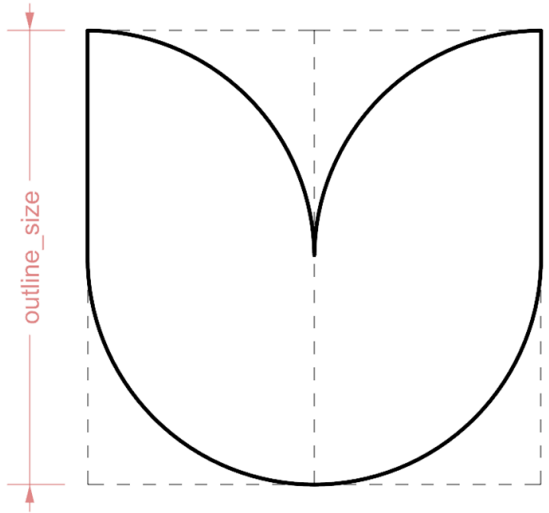


7. Tulip shape

Learning objectives

Understanding Curve operations like:

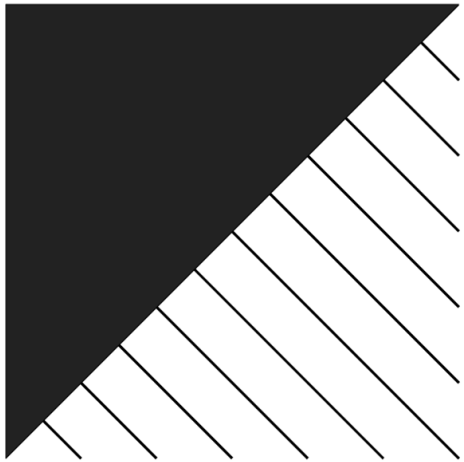
1. **Explode** curves
2. **Fillet** curves
3. **Boolean Union**



8. Solid and hatches triangles

Learning objectives

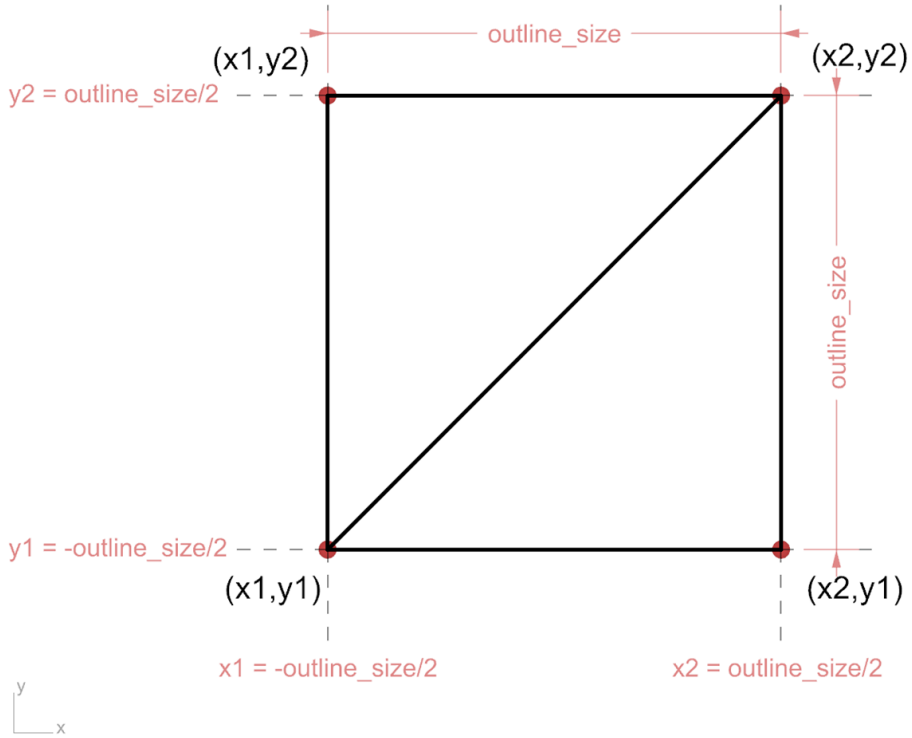
1. Constructing **lines** with start point, direction vector and distance.
2. Dividing a curve with count.
3. Understanding **Curve Intersections**



8. Solid and hatches triangles

Learning objectives

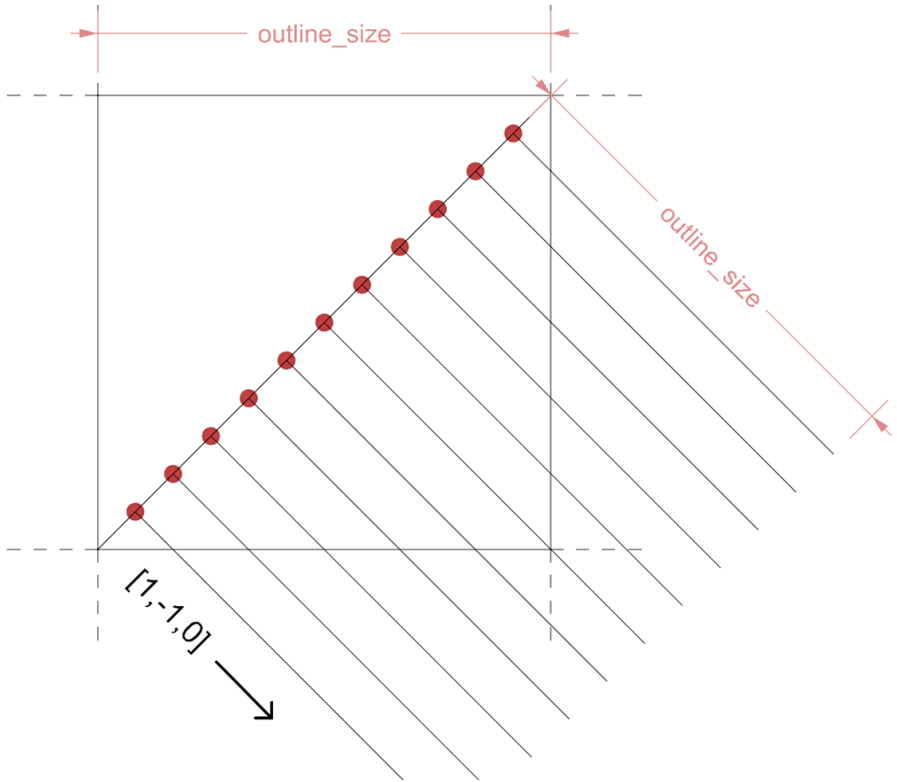
1. Constructing **lines** with start point, direction vector and distance
2. Dividing a curve with count.
3. Understanding **Curve Intersections**



8. Solid and hatches triangles

Learning objectives

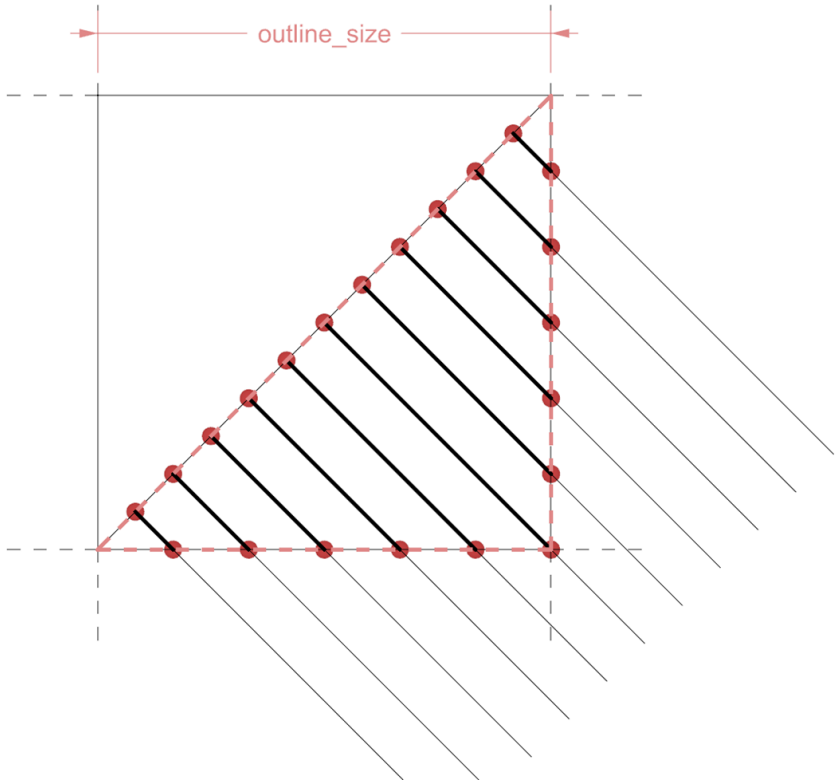
1. Constructing **lines** with start point, direction vector and distance
2. Dividing a curve with count.
3. Understanding **Curve Intersections**



8. Solid and hatches triangles

Learning objectives

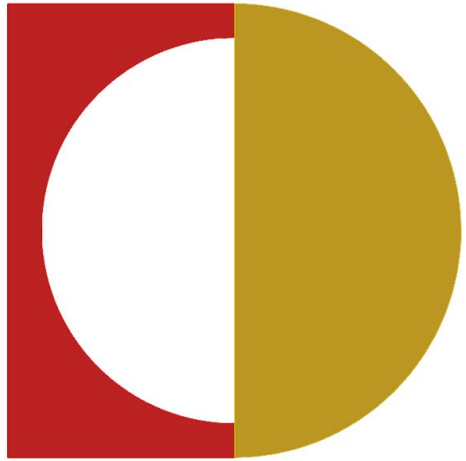
1. Constructing **lines** with start point, direction vector and distance.
2. Dividing a curve with count.
3. Understanding **Curve Intersections**



9. Subtracted shapes

Learning objectives

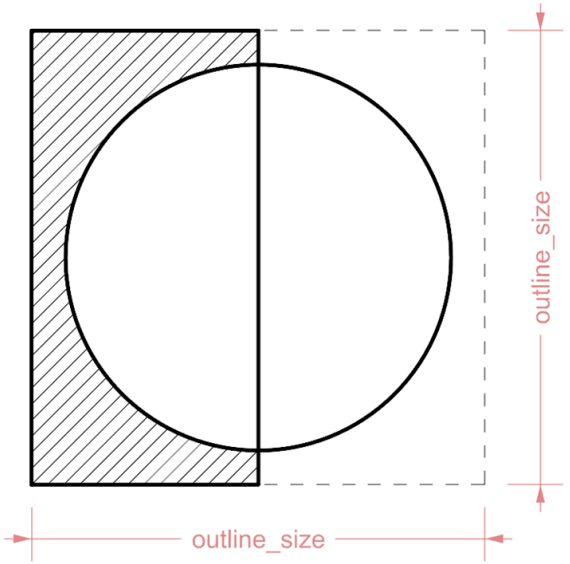
Understanding **Boolean operations**



9. Subtracted shapes

Learning objectives

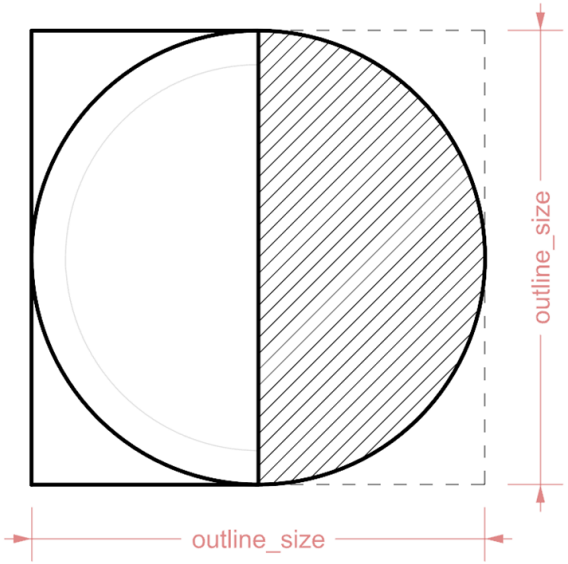
Understanding **Boolean operations**



9. Subtracted shapes

Learning objectives

Understanding **Boolean operations**



Open your Grasshopper/Rhino environment and try out our example geometries!

Look into RhinoCommon API for further geometries and functions.

Enjoy creating!