arch_calculator_v4

March 23, 2022

1 Arch Calculator:

1.1 A Python program for calculating coordinates of arches and angled walls

By Kenneth Burchfiel

This program demonstrates how Python and trigonometry can be used together to generate 2D coordinates of arches and angled walls. This makes the process of creating such arches and walls within a 3D design program like Blender much easier.

I'll first import a number of programs:

```
[ ]: import math
dtr = math.pi/180 # Converts degrees to radians;
# useful because math.tan, math.sin, and math.cos take radian inputs.
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon # From:
# https://matplotlib.org/stable/gallery/shapes_and_collections/patch_collection.
↪html#sphx-glr-gallery-shapes-and-collections-patch-collection-py
from IPython.display import Image # from: Magno Naoli at:
# https://stackoverflow.com/a/62985628/13097194
```

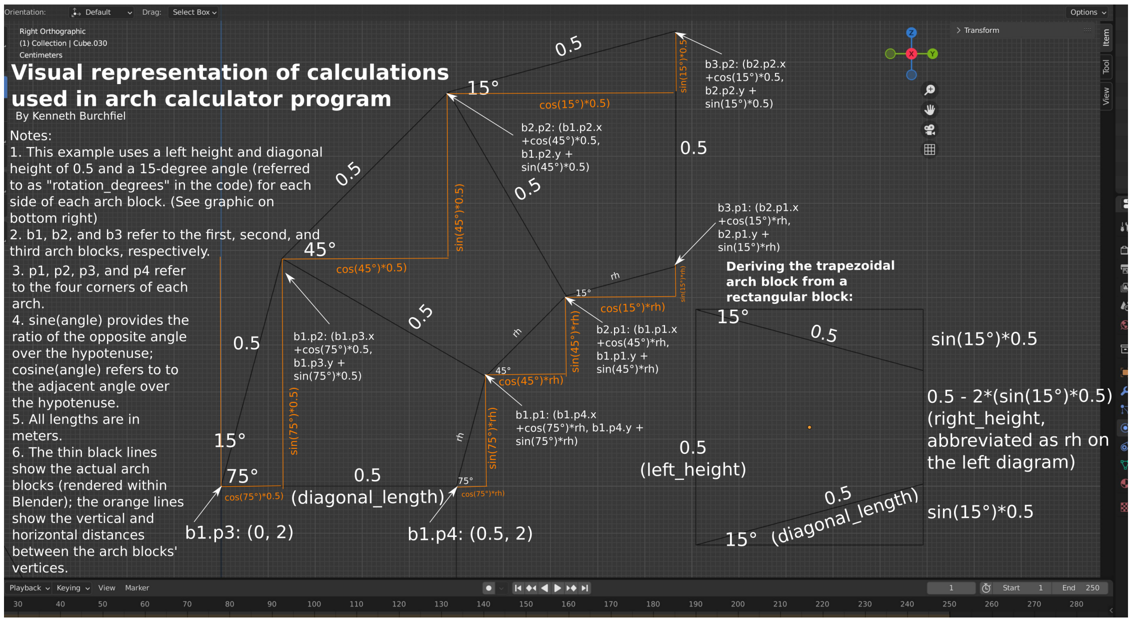
The main purpose of this program is to calculate the points in (x, y) space that serve as coordinates for components of an arch. Calculating these points involves trigonometry, as the positions of the points are influenced by the sizes of the angles that help determine each block's shape. The image below provides a visual description of these calculations.

The program first calculates the dimensions of a single arch block (see the bottom right of the image). Once that block is calculated, it places the block on its side based on the starting coordinates provided. These starting coordinates comprise p3 and p4, the two points on the bottom of this block.

Next, it uses trigonometry to calculate the coordinates of p1 and p2, the block's top two points. These points then serve as p3 and p4 of the next block. The same trigonometric calculations are used to calculate p1 and p2 of this new block; the only difference is that the angles of this block relative to the coordinate space have changed. This process is then repeated until all blocks specified in the function have been created.

```
[ ]: # The image display code below comes from Magno Naoli at:
     # https://stackoverflow.com/a/62985628/13097194
     Image(filename = 'annotated_arch_output.png')
```

[ ]:



Visual representation of calculations used in arch calculator program

By Kenneth Burchfiel

Notes:
1. This example uses a left height and diagonal height of 0.5 and a 15-degree angle (referred to as "rotation_degrees" in the code) for each side of each arch block. (See graphic on bottom right)
2. b1, b2, and b3 refer to the first, second, and third arch blocks, respectively.
3. p1, p2, p3, and p4 refer to the four corners of each arch.
4. sine(angle) provides the ratio of the opposite angle over the hypotenuse; cosine(angle) refers to to the adjacent angle over the hypotenuse.
5. All lengths are in meters.
6. The thin black lines show the actual arch blocks (rendered within Blender); the orange lines show the vertical and horizontal distances between the arch blocks' vertices.

b3.p2: (b2.p2.x +cos(15°)*0.5, b2.p2.y + sin(15°)*0.5)

b2.p2: (b1.p2.x +cos(45°)*0.5, b1.p2.y + sin(45°)*0.5)

b3.p1: (b2.p1.x +cos(15°)*rh, b2.p1.y + sin(15°)*rh)

b2.p1: (b1.p1.x +cos(45°)*rh, b1.p1.y + sin(45°)*rh)

b1.p2: (b1.p3.x +cos(75°)*0.5, b1.p3.y + sin(75°)*0.5)

b1.p1: (b1.p4.x +cos(75°)*rh, b1.p4.y + sin(75°)*rh)

b1.p3: (0, 2)

b1.p4: (0.5, 2)

Deriving the trapezoidal arch block from a rectangular block:

0.5 - 2*(sin(15°)*0.5) (right_height, abbreviated as rh on the left diagram)

The following code generates an ‘Arch_block’ class; each block of each arch is an instance of this class.

```
[ ]: class Arch_block:
         def __init__(self, left_height = 0.5, diagonal_length = 0.5,
         p4 = (0.5, 2), p3 = (0, 2), rotation_degrees = 15, rotation_multiplier = 1):
             '''left_height refers to the height of the left side of the block, and
             diagonal_length refers to the height of the bottom side of the block.
             See the above image for a reference.
             p4 and p3 refer to the bottom-right and bottom-left points of the block.
             rotation_degrees refers to the angle of the cut made to one side of
             the block to turn it into a wedge shape. See the diagram on the bottom
             right of the above image. Note that this cut is made to both sides of
             the block; as a result, the total rotation of the block equals twice
             the value of rotation_degrees.
             Rotation_multiplier is used in conjunction with rotation_degrees to
             calculate the angles of each block relative to the coordinate plane.
             Note that right_height, p1, and p2 are not passed as arguments, as
             their values are calculated based on the other values shown.
             '''
             self.left_height = left_height
             self.rotation_degrees = rotation_degrees
             self.diagonal_length = diagonal_length
```

```
        self.right_height = left_height - 2*(
            math.sin(rotation_degrees * dtr))*diagonal_length
            # See the bottom right diagram within the above image for a
            # visual representation of this calculation
        self.p4 = p4
        self.p3 = p3
        # Next, p1 and p2 are calculated based on p3 and p4. The above image
        # may make these calculations more intuitive.
        self.p1 = (self.p4[0] + math.cos((90-rotation_degrees *
        rotation_multiplier)*dtr) * self.right_height, self.p4[1] +
        math.sin((90 - rotation_degrees * rotation_multiplier)*dtr) *
        self.right_height)
        self.p2 = (self.p3[0] + math.cos((90-rotation_degrees *
        rotation_multiplier)*dtr) * self.left_height, self.p3[1] +
        math.sin((90 - rotation_degrees * rotation_multiplier)*dtr) *
        self.left_height)
    def __str__(self): # This function allows Arch_block objects to be
        # represented as a series of points when printed
        return '{} {} {} {} {} {} {} {}'.format("P1:", self.p1, "P2:",
        self.p2, "P3:", self.p3, "P4:", self.p4)
        # The .format() function converts the tuple values into a string.
        # See https://stackoverflow.com/a/39884000/13097194
```

The following function creates a series of Arch_block objects that form an arch or an angled wall.

```
[ ]: def create_arch(rotation_degrees, starting_p4, starting_p3, arch_count,␣
     ↪left_height, diagonal_length):
         ''' starting_p4 and starting_p3 refer to the bottom-right and bottom-left
         points, respectively, of the first Arch_block object. arch_count designates
         the number of Arch_block objects to be created.
         '''
         arch_block_list = []
         for i in range(arch_count):
             if i == 0: # The first block to be created will use starting_p4 and
                 # starting_p3 as its p4 and p3 coordinates.
                 arch_block_list.append(Arch_block(left_height = left_height,
                 diagonal_length=diagonal_length, p4 = starting_p4, p3=starting_p3,
                 rotation_degrees = rotation_degrees, rotation_multiplier = 1))
             else: # Each subsequent lbock will use the p2 and p1 coordinates of
                 # the last block to be created as its p3 and p4 coordinates,
                 # respectively.
                 arch_block_list.append(Arch_block(left_height = left_height,
                 diagonal_length=diagonal_length, p4 = arch_block_list[-1].p1,
                 p3=arch_block_list[-1].p2, rotation_degrees = rotation_degrees,
                 rotation_multiplier = 1 + i*2))
                 # The rotation_multiplier goes up by i*2 because each previous
```

```
            # block (other than the first) has been rotated by 2 times the
            # rotation_degrees value in order to align with the previous
            # blocks. The first block was only rotated by the rotation_degrees
            # value, hence the addition of 1 to rotation_multiplier.

    return arch_block_list
```

Finally, I'll create a function that displays all the coordinates of a given set of Arch_block objects, then plots those objects within a Matplotlib graph.

```
[ ]: def print_and_plot_points(arch_block_list, image_name):
         for i in range(len(arch_block_list)):
             # The following if and else statements calculate the lowest and
             # highest x and y values of all points within the arch_block_list.
             # This helps determine what x and y axis limits to use.
             if i == 0:
                 xmax = max([arch_block_list[i].p1[0], arch_block_list[i].p2[0],
                 arch_block_list[i].p3[0], arch_block_list[i].p4[0]])
                 xmin = min([arch_block_list[i].p1[0], arch_block_list[i].p2[0],
                 arch_block_list[i].p3[0], arch_block_list[i].p4[0]])
                 ymax = max([arch_block_list[i].p1[1], arch_block_list[i].p2[1],
                 arch_block_list[i].p3[1], arch_block_list[i].p4[1]])
                 ymin = min([arch_block_list[i].p1[1], arch_block_list[i].p2[1],
                 arch_block_list[i].p3[1], arch_block_list[i].p4[1]])
             else:
                 local_xmax = max([arch_block_list[i].p1[0],
                 arch_block_list[i].p2[0], arch_block_list[i].p3[0],
                 arch_block_list[i].p4[0]])
                 local_xmin = min([arch_block_list[i].p1[0],
                 arch_block_list[i].p2[0], arch_block_list[i].p3[0],
                 arch_block_list[i].p4[0]])
                 local_ymax = max([arch_block_list[i].p1[1],
                 arch_block_list[i].p2[1], arch_block_list[i].p3[1],
                 arch_block_list[i].p4[1]])
                 local_ymin = min([arch_block_list[i].p1[1],
                 arch_block_list[i].p2[1], arch_block_list[i].p3[1],
                 arch_block_list[i].p4[1]])
                 if local_xmax > xmax:
                     xmax = local_xmax
                 if local_xmin < xmin:
                     xmin = local_xmin
                 if local_ymax > ymax:
                     ymax = local_ymax
                 if local_ymin < ymin:
                     ymin = local_ymin
             print("Block number",str(i)+":")
             print(arch_block_list[i])
```

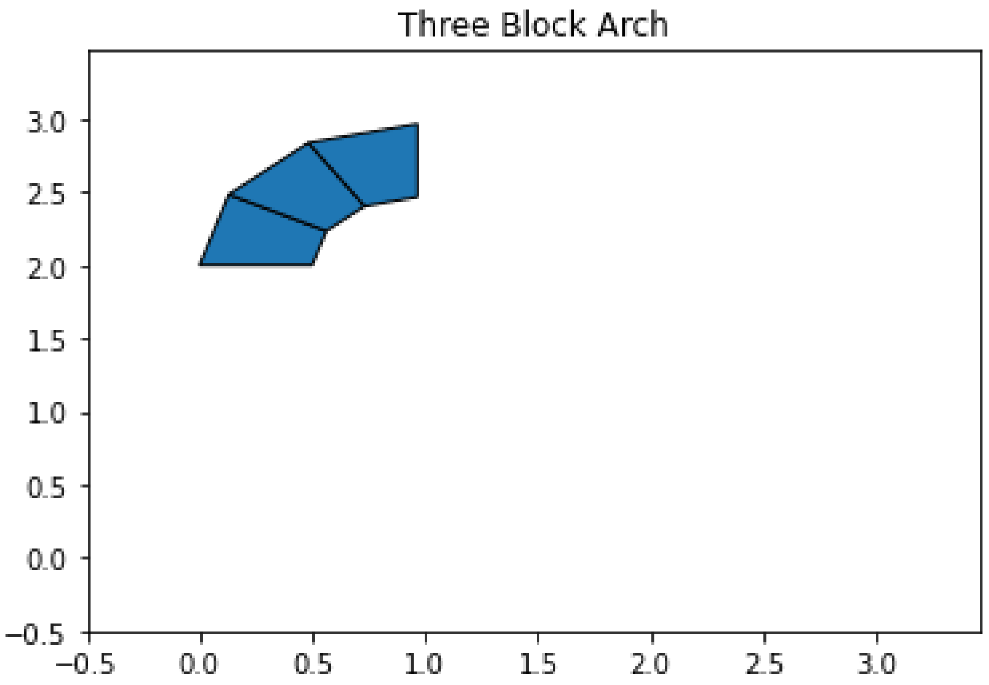
```
    # The Polygon plotting code below is based on Shrish's StackOverflow
    # answer at https://stackoverflow.com/a/68532480/13097194 .
    fig, ax = plt.subplots()
    fig.set_facecolor('white')
    for arch_block in arch_block_list:
        arch_block_polygon = Polygon([arch_block.p1, arch_block.p2,
        arch_block.p3, arch_block.p4], edgecolor = 'black', closed = 'True')
        ax.add_patch(arch_block_polygon)
    # If the distance between ymin and ymax is different from the distance
    # between xmin and xmax, the shape of the arch in the output could be
    # distorted. Therefore, I combined both min and max values into
    # 'universal' min and max values that can be used for both axes. This
    # ensures that the arches' shape will not be skewed by differing axis
    # lengths.
    universal_min = min(xmin, ymin)
    universal_max = max(xmax, ymax)
    plt.xlim(universal_min - 0.5, universal_max + 0.5)
    plt.ylim(universal_min -0.5, universal_max + 0.5)
    plt.title(image_name)
    file_save_name = str(image_name).lower().replace(' ','_')+'.png'
    # For instance, if image_name is "Three Block Arch," file_save_name
    # will be three_block_arch.
    plt.savefig(file_save_name)
    print("Saved image as:",file_save_name)
```

Now that these functions have been created, it's finally time to create different arches! The first arch to be created (three_block_arch) is the same as that shown in the example image near the beginning of this notebook. Only half of the arch is created since it's easy to copy, paste, and rotate this section to create the right half of the arch. Note that the total rotation represented by the arch is equal to rotation degrees * 2 * arch_count. For three_block_arch, each side of each block within the arch has been cut at a 15 degree angle, amounting to 30 degrees of rotation per block. 30 degrees * 3 = 90 degrees, which is why the final edge of the 3rd block is vertical.

[ ]:
```
three_block_arch = create_arch(rotation_degrees = 15,
starting_p4 = (0.5, 2), starting_p3 = (0, 2), left_height = 0.5,
diagonal_length = 0.5, arch_count = 3)
print_and_plot_points(three_block_arch,'Three Block Arch')
```

```
Block number 0:
P1: (0.5624222244434797, 2.2329629131445343) P2: (0.12940952255126037,
2.4829629131445343) P3: (0, 2) P4: (0.5, 2)
Block number 1:
P1: (0.7329629131445341, 2.4035036018455886) P2: (0.48296291314453416,
2.8365163037378083) P3: (0.12940952255126037, 2.4829629131445343) P4:
(0.5624222244434797, 2.2329629131445343)
Block number 2:
P1: (0.9659258262890683, 2.465925826289068) P2: (0.9659258262890683,
2.9659258262890686) P3: (0.48296291314453416, 2.8365163037378083) P4:
```

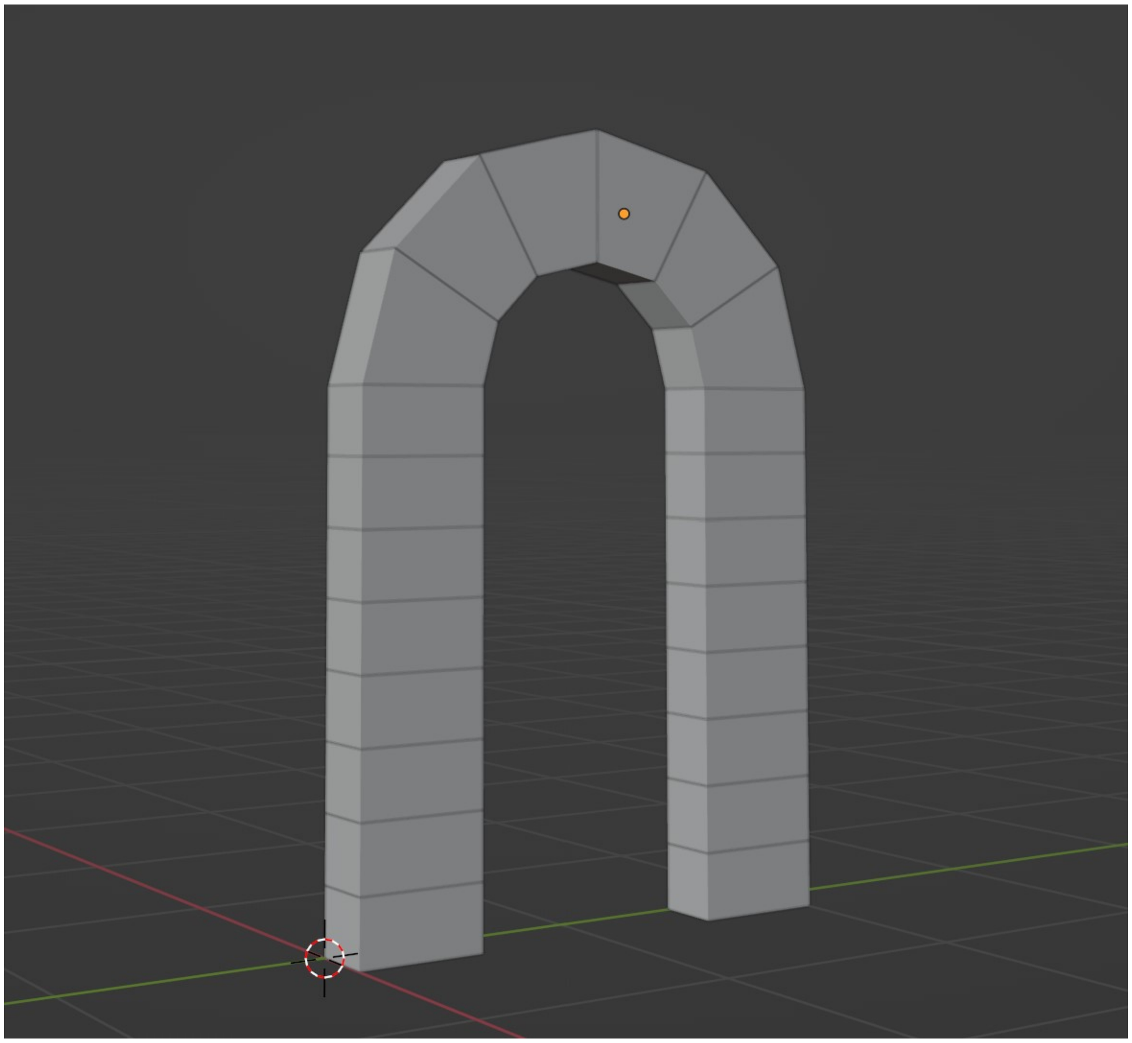
```
(0.7329629131445341, 2.4035036018455886)
Saved image as: three_block_arch.png
```



I copied and pasted these coordinates into Blender in order to position each arch segment. Next, I copied, pasted, and rotated the entire segment in order to create the right segment of the arch. I tweaked the x positions of the two vertical edges so that they would better align with the coordinate space in my Blender file. (I believe that it's possible to do all of these steps within Python as well, but I haven't yet learned how to take full advantage of Blender's Python interface.)

The image below shows what this arch looks like in Blender. The arch is placed on top of eight 0.25-meter-tall blocks on each side, which is why the arch calculations begin at a 2-meter height. Each arch component is 0.25 meters thick, resulting in a 3D effect.

```
[ ]: Image(filename = 'completed_arch_with_tweaks.jpg')
[ ]:
```

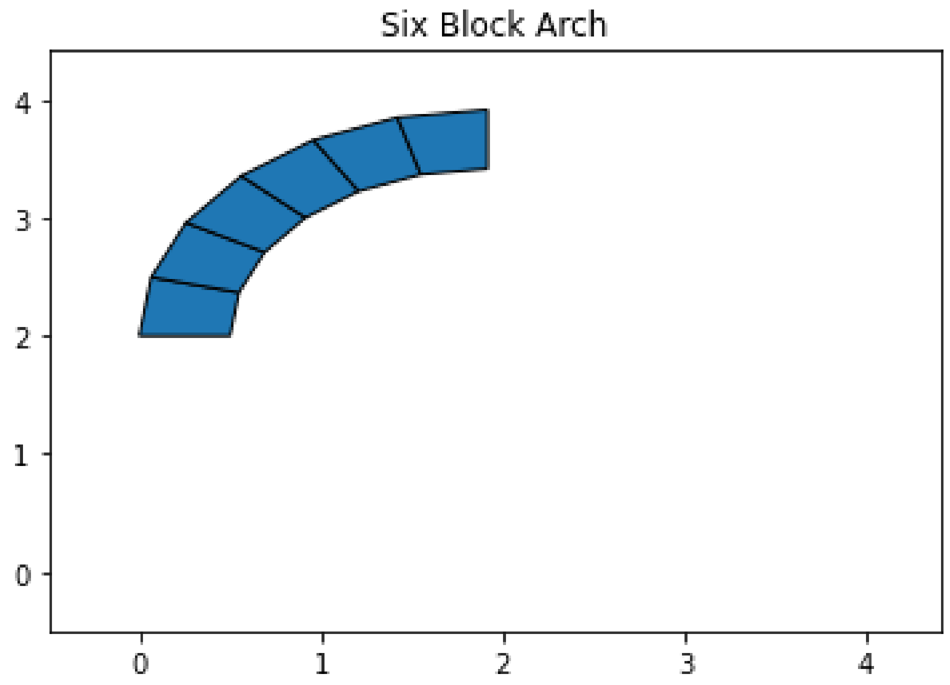


The same create_arch function can be used to generate other arch forms as well, such as a 6-block arch. The rotation_degrees value was cut in half so that the final edge would still be vertical.

```
[ ]: six_block_arch = create_arch(rotation_degrees = 7.5,
     starting_p4 = (0.5, 2), starting_p3 = (0, 2), left_height = 0.5,
     diagonal_length = 0.5, arch_count = 6)
     print_and_plot_points(six_block_arch, 'Six Block Arch')
```

```
Block number 0:
P1: (0.54822600925456, 2.3663129081356447) P2: (0.06526309611002586,
2.495722430686905) P3: (0, 2) P4: (0.5, 2)
Block number 1:
P1: (0.6896175141847901, 2.7076621969425485) P2: (0.25660481229257076,
2.9576621969425485) P3: (0.06526309611002586, 2.495722430686905) P4:
(0.54822600925456, 2.3663129081356447)
Block number 2:
```

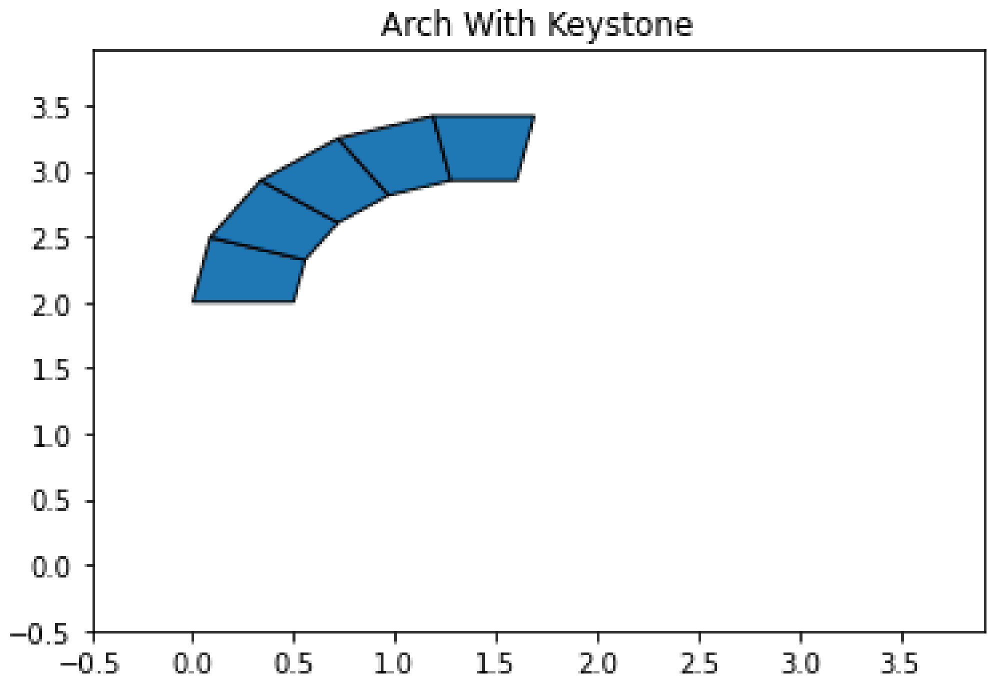
```
P1: (0.9145389173902049, 3.0007854764948925) P2: (0.560985526796931,
3.354338867088166) P3: (0.25660481229257076, 2.9576621969425485) P4:
(0.6896175141847901, 2.7076621969425485)
Block number 3:
P1: (1.2076621969425487, 3.2257068797003075) P2: (0.9576621969425486,
3.658719581592526) P3: (0.560985526796931, 3.354338867088166) P4:
(0.9145389173902049, 3.0007854764948925)
Block number 4:
P1: (1.5490114857494526, 3.3670983846305376) P2: (1.419601963198192,
3.850061297775071) P3: (0.9576621969425486, 3.658719581592526) P4:
(1.2076621969425487, 3.2257068797003075)
Block number 5:
P1: (1.9153243938850975, 3.4153243938850975) P2: (1.9153243938850972,
3.915324393885097) P3: (1.419601963198192, 3.850061297775071) P4:
(1.5490114857494526, 3.3670983846305376)
Saved image as: six_block_arch.png
```



I doubt that an arch with two adjacent vertical edges would be very stable structurally, however, since one of the two stones with a vertical edge could slide out of the arch. It seems safer to build an arch with a keystone in the middle. The following code produces such an arch.

[ ]:
```
arch_with_keystone = create_arch(rotation_degrees = 10,
starting_p4 = (0.5, 2), starting_p3 = (0, 2), left_height = 0.5,
diagonal_length = 0.5, arch_count = 5)
print_and_plot_points(arch_with_keystone, 'Arch With Keystone')
```

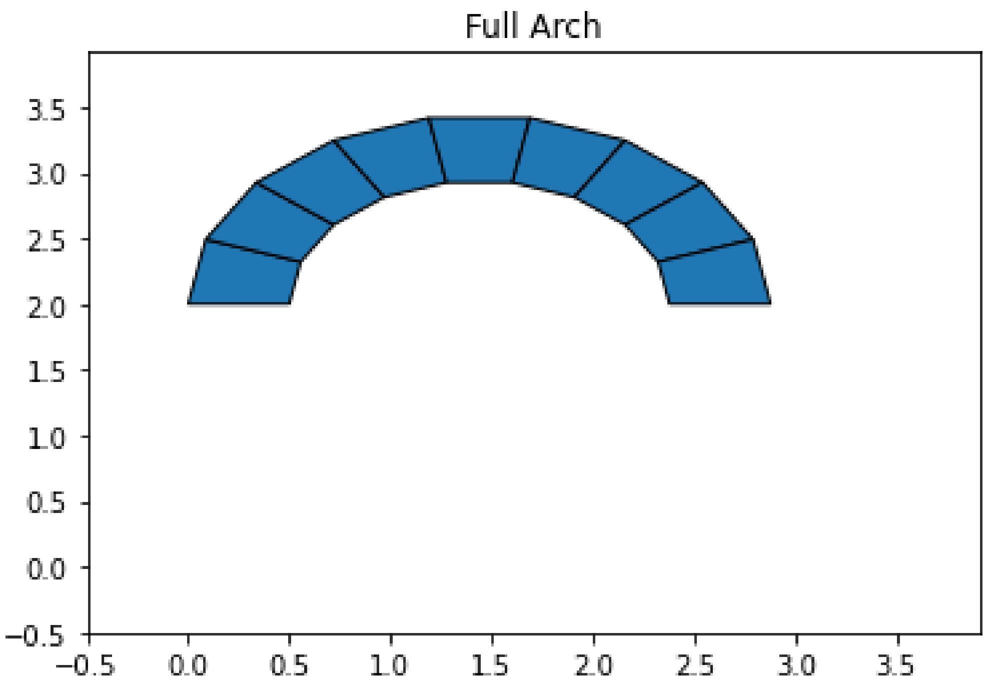
```
Block number 0:
P1: (0.5566703992264194, 2.32139380484327) P2: (0.08682408883346521,
2.492403876506104) P3: (0, 2) P4: (0.5, 2)
Block number 1:
P1: (0.7198463103929542, 2.604022773555054) P2: (0.33682408883346526,
2.9254165783983233) P3: (0.08682408883346521, 2.492403876506104) P4:
(0.5566703992264194, 2.32139380484327)
Block number 2:
P1: (0.9698463103929542, 2.813797681349374) P2: (0.7198463103929542,
3.246810383241593) P3: (0.33682408883346526, 2.9254165783983233) P4:
(0.7198463103929542, 2.604022773555054)
Block number 3:
P1: (1.2765167096193737, 2.9254165783983237) P2: (1.1896926207859084,
3.4178204549044273) P3: (0.7198463103929542, 3.246810383241593) P4:
(0.9698463103929542, 2.813797681349374)
Block number 4:
P1: (1.6028685319524434, 2.9254165783983237) P2: (1.6896926207859084,
3.4178204549044273) P3: (1.1896926207859084, 3.4178204549044273) P4:
(1.2765167096193737, 2.9254165783983237)
Saved image as: arch_with_keystone.png
```



If you wish to create an entire arch rather than just half of one, you can simply increase the value of arch_count:

```
[ ]: full_arch = create_arch(rotation_degrees = 10,
     starting_p4 = (0.5, 2), starting_p3 = (0, 2), left_height = 0.5,
     diagonal_length = 0.5, arch_count = 9)
     print_and_plot_points(full_arch, 'Full Arch')
```

```
Block number 0:
P1: (0.5566703992264194, 2.32139380484327) P2: (0.08682408883346521,
2.492403876506104) P3: (0, 2) P4: (0.5, 2)
Block number 1:
P1: (0.7198463103929542, 2.604022773555054) P2: (0.33682408883346526,
2.9254165783983233) P3: (0.08682408883346521, 2.492403876506104) P4:
(0.5566703992264194, 2.32139380484327)
Block number 2:
P1: (0.9698463103929542, 2.813797681349374) P2: (0.7198463103929542,
3.246810383241593) P3: (0.33682408883346526, 2.9254165783983233) P4:
(0.7198463103929542, 2.604022773555054)
Block number 3:
P1: (1.2765167096193737, 2.9254165783983237) P2: (1.1896926207859084,
3.4178204549044273) P3: (0.7198463103929542, 3.246810383241593) P4:
(0.9698463103929542, 2.813797681349374)
Block number 4:
P1: (1.6028685319524434, 2.9254165783983237) P2: (1.6896926207859084,
3.4178204549044273) P3: (1.1896926207859084, 3.4178204549044273) P4:
(1.2765167096193737, 2.9254165783983237)
Block number 5:
P1: (1.9095389311788629, 2.813797681349374) P2: (2.1595389311788624,
3.246810383241593) P3: (1.6896926207859084, 3.4178204549044273) P4:
(1.6028685319524434, 2.9254165783983237)
Block number 6:
P1: (2.159538931178863, 2.604022773555054) P2: (2.5425611527383514,
2.9254165783983233) P3: (2.1595389311788624, 3.246810383241593) P4:
(1.9095389311788629, 2.813797681349374)
Block number 7:
P1: (2.322714842345398, 2.32139380484327) P2: (2.7925611527383514,
2.492403876506104) P3: (2.5425611527383514, 2.9254165783983233) P4:
(2.159538931178863, 2.604022773555054)
Block number 8:
P1: (2.3793852415718173, 2.0) P2: (2.8793852415718164, 2.0) P3:
(2.7925611527383514, 2.492403876506104) P4: (2.322714842345398,
2.32139380484327)
Saved image as: full_arch.png
```

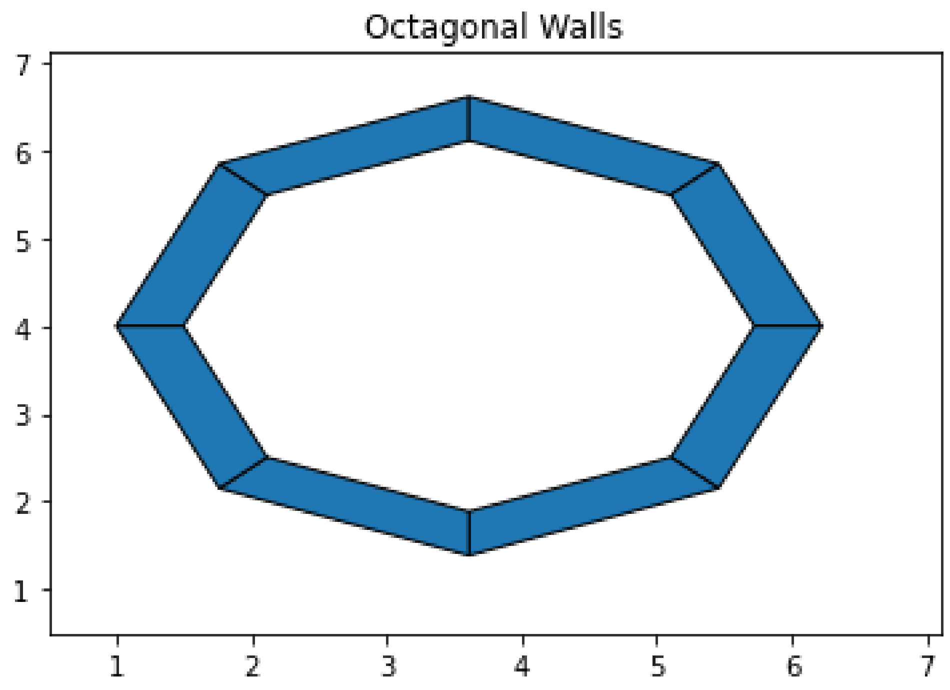


Finally, this code can also be applied to calculating the coordinates of a room with angled walls. Suppose that you wanted to create an octagonal room. Given that a circle is 360 degrees and you'd have 8 walls, each wall would need to rotate 45 degrees. (Since each segment of the wall would have two diagonal sides, these sides would each need to be 22.5 degrees.) By plugging in 22.5 as the rotation_degrees value and 8 as the arch_count, you can easily calculate the coordinates of each 2D component of this wall from a top-down perspective.

```
[ ]: octagonal_walls = create_arch(rotation_degrees = 22.5,
     starting_p4 = (1.5, 4), starting_p3 = (1, 4), left_height = 2,
     diagonal_length = 0.5, arch_count = 8)
     print_and_plot_points(octagonal_walls,'Octagonal Walls')
```

```
Block number 0:
P1: (2.118920255323453, 5.4942056744293) P2: (1.7653668647301797,
5.847759065022574) P3: (1, 4) P4: (1.5, 4)
Block number 1:
P1: (3.613125929752753, 6.113125929752753) P2: (3.613125929752753,
6.613125929752753) P3: (1.7653668647301797, 5.847759065022574) P4:
(2.118920255323453, 5.4942056744293)
Block number 2:
P1: (5.107331604182053, 5.4942056744293) P2: (5.460884994775327,
5.847759065022574) P3: (3.613125929752753, 6.613125929752753) P4:
(3.613125929752753, 6.113125929752753)
Block number 3:
P1: (5.726251859505506, 4.0) P2: (6.226251859505506, 4.0) P3:
```

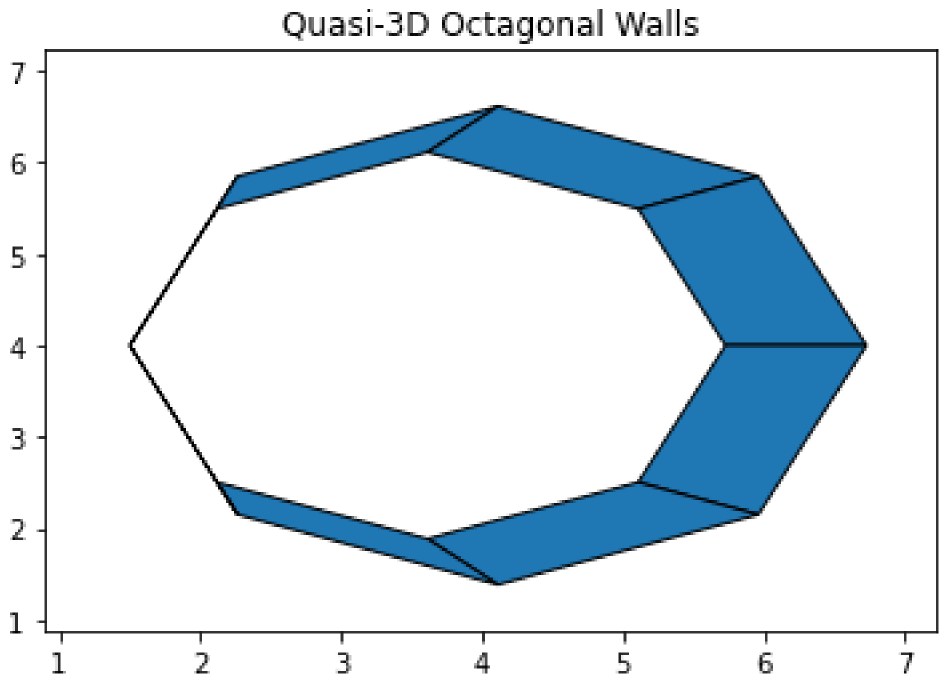
```
(5.460884994775327, 5.847759065022574) P4: (5.107331604182053, 5.4942056744293)
Block number 4:
P1: (5.107331604182053, 2.5057943255707) P2: (5.460884994775327,
2.1522409349774265) P3: (6.226251859505506, 4.0) P4: (5.726251859505506, 4.0)
Block number 5:
P1: (3.613125929752753, 1.8868740702472464) P2: (3.6131259297527536,
1.3868740702472468) P3: (5.460884994775327, 2.1522409349774265) P4:
(5.107331604182053, 2.5057943255707)
Block number 6:
P1: (2.118920255323453, 2.5057943255706996) P2: (1.76536686473018,
2.152240934977426) P3: (3.6131259297527536, 1.3868740702472468) P4:
(3.613125929752753, 1.8868740702472464)
Block number 7:
P1: (1.5000000000000004, 3.999999999999999) P2: (1.0000000000000009, 4.0) P3:
(1.76536686473018, 2.152240934977426) P4: (2.118920255323453,
2.5057943255706996)
Saved image as: octagonal_walls.png
```



Finally, and just for fun, you can also achieve a quasi-3D effect by setting starting_p4 and starting_p3 equal to one another.

[ ]:
```
quasi_3d_octagonal_walls = create_arch(rotation_degrees = 22.5,
starting_p4 = (1.5, 4), starting_p3 = (1.5, 4), left_height = 2,
diagonal_length = 0.5, arch_count = 8)
print_and_plot_points(quasi_3d_octagonal_walls,'Quasi-3D Octagonal Walls')
```

```
Block number 0:
P1: (2.118920255323453, 5.4942056744293) P2: (2.2653668647301797,
5.847759065022574) P3: (1.5, 4) P4: (1.5, 4)
Block number 1:
P1: (3.613125929752753, 6.113125929752753) P2: (4.113125929752753,
6.613125929752753) P3: (2.2653668647301797, 5.847759065022574) P4:
(2.118920255323453, 5.4942056744293)
Block number 2:
P1: (5.107331604182053, 5.4942056744293) P2: (5.960884994775327,
5.847759065022574) P3: (4.113125929752753, 6.613125929752753) P4:
(3.613125929752753, 6.113125929752753)
Block number 3:
P1: (5.726251859505506, 4.0) P2: (6.726251859505506, 4.0) P3:
(5.960884994775327, 5.847759065022574) P4: (5.107331604182053, 5.4942056744293)
Block number 4:
P1: (5.107331604182053, 2.5057943255707) P2: (5.960884994775327,
2.1522409349774265) P3: (6.726251859505506, 4.0) P4: (5.726251859505506, 4.0)
Block number 5:
P1: (3.613125929752753, 1.8868740702472464) P2: (4.113125929752753,
1.3868740702472468) P3: (5.960884994775327, 2.1522409349774265) P4:
(5.107331604182053, 2.5057943255707)
Block number 6:
P1: (2.118920255323453, 2.5057943255706996) P2: (2.2653668647301792,
2.152240934977426) P3: (4.113125929752753, 1.3868740702472468) P4:
(3.613125929752753, 1.8868740702472464)
Block number 7:
P1: (1.5000000000000004, 3.999999999999999) P2: (1.5000000000000002, 4.0) P3:
(2.2653668647301792, 2.152240934977426) P4: (2.118920255323453,
2.5057943255706996)
Saved image as: quasi-3d_octagonal_walls.png
```



I hope this code will help you with your own 3D modeling projects!