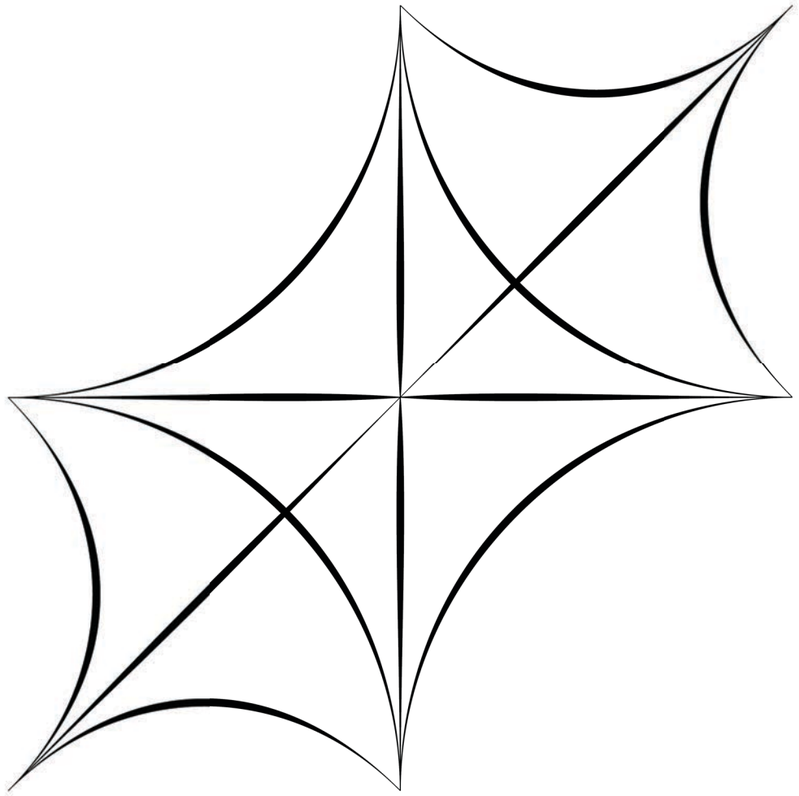
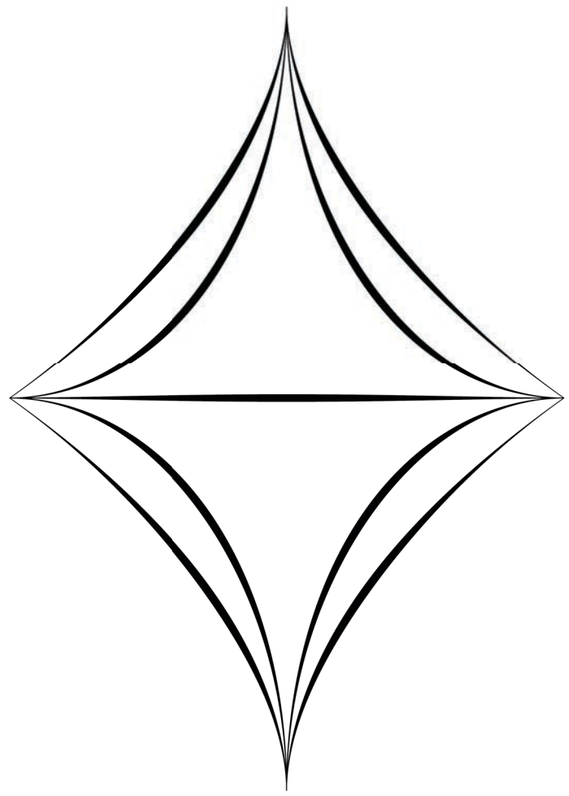
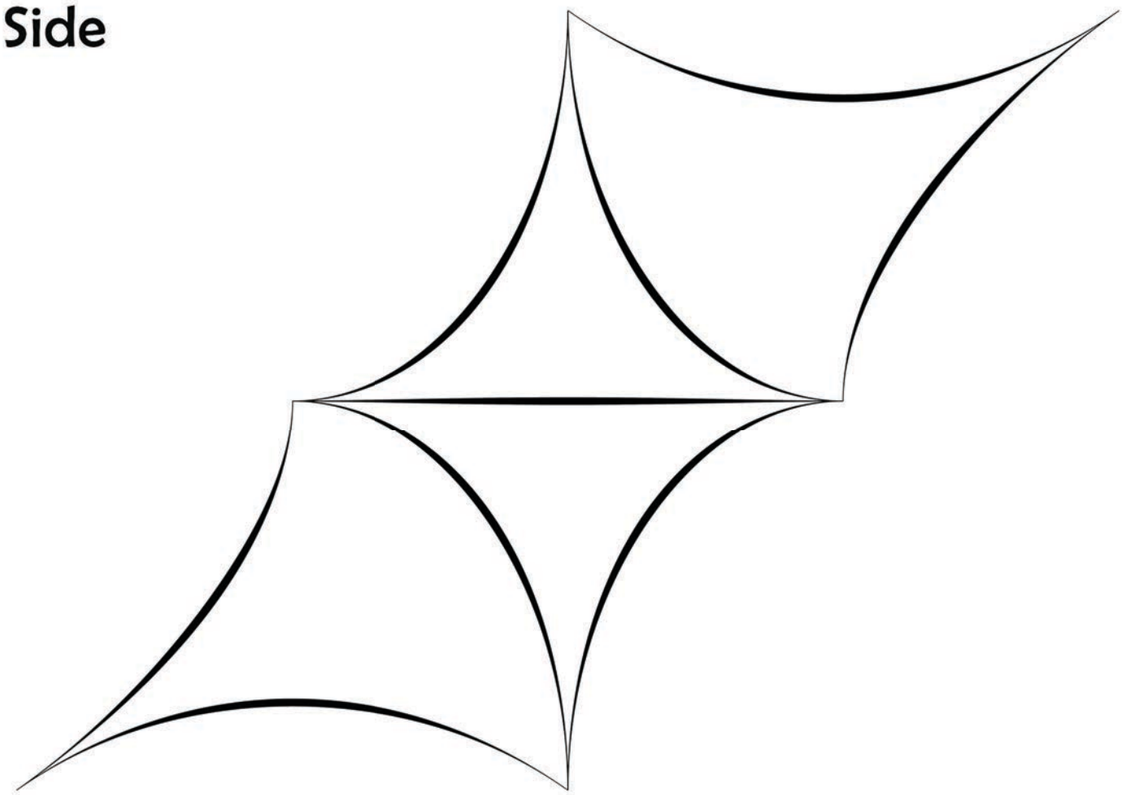
Top



Front



Side



Wave Conjugations Flower of Life

Author Name
Terrence Howard
& David Johnson

Creation Date
Jan 20, 2019

Format:
A3

Description:
Illustrations - Work Book

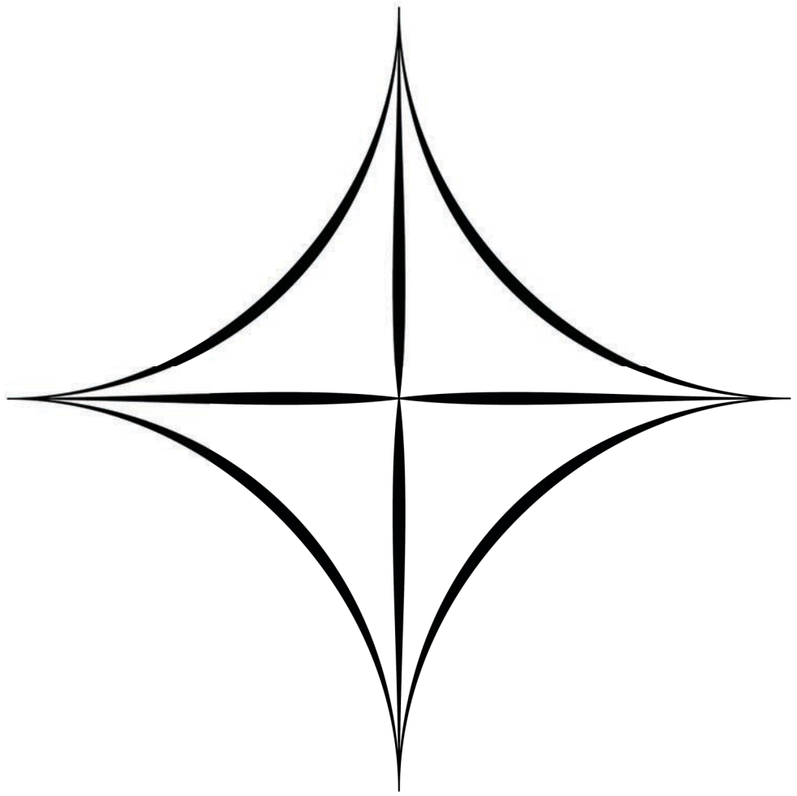
Draftsman:
/dj

Produced by:
Argos.Vu

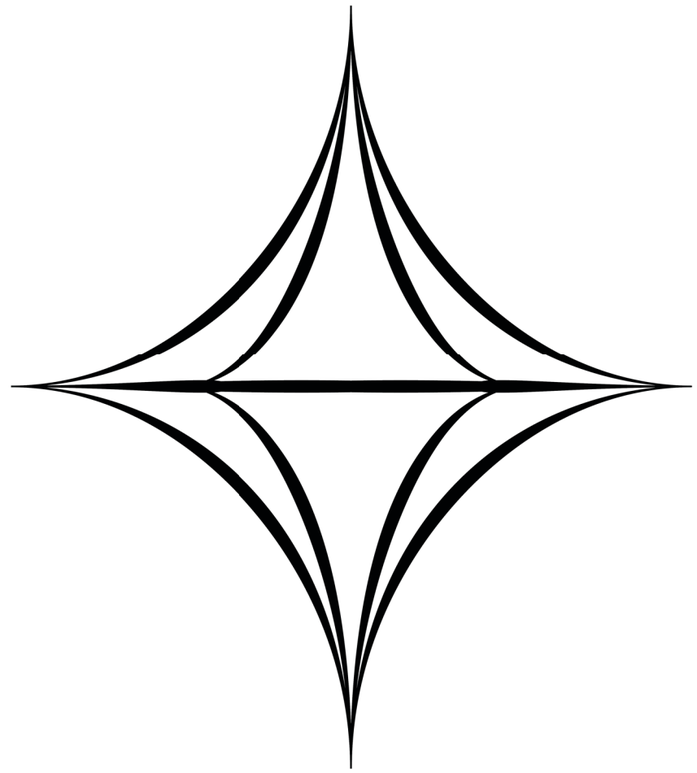
Version:
001

Page:
1

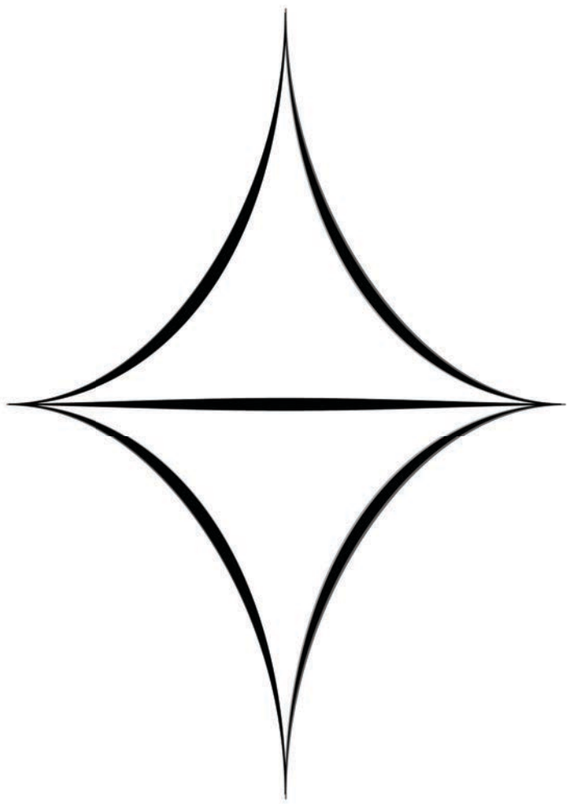
Top



Front

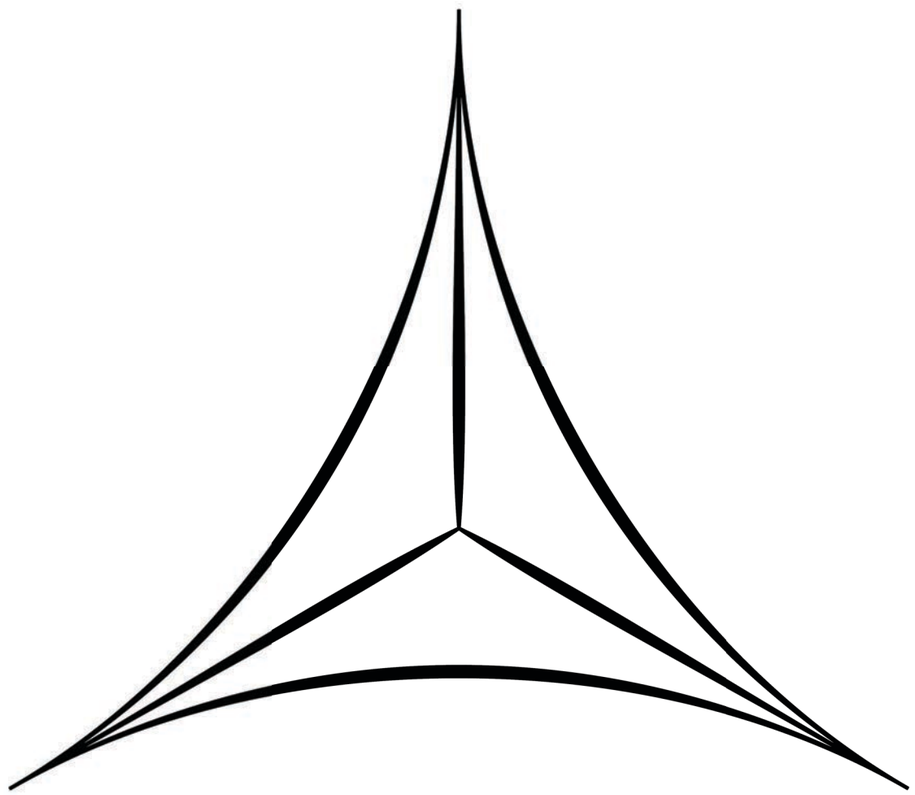


Side

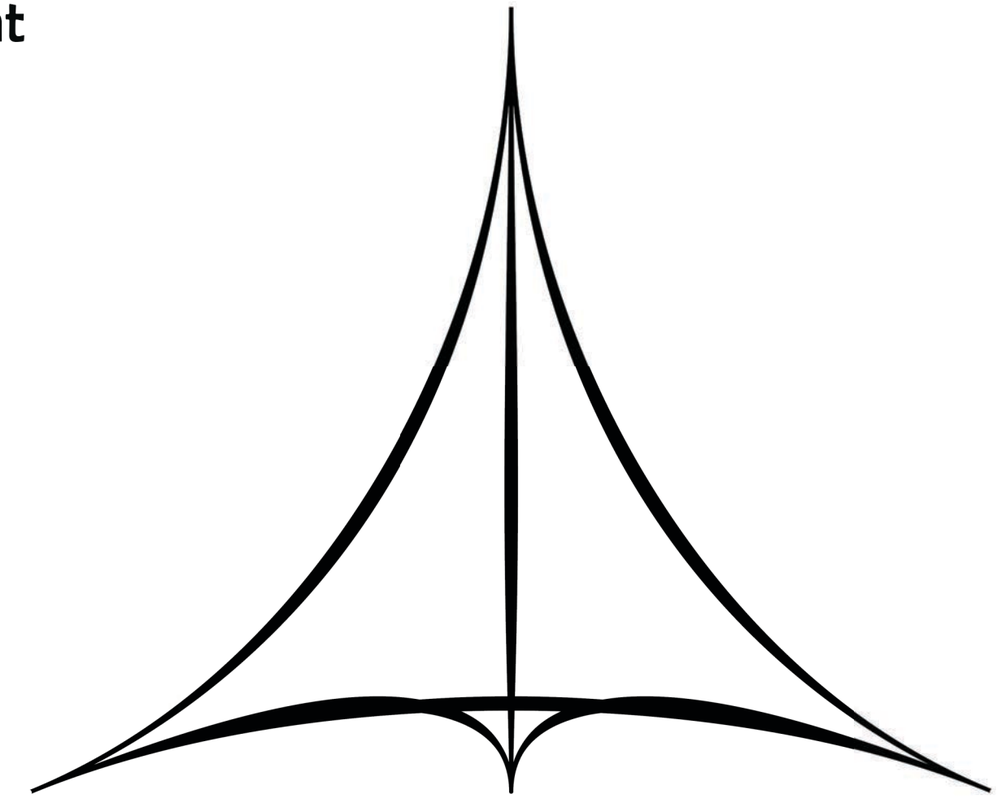


Author Name Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 2
Copyright: Terrence Howard			

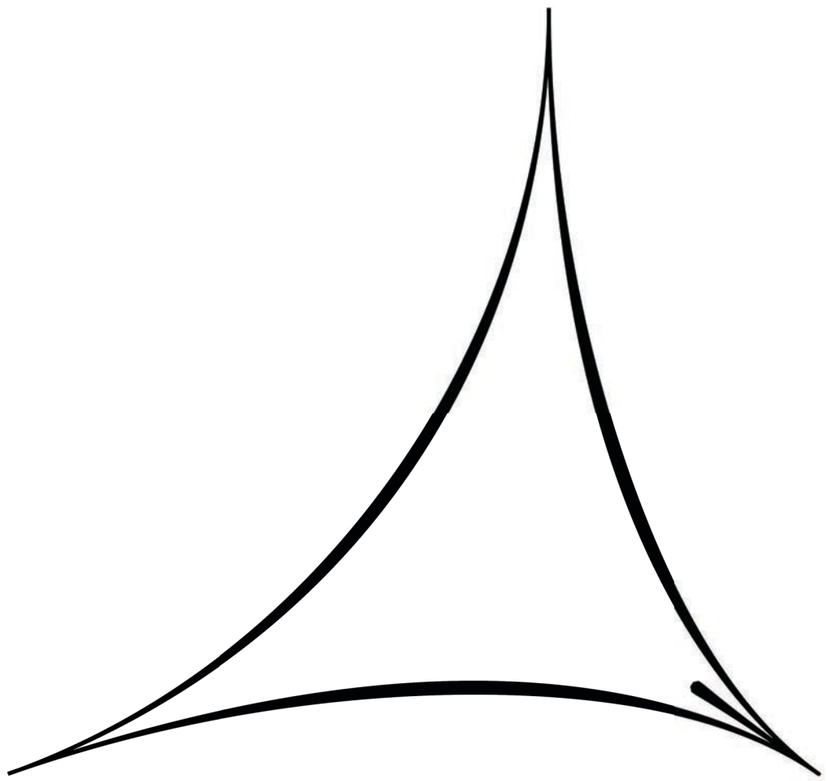
Top



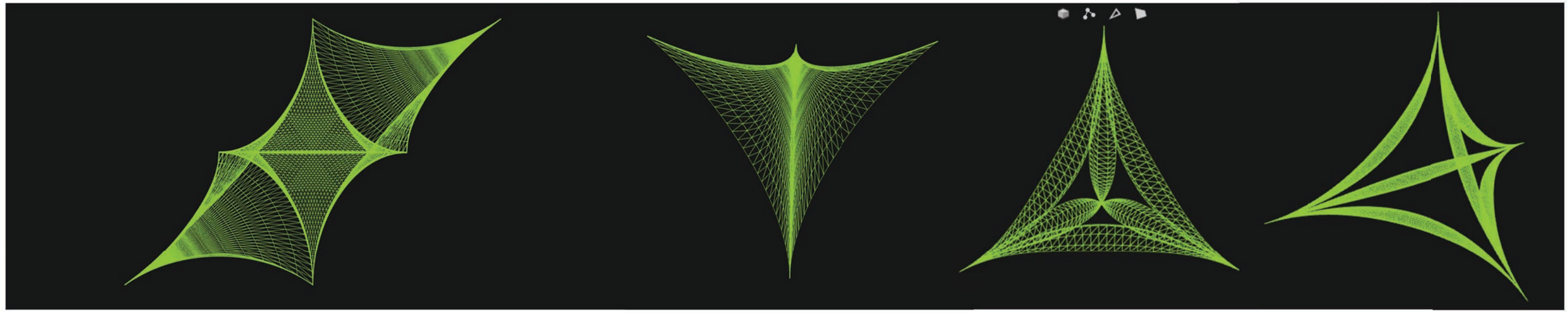
Front

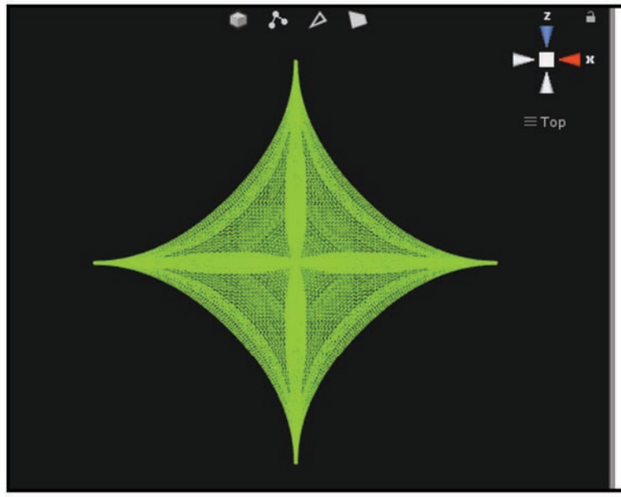


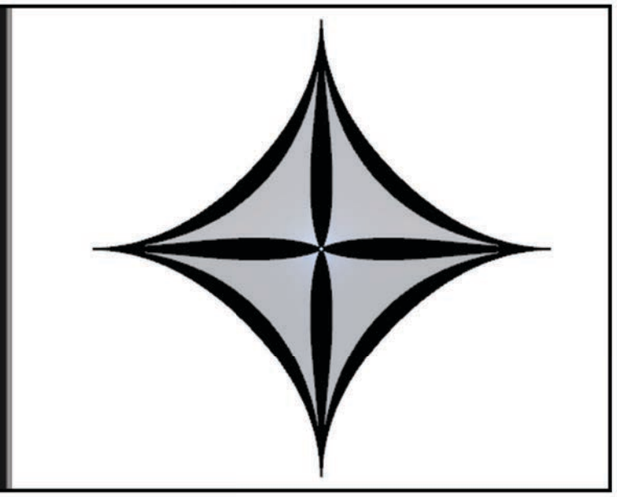
Side

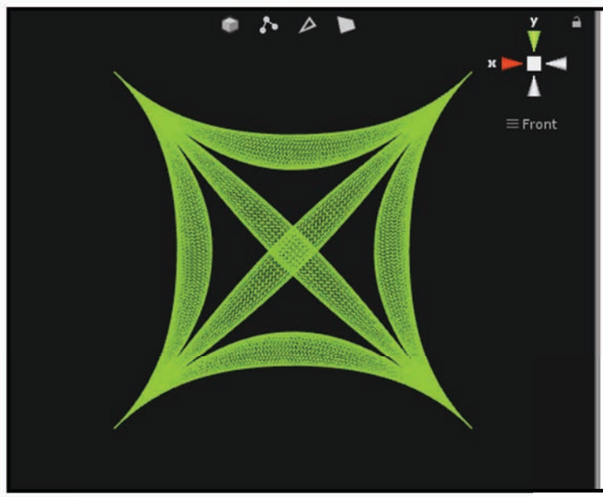


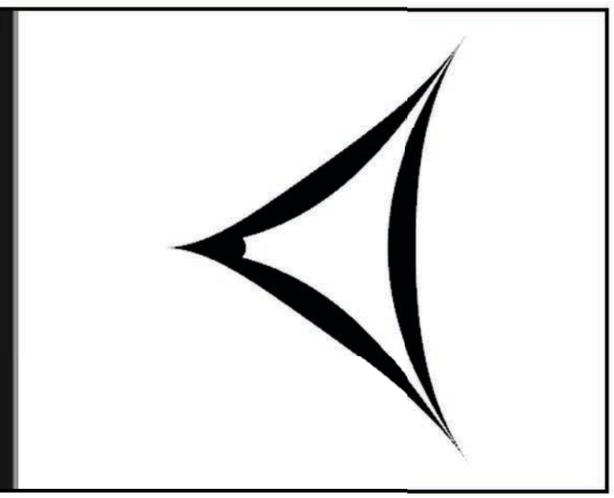
Author Name: Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date: Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 3

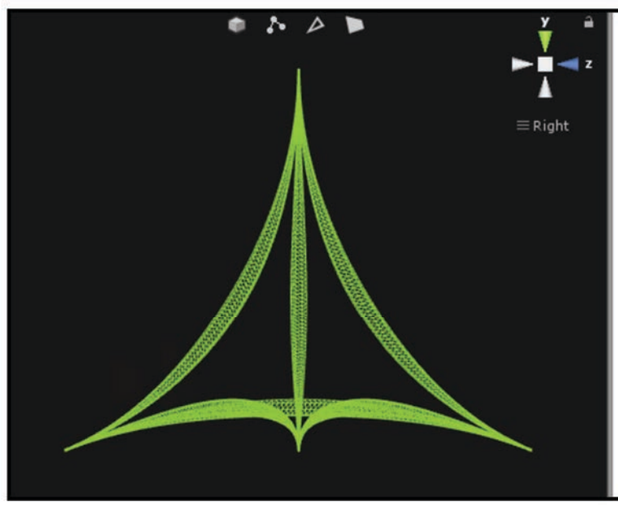


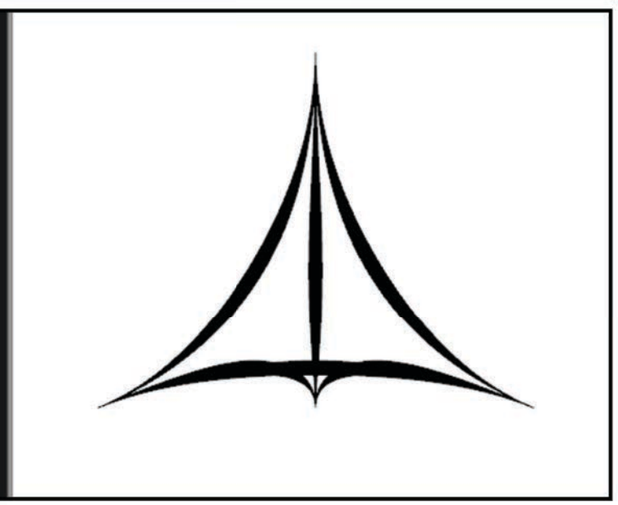




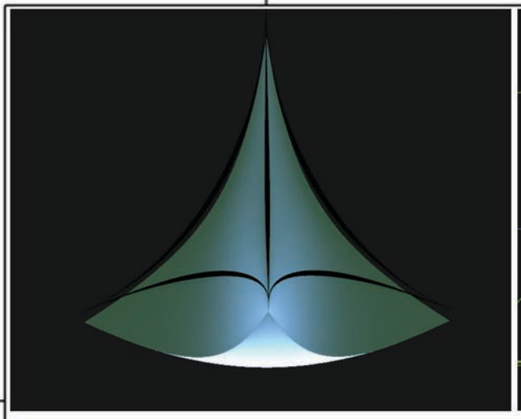


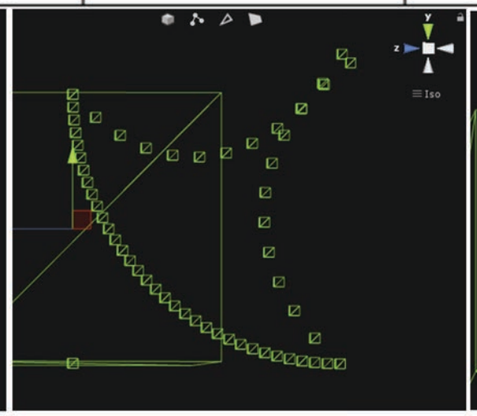


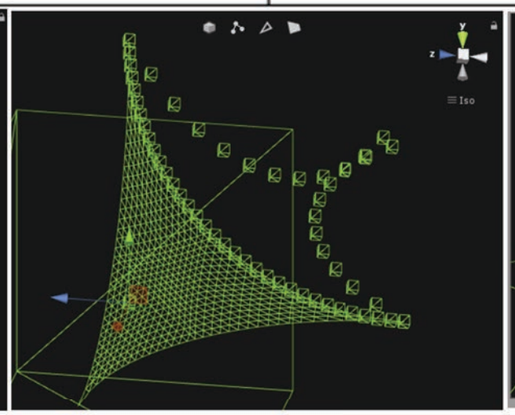


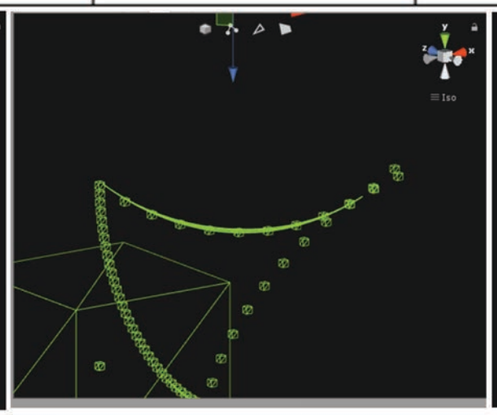


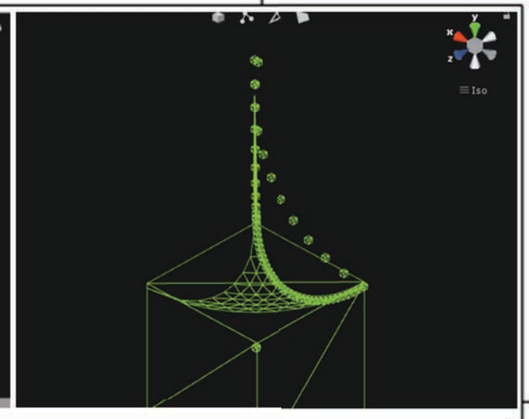
Author Name Terrence Howard & David Johnson		# Wave Conjugations # Flower of Life	
Creation Date Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 4

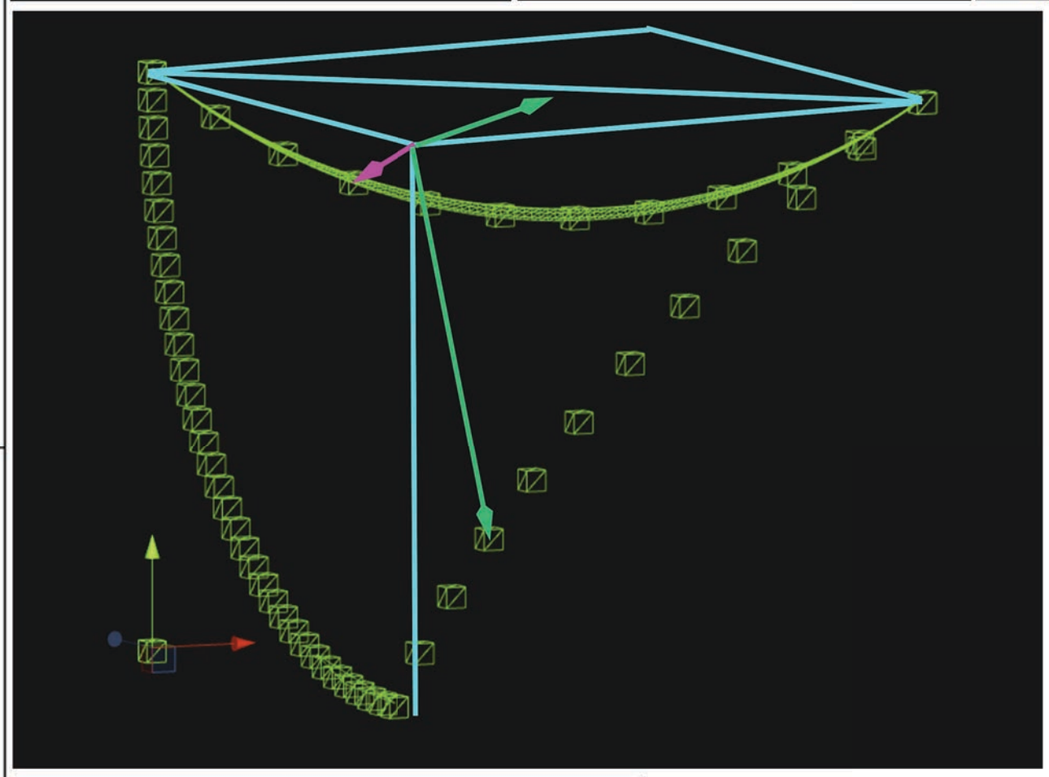


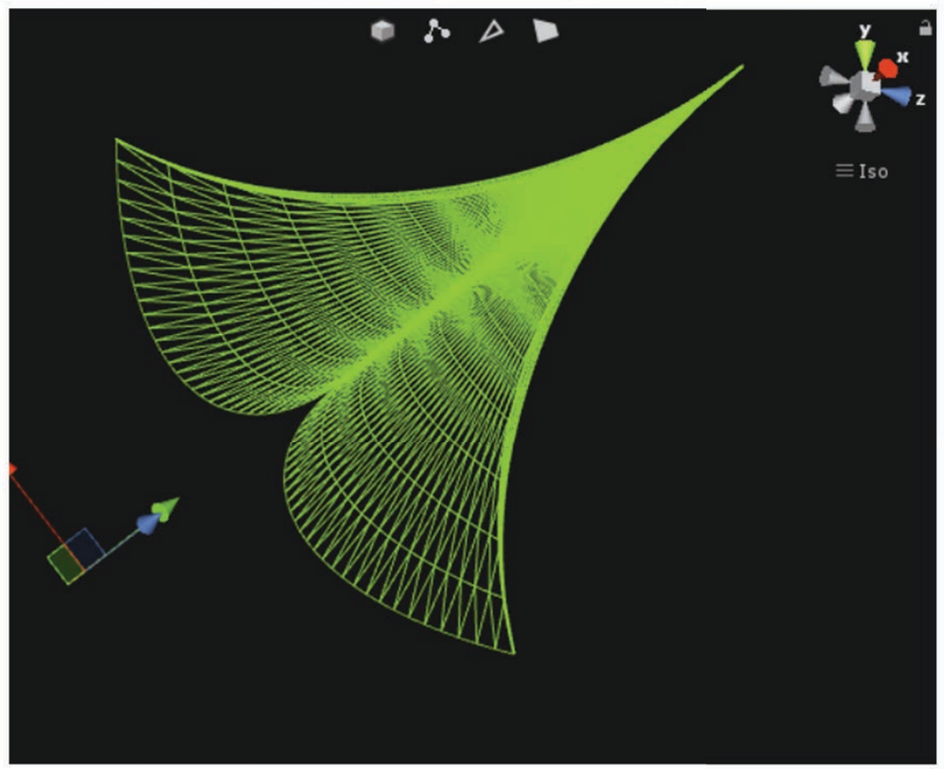


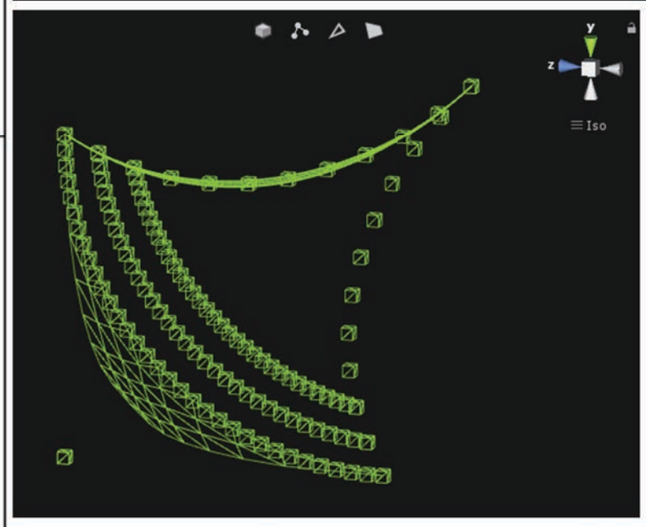


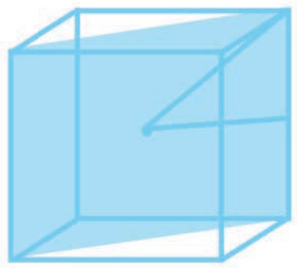




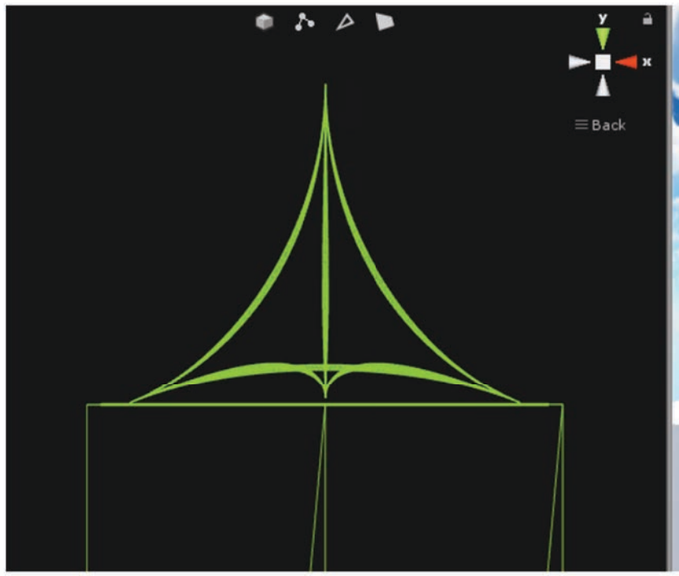


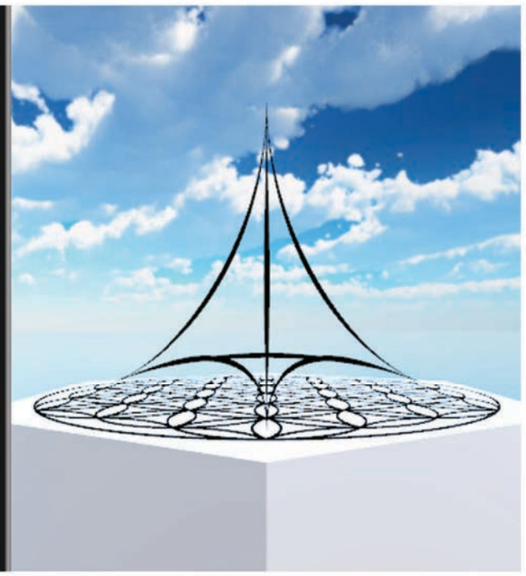


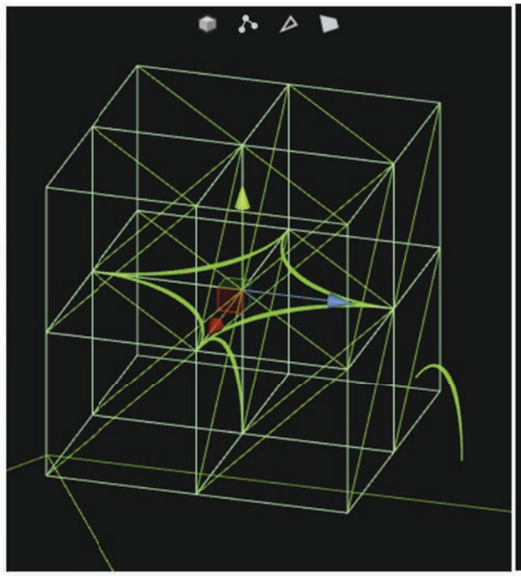


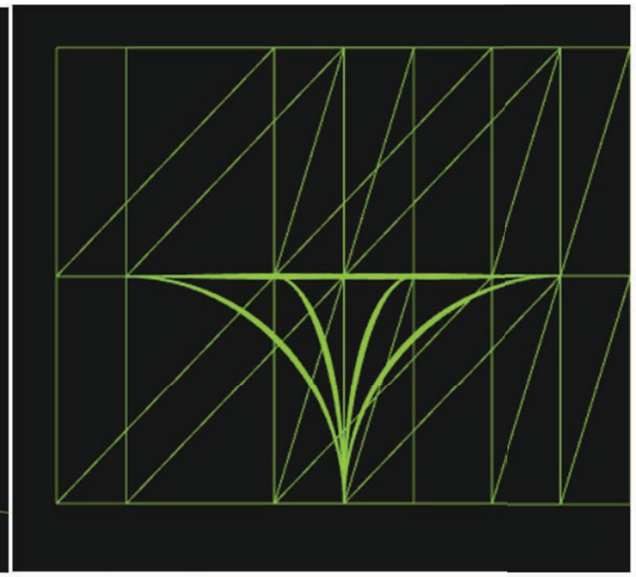


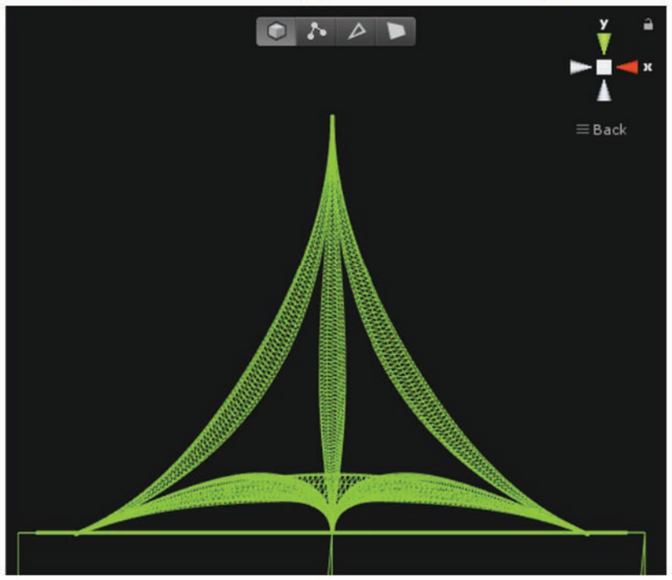
Author Name Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 5

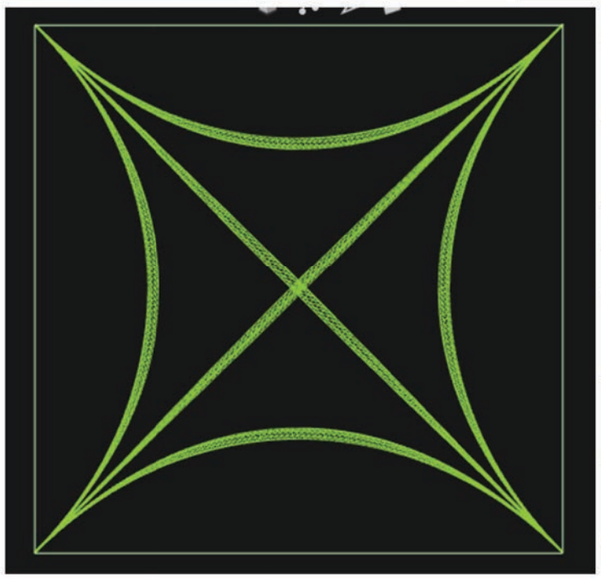


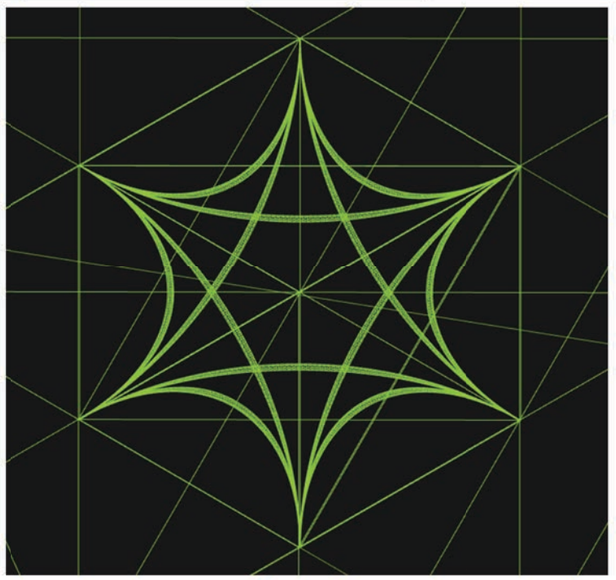


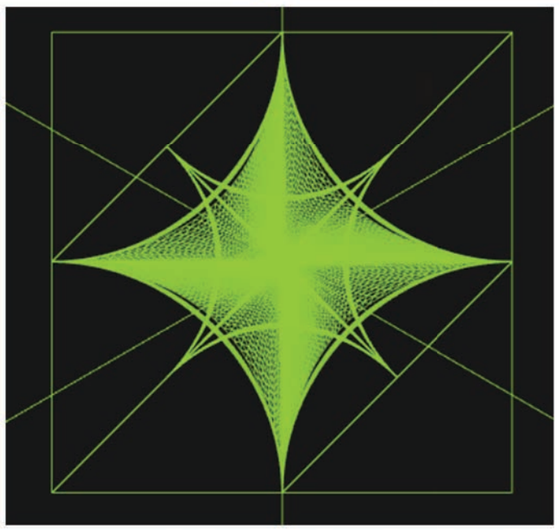




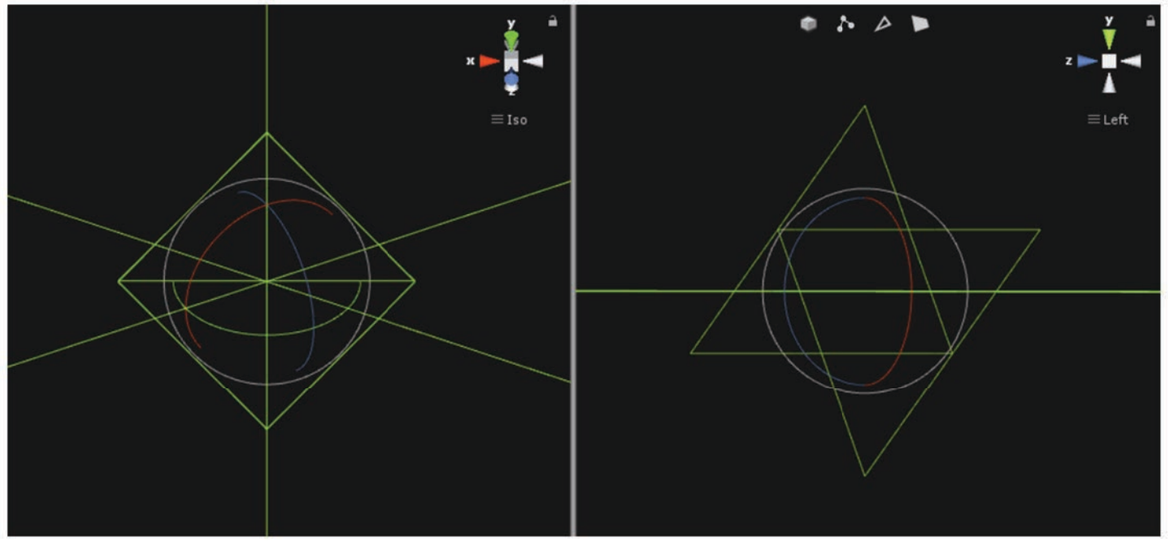


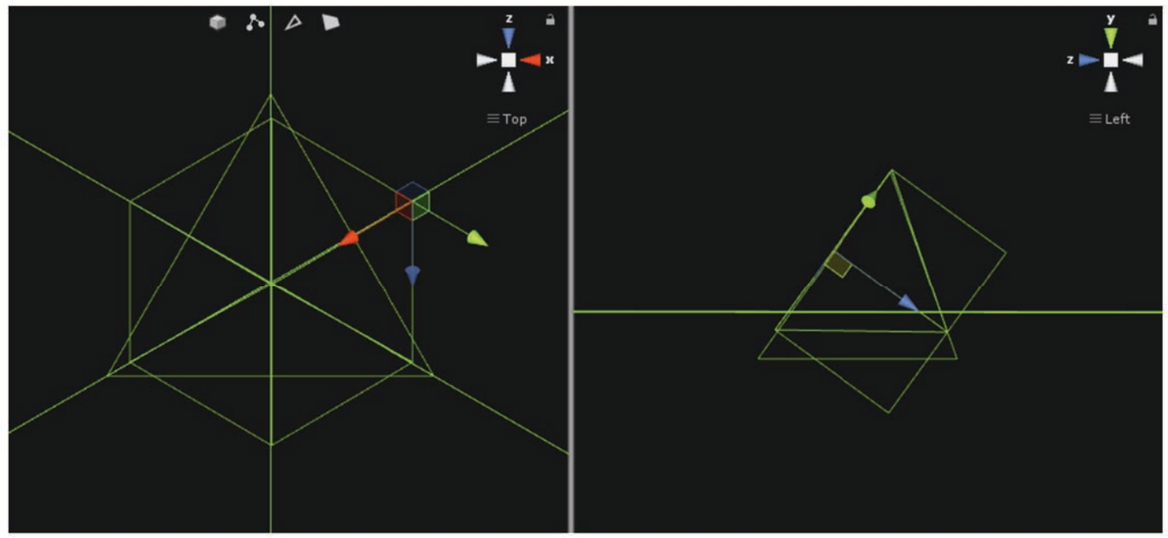


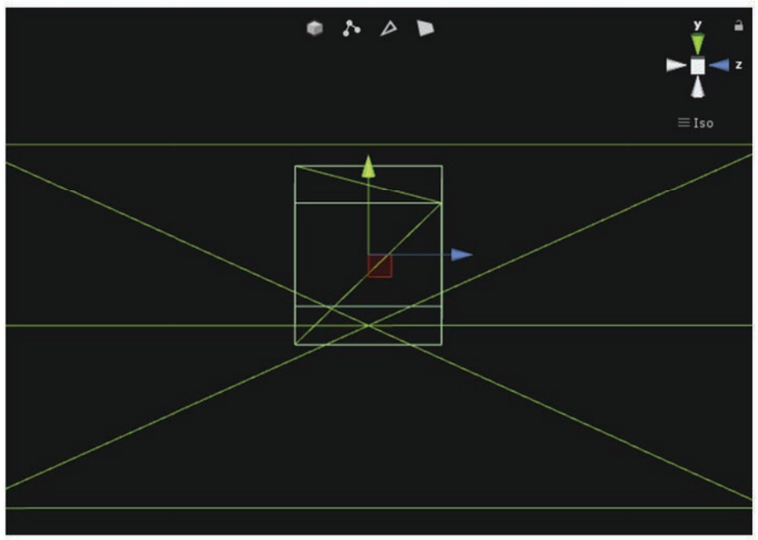


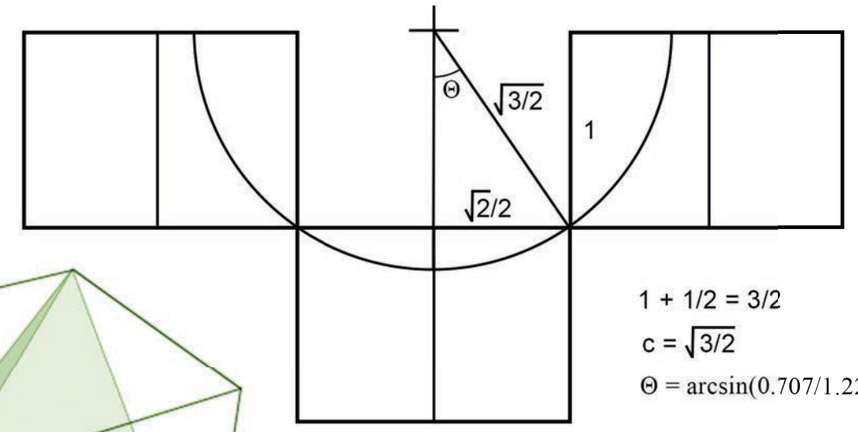


Author Name: Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date: Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 6







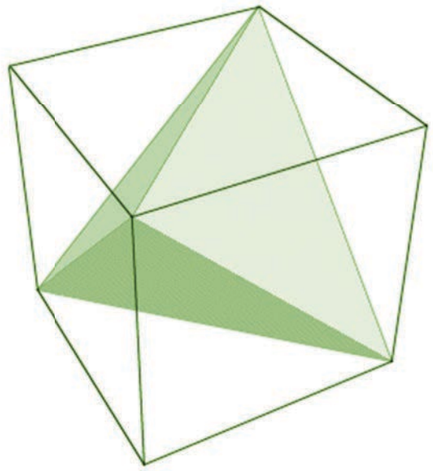


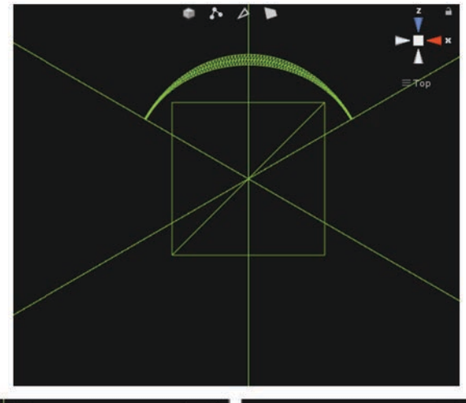
1 + 1/2 = 3/2

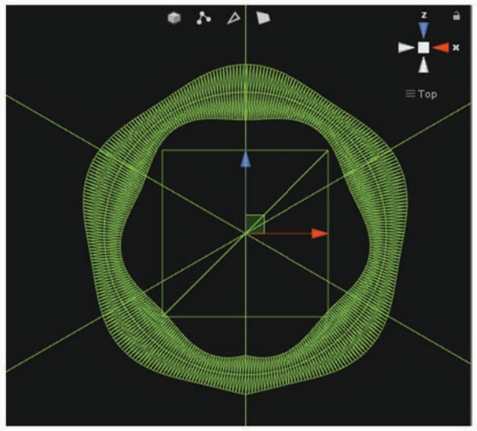
$c = \sqrt{3/2}$

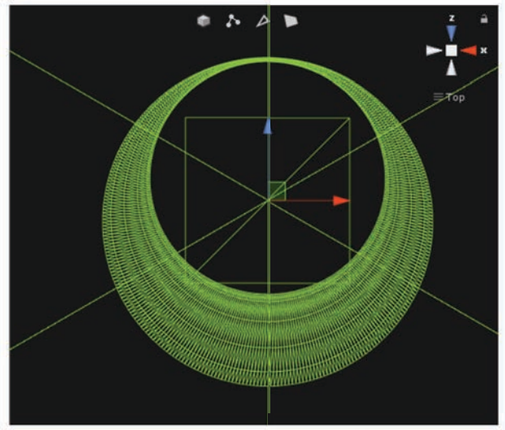
$\Theta = \arcsin(0.707/1.225)$

$\Theta = 35.26°$









Author Name
Terrence Howard
& David Johnson

Creation Date
Jan 20, 2019

Wave Conjugations
Flower of Life

Format: A3

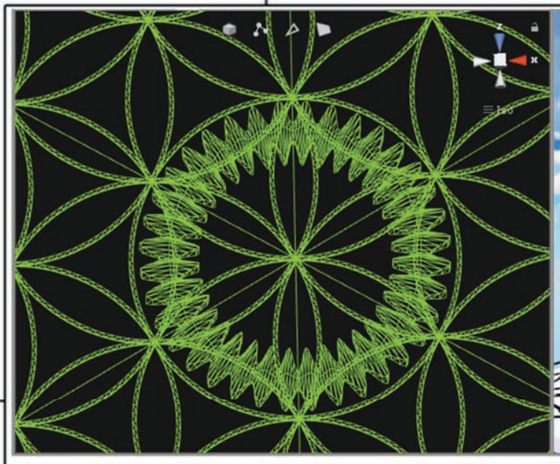
Description: Illustrations - Work Book

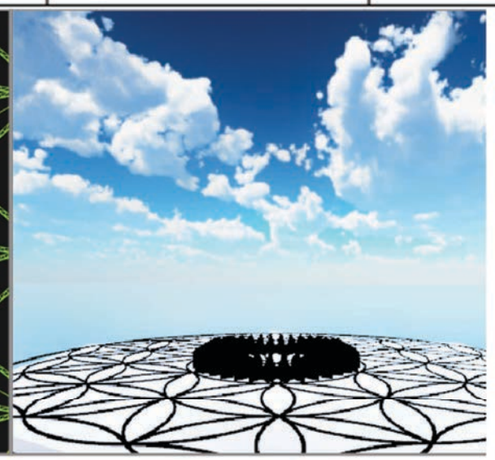
Draftsman: /dj

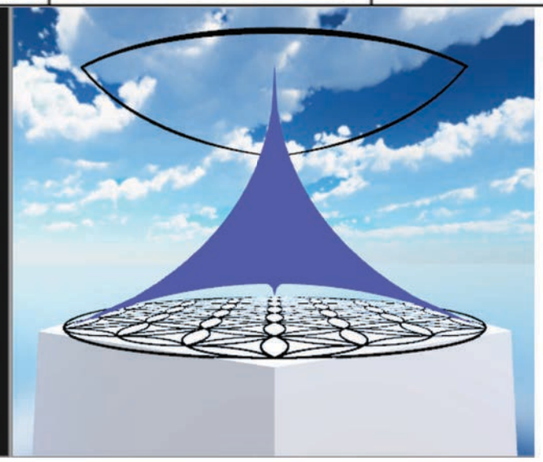
Produced by: Argos.Vu

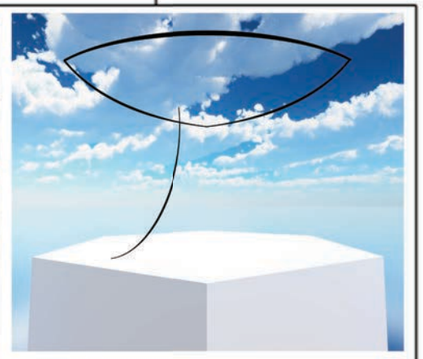
Version: 001

Page: 7

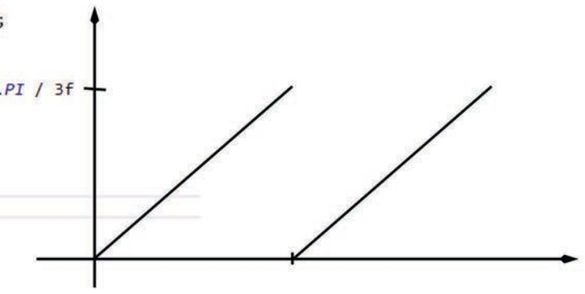


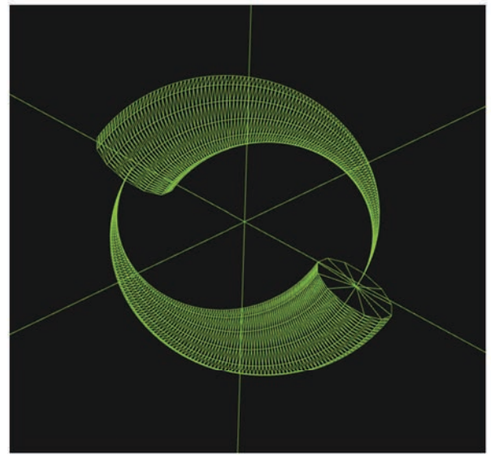






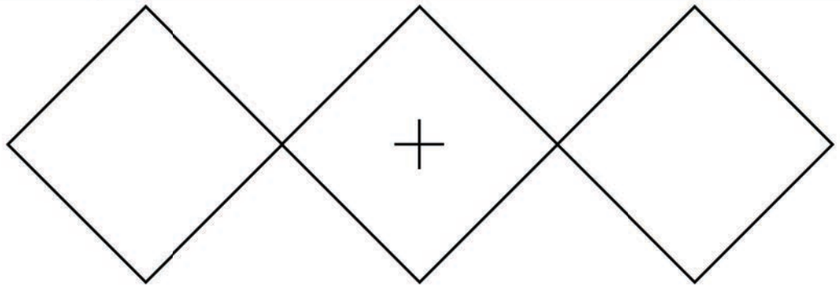
```
for (int j = 0; j < torusSegments; j++)
{
    vPointingR = PTUtils.PointOnCircle3(outerRadius, currentTorusAngle);
    nR = vPointingR.normalized;
    nX = Vector3.up;

    float currentAngle = segmentAngle / 2f;

    var upperRing = new List<Vector3>(segments);

    mW = currentTorusAngle % Mathf.PI / 3f;
    modW = 1 - (modWidth * Mathf.Cos(mW * 18));

    for (var i = 0; i < segments; i++)
    {
        var point = PTUtils.PointOnCircle3_XY(radius, currentAngle);

        var vLoc = origin + vPointingR + modW * (point.y * nR + point.x * nX);
        upperRing.Add(vLoc);
        currentAngle -= segmentAngle;
    }

    draft.Add(Band(lowerRing, upperRing));

    lowerRing = upperRing;
    currentTorusAngle += torusAngle;
}
draft.name = "Torus";
```

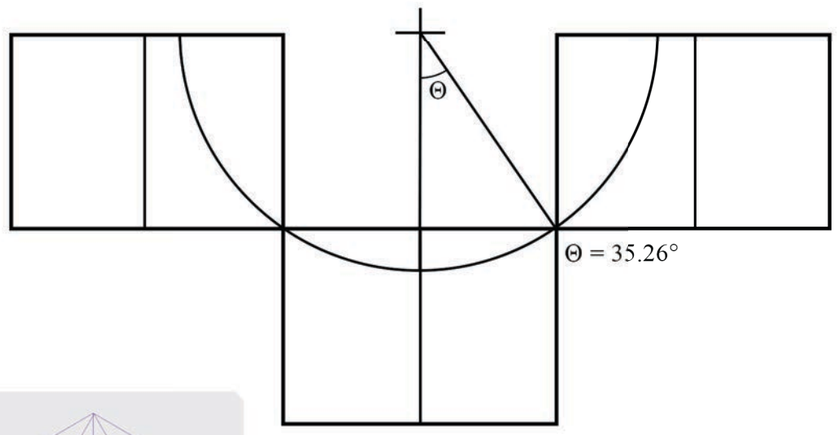


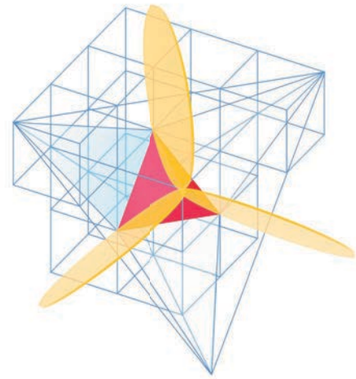


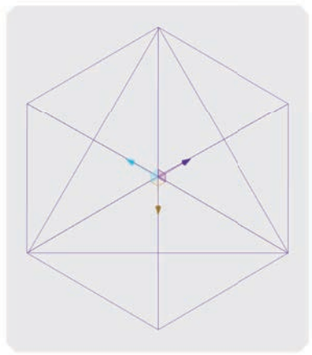
```
float mW = currentTorusAngle % (degrees_of_Arc / 360) * (Mathf.PI * 2);
modMult = (360 / degrees_of_Arc) * 3;
float modW = 1 - (modWidth * Mathf.Cos(mW * modMult));
```

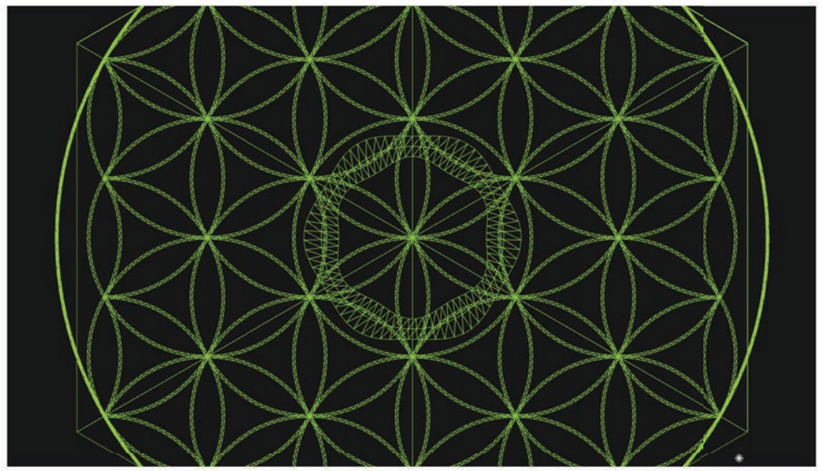
Author Name: Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date: Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 8

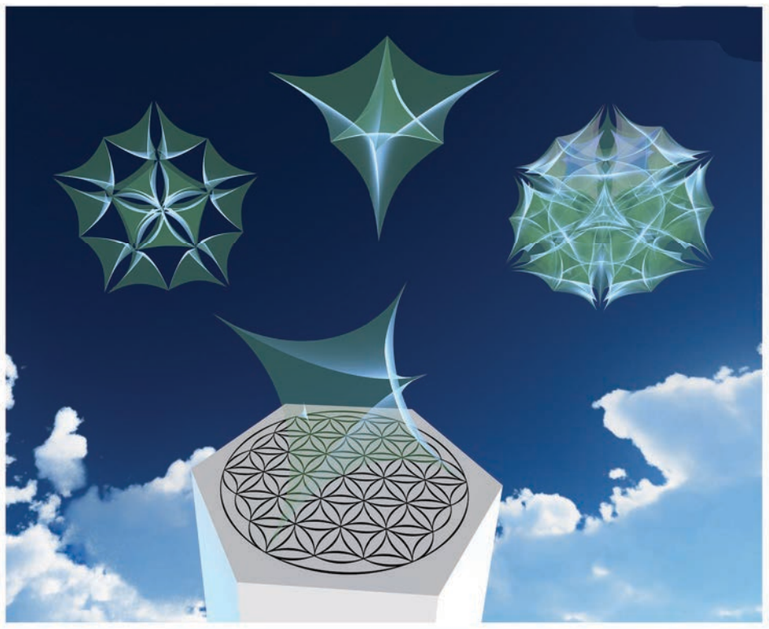


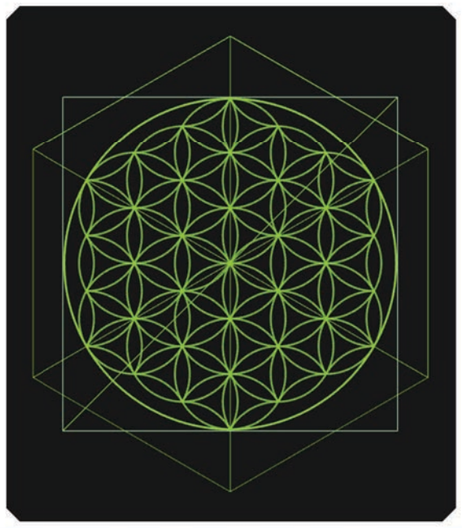






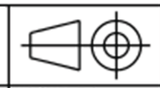






Author Name: Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date: Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 9
Copyright: Terrence Howard			

```
public class Huntien_Tetraterian_Face_Meshing : MonoBehaviour
{
    public GameObject meshPF;

    StreamWriter sWrite;

    public bool bDraw_Huntien_Face = true;
    public bool bDraw_TetraTerrien_Face = true;
    List<Vector3> vHuntian_Edge_List;
    VectorLine vLine_Huntian;
    List<Vector3> vTrine_Edge_List;

    ArgosMeshDraft amd_Fin;
    GameObject finGO;

    ArgosMeshDraft amd_Face_Huntien;
    GameObject faceGO_Huntien;

    ArgosMeshDraft amd_Face_TetraTerrien;
    GameObject faceGO_TetraTerrien;

    public int subDivisionDepth = 3;

    public Texture LineTex;
    List<float> lineWidths;

    [Range(0, 12f)]
    public float width;
    public Color color;

    [Range(0, 0.5f)]
    public float fade_ends;

    float fade_ends_last;

    [Range(-0.5f, 0.5f)]
    public float radRANGE;

    [Range(0.01f, 1f)]
    public float ttRAD;

    [Range(0.0f, 3f)]
    public float Adj_q_ANGLE;

    GameObject big_PIVOT;
    GameObject small_PIVOT;

    public enum LINEDENSITY
    {
        SEVEN = 7,
        THIRTEEN = 13,
        T25 = 25,
        T33 = 33,
        T65 = 65,
    }

    public LINEDENSITY ld = LINEDENSITY.T33;

    public int elDEE = 33;

    /// <image url="$(SolutionDir)\EMB\arc.png" scale="0.3"></image>
    /// <image url="$(SolutionDir)\EMB\Huntian.png" scale="0.5"></image>

    private Vector3 vA = new Vector3(0, 1, -1);
    private Vector3 vB = new Vector3(1, 0, -1);
    private Vector3 vC = new Vector3(1, 1, 0);

    private Vector3 vO = new Vector3(0, 1, 0);
    private Vector3 vP = new Vector3(0, 0, -1);
    private Vector3 vQ = new Vector3(1, 0, 0);

    private Vector3 vR = new Vector3(1, 1, -1);
    private Vector3 vS = new Vector3(0, 0, 0);
    private Vector3 vT = new Vector3(0.5f, 0.5f, -0.5f);

    private Vector3 vAR = new Vector3(1, 0, 0);
    private Vector3 vBR = new Vector3(0, 1, 0);
    private Vector3 vCR = new Vector3(0, 0, -1);

    private Vector3 vTetra_Tip;

    public GameObject cubeMark;
    public List<GameObject> goMarks = new List<GameObject>();

    private List<Vector3> vSharedEdge = new List<Vector3>();

    private void Start()
    {
        sWrite = new StreamWriter("Huntien_Vertices.txt");

        vTetra_Tip = vA + Mathf.Sqrt(3) * (vA - vT).normalized;

        vHuntian_Edge_List = new List<Vector3>() { new Vector3(0.000f, 0.000f, 0.000f), new Vector3(0.000f, 0.000f, 0.001f) };
        vLine_Huntian = new VectorLine("Circ_Line", vHuntian_Edge_List, width);
        vLine_Huntian.drawTransform = transform;
        vLine_Huntian.texture = LineTex;
        vLine_Huntian.SetColor(color);

        lineWidths = new List<float>();

        vTrine_Edge_List = new List<Vector3>();

        amd_Fin = new ArgosMeshDraft();
        finGO = Instantiate(meshPF, transform);
        finGO.name = "finGO";

        amd_Face_Huntien = new ArgosMeshDraft();
        faceGO_Huntien = Instantiate(meshPF, transform);
        faceGO_Huntien.name = "faceGO_Huntien";

        amd_Face_TetraTerrien = new ArgosMeshDraft();
        faceGO_TetraTerrien = Instantiate(meshPF, transform);
        faceGO_TetraTerrien.name = "faceGO_TetraTerrien";

        int objCount = (int)ld * 7;
        int runner = (int)ld;
        //bool bDone = false;


        big_PIVOT = new GameObject();
        big_PIVOT.gameObject.name = "BIG PIVOT";
        big_PIVOT.transform.position = new Vector3(-1, 2, -2);

        big_PIVOT.transform.rotation = Quaternion.LookRotation(-big_PIVOT.transform.position, new Vector3(0, 1, 1));

        small_PIVOT = new GameObject();
        small_PIVOT.gameObject.name = "small PIVOT";
        small_PIVOT.gameObject.transform.SetParent(big_PIVOT.transform);

        small_PIVOT.transform.localPosition = new Vector3(0, 0, 0);

        small_PIVOT.transform.localRotation = Quaternion.identity;

        GameObject go;
        for (int i = 0; i < 144; i++)
        {
            go = Instantiate(cubeMark);
            go.name = "vert_CubeMark";
            goMarks.Add(go);
        }
    }

    private float Width_Mirror(float t)
    {
        float mir = t;
        if (t > 0.5f)
        {
            mir = Mathf.Abs(1f - t);
        }
        float wid = width * Mathf.Clamp((mir / fade_ends), 0, 1);
        return wid;
    }

    private void Placement_Test()
    {
        float ang = 0;
        float dThet = 90 / (float)ld;
        Vector3 v;

        //AO
        Vector3 vAxis = vAR;
        Vector3 vOA = vO - vA;
        Quaternion q;

        List<Vector3> vList = new List<Vector3>();

        int cIdx = 0;

        for (int i = 0; i <= (int)ld; i++)
        {
            q = Quaternion.AngleAxis(ang, vAxis);
            goMarks[cIdx++].transform.localPosition = vA + q * vOA;
            vList.Add(vA + q * vOA);
            ang += dThet;
        }


        for (int j = 0; j <= 1; j++)
        {
            float mu = radRANGE * (float)(j + 1);
            ang = 0;
            Vector3 vVR;
            vAxis = vAR;
            for (int i = 0; i < (int)ld; i++)
            {
                q = Quaternion.AngleAxis(ang, vAxis);
                v = vA + q * vOA * (1 + mu);
                vVR = (v - vR).normalized * Mathf.Sqrt(2);

                goMarks[cIdx++].transform.localPosition = vR + vVR;
                vList.Add(vR + vVR);
                ang += dThet;
            }
        }
        amd_Fin.Clear();

        int el = (int)ld;

        amd_Fin.Add(MeshDraft.Triangle(vList[0], vList[el + 5], vList[1]));
        amd_Fin.Add(MeshDraft.Triangle(vList[1], vList[el + 5], vList[2]));
        amd_Fin.Add(MeshDraft.Triangle(vList[2], vList[el + 5], vList[3]));
        amd_Fin.Add(MeshDraft.Triangle(vList[3], vList[el + 5], vList[4]));

        for (int i = 4; i < el - 4; i++)
        {
            amd_Fin.Add(MeshDraft.Triangle(vList[i], vList[el + i + 1], vList[el + i + 2]));
            amd_Fin.Add(MeshDraft.Triangle(vList[i], vList[el + i + 2], vList[i + 1]));
        }

        amd_Fin.Add(MeshDraft.Triangle(vList[el - 1], vList[2 * el - 3], vList[el - 0]));
        amd_Fin.Add(MeshDraft.Triangle(vList[el - 2], vList[2 * el - 3], vList[el - 1]));
        amd_Fin.Add(MeshDraft.Triangle(vList[el - 3], vList[2 * el - 3], vList[el - 2]));
        amd_Fin.Add(MeshDraft.Triangle(vList[el - 4], vList[2 * el - 3], vList[el - 3]));

        finGO.GetComponent<MeshFilter>().mesh = amd_Fin.ToMeshInternal();
    }

    void Create_Shared_Edge()
    {
        //amd_Face_TetraTerrien.Clear();
        float ang = 0;
        float dThet = 90f / (float)elDEE;

        //AO
        Vector3 vAxis = vAR;
        Vector3 vOA = vO - vA;
        Quaternion q;
        List<Vector3> vList = new List<Vector3>();
        Vector3 v;
        int cIdx = 0;
        Vector3 vCircAxis_1 = new Vector3(1, 0, 1);
        Vector3 vCircPosition_1 = new Vector3(0.5f, 2f, -0.5f);
        Vector3 vCirc_1_Start = new Vector3(0, 1, 0);
        Vector3 vCircAxis_2 = new Vector3(-1, 1, 0);
        Vector3 vCircPosition_2 = new Vector3(0.5f, 0.5f, -2f);
        Vector3 vCirc_2_Start = new Vector3(0, 0, -1);

        Vector3 v1 = vCirc_1_Start - vCircPosition_1;
        Vector3 v2 = vCirc_2_Start - vCircPosition_2;
        float angle = 0;
        Quaternion qCirc1;
        Quaternion qCirc2;
        int start_CIdx = cIdx;

        for (int i = 0; i < 12; i++)
        {
            qCirc1 = Quaternion.AngleAxis(angle, vCircAxis_1);
            goMarks[cIdx].transform.localPosition = vCircPosition_1 + qCirc1 * v1;
            //goMarks[cIdx].transform.SetParent(big_PIVOT.transform);
            cIdx++;
            qCirc2 = Quaternion.AngleAxis(angle, vCircAxis_2);
            goMarks[cIdx].transform.localPosition = vCircPosition_2 + qCirc2 * v2;
            //goMarks[cIdx].transform.SetParent(big_PIVOT.transform);
            cIdx++;
            angle += 70.52f / 11f;
        }


        float dTheta;
        float theta;

        for (int j = 0; j < 12; j++)
        {
            Vector3 vT = goMarks[start_CIdx + 2 * j].transform.localPosition - vA;
            Vector3 vB = goMarks[start_CIdx + 2 * j + 1].transform.localPosition - vA;
            Vector3 vCross = Vector3.Cross(vT, vB);
            float ang_VT_VB = Vector3.Angle(vT, vB);
            Quaternion q1;
            dTheta = ang_VT_VB / (elDEE - 1);
            theta = 0;

            for (int i = 0; i < elDEE; i++)
            {
                q1 = Quaternion.AngleAxis(theta, vCross);
                vList.Add(vA + q1 * vT);
                //goMarks[cIdx].transform.position = vA + q1 * vT;
                cIdx++;
                theta += dTheta;
            }
        }

        amd_Face_TetraTerrien.Clear();

        int el = elDEE;

        for (int j = 0; j < 11; j++)
        {
            for (int i = 0; i < elDEE - 1; i++)
            {
                amd_Face_TetraTerrien.Add(MeshDraft.Triangle(vList[j * el + i], vList[j * el + el + i], vList[j * el + el + i + 1]));
                amd_Face_TetraTerrien.Add(MeshDraft.Triangle(vList[j * el + i], vList[j * el + el + i + 1], vList[j * el + i + 1]));
            }
        }
        faceGO_TetraTerrien.GetComponent<MeshFilter>().mesh = amd_Face_TetraTerrien.ToMeshInternal();
    }
}
```

Author Name Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 10
Copyright: Terrence Howard			

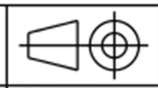
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ProceduralToolkit;

public class Flower_Of_Life : MonoBehaviour
{
    public enum FOL_RENDER_MODE
    {
        THREE_PART,
        FULL_TILE,
        INNER_TILE,
        INSTANCE_TILES,
    }

    //SPACE
    [Header("FLOWER OF LIFE - Terrence Howard Project")]
    [Space(12)]
    public FOL_RENDER_MODE FOL_Render_Mode;

    public GameObject meshPF_Rings;
    public GameObject meshPF_Arcs;
    public GameObject meshPF_EnCircle;

    public ArgosMeshDraft amd_RINGs;
    public ArgosMeshDraft amd_ARCs;
    public ArgosMeshDraft amd_EnCircle;

    [Header("Direct Instancing")]
    [Space(12)]

    public bool bInstance = false;

    [Range(0, 2f)]
    public float instancing_Rate = 0.1f;
    public GameObject Flower_Instance;
    public int num_to_Instance;
    public List<GameObject> flower_List = new List<GameObject>();

    [Header("Generate")]
    [Space(12)]
    [Range(0, 0.5f)]
    public float tubeRadius;
    [Range(0, 12.0f)]
    public int segments;
    [Range(0, 2.0f)]
    public float outerRadius;
    [Range(0, 360.0f)]
    public int torusSegments;

    [Range(0, 1.0f)]
    public float modWidth;

    public  bool bRun_Animation = false;
    private bool bRun_Animation_Last = false;
    public  float time_per_Circle = 0.4f;
    public float time_draw_EnCircle = 0.4f;
    private bool bDrawEncircle = false;

    private int curr_Ring_Segments = 0;
    private int curr_Arc_Segments = 0;
    private int curr_EnCircle_Segments = 0;

    private float ringTime = 0;
    private float arcTime = 0;
    private float enCircleTime = 0;

    private int circ_Counter = 0;
    private int arc_Counter = 0;

    private Vector3[] circ_Locs = new Vector3[19];
    private Vector3[] arc_Locs = new Vector3[36];

    private float[] arcStarts = new float[] { 120f, 60f, 60f, 120f, 120f, 120f };
    private float[] arcLengths = new float[] { 120f, 180f, 180f, 60f, 60f, 60f };

    void Start ()
    {
        amd_RINGs = new ArgosMeshDraft();
        amd_ARCs  = new ArgosMeshDraft();
        amd_EnCircle = new ArgosMeshDraft();

        circ_Locs[0] = Vector3.zero;
        Vector3 vBaseR = Vector3.forward * outerRadius;

        arc_Counter = -1;
        circ_Counter = -1;

        //RINGS
        float ang = 0;
        Quaternion q = Quaternion.identity;
        float dAng = 360 / 6f;
        int curr_Idx = 1;

        for(int i = 0; i<6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            circ_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        ang = 0;
        vBaseR *= 2;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            circ_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        ang = -360/ 12;
        vBaseR = 2*Vector3.forward * outerRadius;
        vBaseR *= Mathf.Cos(Mathf.PI/6);
        q = Quaternion.AngleAxis(ang, Vector3.up);
        for (int i = 0; i < 6; i++)
        {
            circ_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            q = Quaternion.AngleAxis(ang, Vector3.up);
            curr_Idx++;
        }

        //ARCs
        ang = 0;
        q = Quaternion.identity;
        curr_Idx = 0,

        vBaseR = 3 * Vector3.forward * outerRadius;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            arc_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        vBaseR = 2 * Vector3.forward * outerRadius + Quaternion.Euler(new Vector3(0,60,0))*Vector3.forward*outerRadius;
        ang = 0;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            arc_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        vBaseR = Vector3.forward * outerRadius + Quaternion.Euler(new Vector3(0, 60, 0)) * Vector3.forward * outerRadius * 2;
        ang = 0;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            arc_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        vBaseR = 3*Vector3.forward * outerRadius + Quaternion.Euler(new Vector3(0, 60, 0)) * Vector3.forward * outerRadius;
        ang = 0;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            arc_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        vBaseR = 2 * Vector3.forward * outerRadius + Quaternion.Euler(new Vector3(0, 60, 0)) * Vector3.forward * outerRadius * 2;
        ang = 0;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            arc_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }

        vBaseR = Vector3.forward * outerRadius + Quaternion.Euler(new Vector3(0, 60, 0)) * Vector3.forward * outerRadius * 3;
        ang = 0;
        for (int i = 0; i < 6; i++)
        {
            q = Quaternion.AngleAxis(ang, Vector3.up);
            arc_Locs[curr_Idx] = q * vBaseR;
            ang -= dAng;
            curr_Idx++;
        }
    }

    private bool bInstance_Last = false;
    private float instanceTimer = 0;
    private void Instance_Flower_Tiles()
    {
        if(bInstance)
        {
            if(bInstance_Last != bInstance)
            {
                flower_List.Clear();
                instanceTimer = 0;
            }
        }
        bInstance_Last = bInstance_Last;
    }

    private void Three_Part()
    {
        int arc_Segments = 0;

        float arcDuration;
        if (bRun_Animation)
        {
            if (!bRun_Animation_Last)
            {
                curr_Ring_Segments = 0;
                ringTime = 0;
                arcTime = 0;
                enCircleTime = 0;
                circ_Counter = 0;
                arc_Counter = 0;

                bDrawEncircle = false;
            }
            ringTime += Time.deltaTime;
            curr_Ring_Segments = (int)((ringTime / time_per_Circle) * (float)torusSegments);

            if (ringTime > time_per_Circle)
            {
                circ_Counter++;
                curr_Ring_Segments = 0;
                ringTime = 0;
            }
            if (circ_Counter > circ_Locs.Length)
            {
                arcTime += Time.deltaTime;
                arcDuration = (arcLengths[arc_Counter / 6] / 360f) * time_per_Circle;

                arc_Segments = (int)(torusSegments * arcLengths[arc_Counter / 6] / 360f);

                curr_Arc_Segments = (int)((arcTime / arcDuration) * (float)arc_Segments);

                if (arcTime > arcDuration)
                {
                    arc_Counter++;
                    curr_Arc_Segments = 0;
                    arcTime = 0;
                }
                if (arc_Counter > arc_Locs.Length - 1)
                {
                    bDrawEncircle = true;
                    bRun_Animation = false;
                }
            }
            bRun_Animation_Last = bRun_Animation;
            //Torus(float tubeRadius, int segments, float outerRadius, int torusSegments)

            for (int i = 0; i < circ_Locs.Length; i++)
            {
                if (i < circ_Counter)
                {
                    amd_RINGs.Add(MeshDraft.Torus(circ_Locs[i], tubeRadius, segments, outerRadius, torusSegments, torusSegments, modWidth));
                }
                else if (i == circ_Counter)
                {
                    amd_RINGs.Add(MeshDraft.Torus(circ_Locs[i], tubeRadius, segments, outerRadius, torusSegments, curr_Ring_Segments, modWidth));
                }
            }
            meshPF_Rings.GetComponent<MeshFilter>().mesh = amd_RINGs.ToMeshInternal();

            for (int i = 0; i < arc_Locs.Length; i++)
            {
                if (i < arc_Counter)
                {
                    arc_Segments = (int)(torusSegments * arcLengths[i / 6] / 360f);
                    amd_ARCs.Add(MeshDraft.Torus_Arc(arcStarts[i / 6] + (i % 6) * 60, arc_Locs[i], tubeRadius, segments, outerRadius, torusSegments, arc_Segments, modWidth));
                }
                else if (i == arc_Counter)
                {
                    amd_ARCs.Add(MeshDraft.Torus_Arc(arcStarts[i / 6] + (i % 6) * 60, arc_Locs[i], tubeRadius, segments, outerRadius, torusSegments, curr_Arc_Segments, modWidth));
                }
            }
            meshPF_Arcs.GetComponent<MeshFilter>().mesh = amd_ARCs.ToMeshInternal();

            if (bDrawEncircle)
            {
                enCircleTime += Time.deltaTime;
                curr_EnCircle_Segments = (int)((enCircleTime / time_draw_EnCircle) * (float)torusSegments);

                amd_EnCircle.Add(MeshDraft.Torus(Vector3.zero, tubeRadius, segments, 3 * outerRadius, torusSegments, curr_EnCircle_Segments, modWidth));
            }
            meshPF_EnCircle.GetComponent<MeshFilter>().mesh = amd_EnCircle.ToMeshInternal();
            //meshPF.GetComponent<MeshFilter>().mesh.RecalculateNormals(60);
        }
    }
```

Author Name	Terrence Howard & David Johnson
Creation Date	Jan 20, 2019
Format:	A3
Description:	Illustrations - Work Book
Draftsman:	/dj
Produced by:	Argos.Vu
Version:	001
Page:	11

Wave Conjugations Flower of Life

```
private void Full_Tile()
{
    //Outer Ring
    amd_RINGs.Add(MeshDraft.Torus(Vector3.zero, tubeRadius, segments, outerRadius, torusSegments, torusSegments, modWidth));

    //Large Arcs
    int ringSegments = (int)(torusSegments * 120 / 360f);
    amd_RINGs.Add(MeshDraft.Torus_Arc(120, circ_Locs[1], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(180, circ_Locs[2], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(240, circ_Locs[3], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(300, circ_Locs[4], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(0,   circ_Locs[5], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(60,  circ_Locs[6], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    //Scallops
    ringSegments = (int)(torusSegments * 60 / 360f);
    amd_RINGs.Add(MeshDraft.Torus_Arc(180, circ_Locs[13], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(240, circ_Locs[14], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(300, circ_Locs[15], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(0, circ_Locs[16], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(60, circ_Locs[17], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(120, circ_Locs[18], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    meshPF_Rings.GetComponent<MeshFilter>().mesh = amd_RINGs.ToMeshInternal();
}

private void Inner_Tile()
{

    //Large Arcs
    int ringSegments = (int)(torusSegments * 120 / 360f);
    amd_RINGs.Add(MeshDraft.Torus_Arc(120, circ_Locs[1], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(180, circ_Locs[2], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(240, circ_Locs[3], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(300, circ_Locs[4], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(0, circ_Locs[5], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(60, circ_Locs[6], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    //Scallops
    ringSegments = (int)(torusSegments * 60 / 360f);
    amd_RINGs.Add(MeshDraft.Torus_Arc(180, circ_Locs[13], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(240, circ_Locs[14], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(300, circ_Locs[15], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(0, circ_Locs[16], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    amd_RINGs.Add(MeshDraft.Torus_Arc(60, circ_Locs[17], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));
    amd_RINGs.Add(MeshDraft.Torus_Arc(120, circ_Locs[18], tubeRadius, segments, outerRadius, torusSegments, ringSegments, modWidth));

    meshPF_Rings.GetComponent<MeshFilter>().mesh = amd_RINGs.ToMeshInternal();
}

void Update ()
{
    amd_RINGs.Clear();
    amd_ARCs.Clear();
    amd_EnCircle.Clear();

    if(FOL_Render_Mode == FOL_RENDER_MODE.THREE_PART)
    {
        Three_Part();
    }
    else if(FOL_Render_Mode == FOL_RENDER_MODE.FULL_TILE)
    {
        Full_Tile();
    }
    else if (FOL_Render_Mode == FOL_RENDER_MODE.INNER_TILE)
    {
        Inner_Tile();
    }
    else if(FOL_Render_Mode == FOL_RENDER_MODE.INSTANCE_TILES)
    {
        Instance_Flower_Tiles();
    }
}
```

```
public static MeshDraft TH_Arc(float startAng, Vector3 origin, float radius, int segments,
    float outerRadius, int torusSegments, float degrees_of_Arc, float modWidth, float modMult)
{
    var draft = new MeshDraft();
    float torusAngle = (degrees_of_Arc/360)*(Mathf.PI * 2 / torusSegments);
    float segmentAngle = Mathf.PI * 2 / segments;
    Vector3 vPointingR;
    //float orbitAngle = Mathf.PI * 2 / torusSegments;
    float currentTorusAngle = startAng * Mathf.PI * 2 / 360f;

    //First ring
    vPointingR = PTUtils.PointOnCircle3(outerRadius, currentTorusAngle);
    var lowerRing = new List<Vector3>(segments);

    var nR = vPointingR.normalized;
    var nX = Vector3.up;

    float mW = currentTorusAngle % (degrees_of_Arc / 360) * (Mathf.PI * 2);
    modMult = (360 / degrees_of_Arc) * 3;
    float modW = 1 - (modWidth * Mathf.Cos(mW * modMult));

    float cA = segmentAngle / 2f;
    for (var i = 0; i < segments; i++)
    {
        var point = PTUtils.PointOnCircle3_XY(radius, cA);
        var vLoc = origin + vPointingR + modW * (point.y * nR + point.x * nX);
        lowerRing.Add(vLoc);
        cA -= segmentAngle;
    }

    currentTorusAngle += torusAngle;

    Quaternion q;

    for (int j = 0; j < torusSegments; j++)
    {
        vPointingR = PTUtils.PointOnCircle3(outerRadius, currentTorusAngle);
        nR = vPointingR.normalized;
        nX = Vector3.up;

        float currentAngle = segmentAngle / 2f;

        var upperRing = new List<Vector3>(segments);

        mW = currentTorusAngle % Mathf.PI / 3f;
        modW = 1 - (modWidth * Mathf.Cos(mW * modMult));

        for (var i = 0; i < segments; i++)
        {
            var point = PTUtils.PointOnCircle3_XY(radius, currentAngle);

            var vLoc = origin + vPointingR + modW * (point.y * nR + point.x * nX);
            upperRing.Add(vLoc);
            currentAngle -= segmentAngle;
        }

        draft.Add(Band(lowerRing, upperRing));

        lowerRing = upperRing;
        currentTorusAngle += torusAngle;
    }
    draft.name = "Torus";
    return draft;
}
```

Author Name Terrence Howard & David Johnson		# Wave Conjugations Flower of Life	
Creation Date Jan 20, 2019			
Format: A3		Description: Illustrations - Work Book	
Draftsman: /dj	Produced by: Argos.Vu	Version: 001	Page: 12
Copyright: Terrence Howard			