Notes on the Pup Tent

Richard Evan Schwartz *

August 22, 2025

Abstract

These notes have material supplementary to my paper *Vertex-Minimal Paper Tori*.

1 Introduction

These notes give material supplementary to my paper

[**S**] R. E. Schwartz, *Vertex-Minimal Paper Tori*
arXiv 2507.14998

In [**S**] I presented my discovery of an 8-vertex paper torus. This is a piecewise affine isometric embedding of a flat torus into $\boldsymbol{R}^3$ such that the result has 8 vertices. Here are the coordinates up to 32-digit precision. I call it the *pup tent*. There is a 6-parameter family of inequivalent pup tents. Here is the main one from the paper.

$$\begin{array}{lll} +0.755 & +0.650 & z_0 \\ -0.455 & +0.345 & z_1 \\ -0.170 & +1.140 & z_2 \\ +0.455 & -0.345 & z_1 \\ -0.755 & -0.650 & z_0 \\ -0.090 & +0.665 & 0 \\ +0.170 & -1.140 & z_2 \\ +0.090 & -0.665 & 0 \end{array} \qquad \begin{array}{l} z_0 = 0.9805\ 0571\ 5859\ 7793\ 5561\ 6538\ 2008\ 5693 \\ z_1 = 0.9902\ 8162\ 4334\ 3054\ 2934\ 3176\ 1585\ 8328 \\ z_2 = 0.9765\ 3883\ 4703\ 1231\ 7624\ 1842\ 4567\ 2434 \end{array} \tag{1}$$

The notes are something of a grab bag. Here is a guide to their contents.

* Supported by N.S.F. Research Grant DMS-2505281

- §2 extends the work in [**S**] to give a proof of the Hull Theorem from the 1991 paper of Bokowski and Eggert. This result says that a 7-vertex embedded polyhedral torus cannot have all its vertices on its convex hull boundary.

- §3 has all the Java code I used for my proof of the non-existence result in the 7-vertex case.

- §4 gives a folding pattern for the *pup tent*, the 8-vertex paper torus, as well as instructions on how to reproduce the folding diagram yourself – up to a point.

- §5 gives me original proof that the pup tent given above is 10^{-30} flat. This proof is done with purely integer calculations. In the paper I just feed the example into Mathematica and rely on its ability to compute elementary functions to high precisuon.

- §6 gives coordinates for other pup tents. At the moment there is just one.

2 The Hull Theorem

2.1 Context

The Hull Theorem says that an embedded 7-vertex convex torus cannot have all 7 vertices on its convex hull boundary. In [**S**] I prove a somewhat weaker result that is sufficient to prove that there are no 7-vertex paper tori.

Let me state my result in a different way. Given a 7-vertex embedded polyhedral torus, a *flower* is a union of all the triangles incident to a given vertex. Thus there are 7 flowers, each having 6 triangles. What we actually prove in [**S**] is that any 7-vertex polyhedral torus with all 7-vertices on its convex hull boundary must have a flower entirely in the boundary as well. I call this the *Hull Lemma*.

As we remarked in [**S**] this situation makes it easy to prove the Hull Theorem. We just have to rule out a very specific kind of example.

Figure 2.1 shows one of the the 6 *internal edge patterns* associated to such a polyheral torus.

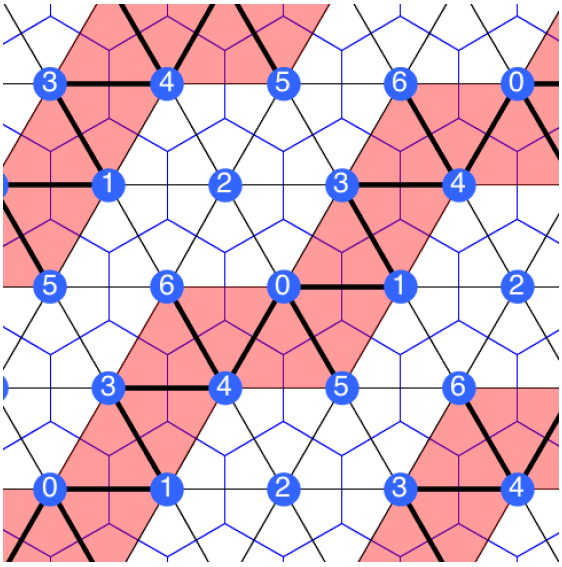


Figure 2.1: The one remaining pattern

What we are showing here is (part of) the univeral cover of the 7-vertex triangulation of the torus. Of the 21 edges in the triangulation, 6 of them do not lie in the convex hull boundary. (The convex hull boundary has 10 faces, and 15 edges, and 7 vertices.) These special 6 edges are drawn thickly in Figure 2.1. The white triangles are on the convex hull boundary and the pink ones are not.

2.2 Projective Space

Let me briefly recall projective geometry *Projective space* $\boldsymbol{P}^3$ is the space of lines through the origin in $\boldsymbol{R}^4$. Points in $\boldsymbol{P}^3$ are usually denoted by homogeneous coordinates $[a : b : c : d]$. This point names the line through (a, b, c, d). Euclidean space $\boldsymbol{R}^3$ sits inside $\boldsymbol{P}^3$ as the *affine patch*. These are points of the form $[a : b : c : 1]$.

An invertible 4×4 linear transformation naturally acts on $\boldsymbol{P}^3$ as a diffeomorphism. These maps are called projective transformations. If we have a polyhedral torus contained in the the affine patch, then its convex hull will also be contained in the affine patch. We can apply a projective transformation that keeps the convex hull in the affine patch. The image of the torus under this map will also be an embedding. In this way – and in a slightly extended way – we will move our example around by projective transformations to get a clearer picture of it.

2.3 The Hull Theorem

Let Ω be an example which supposedly corresponds to Figure 2.1. We think of Ω as a subset of projective space $\boldsymbol{P}^3$. We can apply a projective transformation so that vertex (2) moves to the point $[0 : 0 : 1 : 0]$ at infinity in $\boldsymbol{P}^3$. This point is "infinitely far away" along the Z-axis. We also can arrange that the (now) rays $(2j)$ start at (j) and move downward (rather than upwards) along the Z-axis, limiting on (2). We do all this by a projective transformation that maps $H - \{(2)\}$ into the affine patch $\boldsymbol{R}^3$.

Now, $H - \{(2)\}$ is still a convex subset in $\boldsymbol{R}^3$. It is like a hexagonal prism that has been truncated at one end. Here is the crucial observation: since H is convex, the projection of the hexagon (603541) into the XY-plane is a convex hexagon. We can further normalize so that the projections of (15) and (36) into the XY-plane are parallel line segments. We do this by mapping the line $[215] \cap [236]$ to a line at infinity which contains (2). Here $[abc]$ is the plane containing (a) and (b) and (c).

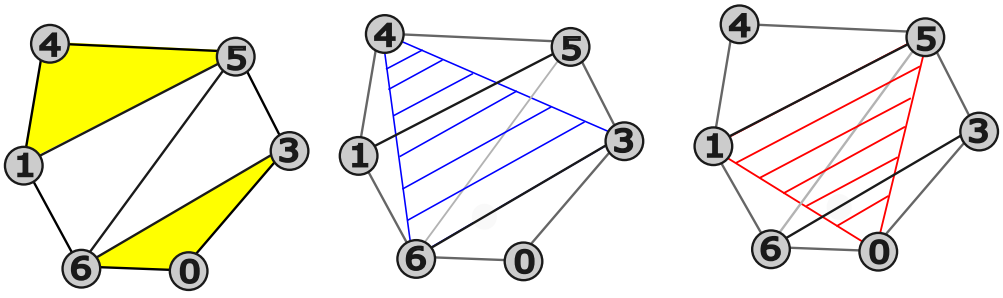


Figure 2.2: Projection of part of ∂H into the XY-plane

The only triangles of ∂H not in Ω are (036) and (145). By convexity, these two triangles are bent downward. What we mean is that the plane containing (036) has the rest of H beneath it. The same goes for the plane containing (145).

The blue triangle in Figure 2.2 is the internal triangle (346). We can foliate this triangle by parallel line segments as shown in the figure. These segments are parallel in space and they project to parallel segments in $\boldsymbol{R}^2$. The red triangle in Figure 2.2 is the internal triangle (015). We make all the same constructions for this triangle.

Say that a *special planes* $\boldsymbol{R}^3$ whose projections to $\boldsymbol{R}^2$ is a line parallel to the projections of our red and blue foliations. One special plane contains (15). In this plane, the red foliation is above the blue foliation. Another special plane contains (36). In this plane, the blue foliation is above the red foliation. So, by the intermediate value theorem, there is a special plane for which these two foliations coincide. But then (015) and (346) intersect. This contradicts the fact that Ω is embedded.

3 Java Code

The code for the proof of the Hull Lemma is distributed in two files. The first file helps manipulate lists. The second file has the actual tests.

3.1 The ListHelp Class

```
import java.util.Arrays;

public class ListHelp {

    /**prints out an integer list*/
    public static void printout(int[] list) {
    if(list==null) return;
    for(int i=0;i<list.length;++i) System.out.print(list[i]+" ");
    System.out.println("");
    }

    /**checks if two lists match up to permutation*/
    public static boolean match(int[] a,int[] b) {
    if(a.length!=b.length) return false;
    int[] aa=Arrays.copyOf(a,a.length);
    int[] bb=Arrays.copyOf(b,b.length);
    Arrays.sort(aa);
    Arrays.sort(bb);
    for(int i=0;i<a.length;++i) {
        if(aa[i]!=bb[i]) return false;
    }
    return true;
    }

    /**checks if element a is amongst the
    first k elements of list b*/
    public static boolean onList(int a,int[] b,int k) {
    for(int i=0;i<k;++i) {
        if(a==b[i]) return true;
    }
    return false;
    }
```

```
/**take an integer list, sorts it, and removes redundancies*/
public static int[] irredundantSortedList(int[] data) {
Arrays.sort(data);
int n=data.length;  int[] temp = new int[n];
int count = 0;
for (int i = 0; i < n; ++i) {
if ((i == 0) ||(data[i] != data[i - 1])) {
        temp[count] = data[i];
        ++count;
      }}
return Arrays.copyOf(temp, count);
}

/**Gives all the 6 element subsets of {0,...,20} having
  0 as the first element*/

public static int[] subsetGenerator(int index) {
int[] subset = {0,0,0,0,0,0};
int x = 1;

for (int i=1; i<6 ;++i) {
    while(choose(20-x,5-i)<=index) {
        index = index - choose(20 - x, 5 - i);
        ++x;
    }
    subset[i] = x;
    ++x;
}
return subset;
}

/**This returns n choose k.*/
public static int choose(int n,int k) {
  int x=1;int y=1;
for(int i=1;i<=k;++i) {
    x=x*(n-i+1);
    y=y*i;
}
return x/y;
}
```

```
/**Gets the kth dihedral permutation of list a*/
public static int[] perDihedral(int[] a,int k) {
int n=a.length;
if(k<n) return cycle(a,k);
return reverse(cycle(a,k));
}

/**reverses list a*/
public static int[] reverse(int[] a) {
int n=a.length;
int[] b=new int[n];
for(int i=0;i<n;++i) b[i]=a[n-i-1];
return b;
}

/**cycles list a by k clicks*/
public static int[] cycle(int[] a,int k) {
int n=a.length;
int[] b=new int[n];
for(int i=0;i<n;++i) b[i]=a[(i+k)%n];
return b;
}
```

3.2 The LinkAnalyzer Class

```
import java.util.Arrays;

public class LinkAnalyzer {

/**This class performs all the tests for the proof of the
   Hull Theorem in our paper. The file works with ListHelp.java
   to manipulate lists*/

/**Returns kth edge of the complete graph K7. This is also the
   1-skeleton of the 7-vertex triangulation of the torus*/
public static int[] edge(int k) {
  int[][] f={{0,1},{0,2},{0,3},{0,4},{0,5},{0,6},
              {1,2},{1,3},{1,4},{1,5},{1,6},
              {2,3},{2,4},{2,5},{2,6},
              {3,4},{3,5},{3,6},
              {4,5},{4,6},{5,6}};
    return f[k];
}

/**Returns kth face in the 7-vertex triangulation of the torus.*/
public static int[] face(int k) {
    int[][] f={{0,1,3},{0,5,1},{0,3,2},{0,2,6},{0,4,5},{0,6,4},
              {1,2,4},{1,4,3},{1,6,2},{1,5,6},{2,3,5},{2,5,4},
              {3,4,6},{3,6,5}};
    return f[k];
}

/**This gets the links of each vertex in the torus*/

    public static int[] torusLink(int k) {
      int[][] L={{1,3,2,6,4,5},{0,5,6,2,4,3},{0,3,5,4,1,6},
      {0,1,4,6,5,2},{0,6,3,1,2,5},{0,4,2,3,6,1},{0,2,1,5,3,4}};
    return L[k];
      }
```

```
/**This gets the kth choice of 6 element subset of the edges and
   then returns the corresponding edges.*/
public static int[][] internalEdges(int k) {
    int[] t=ListHelp.subsetGenerator(k);
    int[][] list=new int[6][2];
    for(int i=0;i<6;++i) {
        list[i]=edge(t[i]);
    }
    return list;
}

/**This gets the triangles incident to the edge
list from the previous routine.*/
public static int[][] internalFaces(int k) {
    int[] t=ListHelp.subsetGenerator(k);
    int[] list1=new int[12];
    int count=0;
    for(int i=0;i<6;++i) {
        int[] ee=edge(t[i]);
        for(int j=0;j<14;++j) {
            if(incident(ee,face(j))==true) {
                list1[count]=j;
                ++count;
            }
        }
    }
    list1=ListHelp.irredundantSortedList(list1);
    int[][] list2=new int[list1.length][3];
    for(int i=0;i<list1.length;++i) list2[i]=face(list1[i]);
    return list2;}

/**Returns true if edge e is incident to face f.*/
public static boolean incident(int[] e,int[] f) {
    if(ListHelp.onList(e[0],f,3)==false) return false;
    if(ListHelp.onList(e[1],f,3)==false) return false;
    return true;}
```

```
/**This routine picks a vertex k and returns all the
external edges that are incident to it.  We are
careful to maintain the correct cyclic order*/

      public static int[] convexLink(int[][] e,int k) {
      int[] L=torusLink(k);
    int[] list=new int[6];
    int count=0;
    for(int i=0;i<6;++i) {
        int[] ee={k,L[i]};
        boolean test=false;
        for(int j=0;j<6;++j) {
        if(ListHelp.match(ee,e[j])==true) {
          test=true;
          break;
        }
        }
        if(test==false) {
           list[count]=L[i];
           ++count;
        }
    }
    return Arrays.copyOf(list,count);
    }

/**Make sure that consecutive elements are not internal edges.*/
public static boolean onlyAllowedConnections(int[][] e,int[] cycle) {
    for(int i=0;i<cycle.length;++i) {
        int ii=(i+1)%cycle.length;
        int[] L={cycle[i],cycle[ii]};
        for(int j=0;j<6;++j) {
            if(ListHelp.match(L,e[j])==true) return false;
        }
    }
    return true;}
```

```
/**This returns the link if it is viable
and otherwise returns null*/

    public static int[] getViableCycle(int k,int i) {
    int[][] edge=internalEdges(k);
    int count=0;
    int[] link=LinkAnalyzer.convexLink(edge,i);
    if(link.length<3) return null;
    if(onlyAllowedConnections(edge,link)==false) return null;
    return link;}

    /**Tests all the cycles associated to the kth internal
    edge pattern.  Returns true if they are all viable. */

    public static boolean mainTest(int[] filter,int k) {
    for(int i=0;i<7;++i) {
        int[] cyc=getViableCycle(k,i);
        if((filter[i]==1)&&(cyc==null)) return false;
    }
    return true;}

/**This final  test. This is what we run.*/
    public void bigTest() {
    int count=0;
    int[] f={1,1,1,1,1,1,1};
    for(int i=0;i<15504;++i) {
          if(mainTest(f,i)==true)  ++count;
    }
    System.out.println("count (should be 0) "+count);}

      }
```

4 Folding Pattern

4.1 The Triangulation

Here is the triangulation on which the pup tent is based on the following triangulation. The dark blue polygon gives a fundamental domain. The 6 blue triangles are the ones on the convex hull boundary.

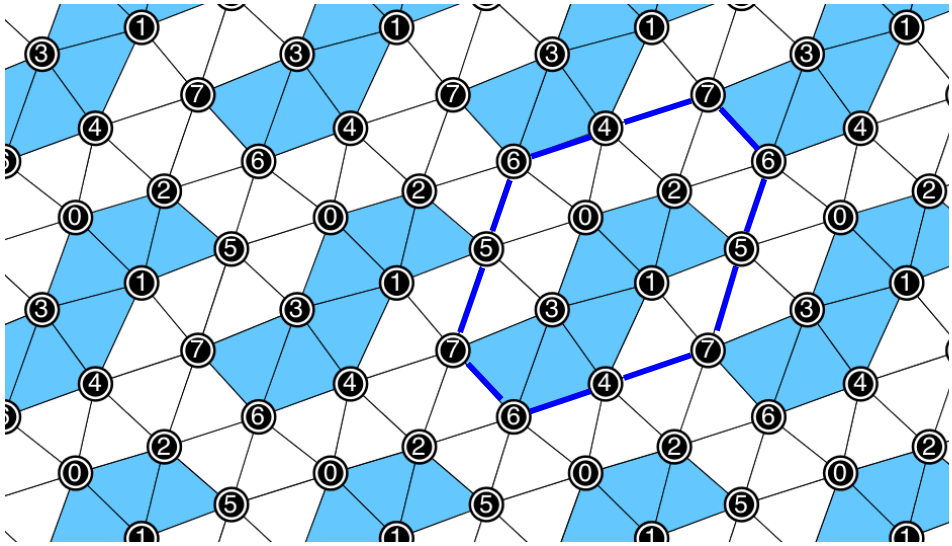


Figure 3.1: The universal cover of the best 8 vertex triangulation

Here is the list of triangles, all oriented counterclockwise.

$$\begin{array}{cccc} \{0,1,2\} & \{0,3,1\} & \{0,2,4\} & \{0,5,3\} \\ \{0,4,6\} & \{0,6,5\} & \{1,5,2\} & \{1,3,4\} \\ \{1,4,7\} & \{1,7,5\} & \{2,7,4\} & \{2,5,6\} \\ \{2,6,7\} & \{3,6,4\} & \{3,5,7\} & \{3,7,6\} \end{array}$$

Figure 3.2 shows a plot of the intrinsic structure of the pup tent, the flat torus on which it is based.

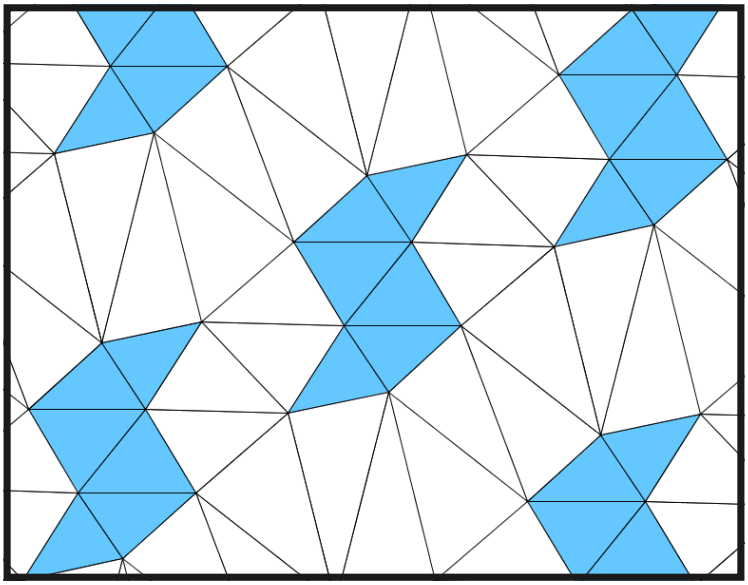


Figure 3.2: The intrinsic structure.

You can generate your own version of Figure 3.2 using the following 8 vertices and the two generators λ_1, λ_2 of the deck group.

Vertices

$$
\begin{array}{cl}
0 & \texttt{+0.3364 3064 6031, +4.5292 0435 4142} \\
1 & \texttt{-1.2385 1945 0198, +2.7553 7307 2854} \\
2 & \texttt{-1.7092 8291 8080, +3.4566 7231 1107} \\
3 & \texttt{+0.8705 9028 3280, +3.6376 9212 7462} \\
4 & \texttt{-0.7043 5981 2948, +1.8638 6084 6174} \\
5 & \texttt{-0.6445 2775 0478, +3.6845 9670 3719} \\
6 & \texttt{+1.3413 5375 1162, +2.9363 9288 9208} \\
7 & \texttt{+0.2765 9858 3560, +2.7084 6849 6596}
\end{array}
\tag{2}
$$

Lattice generators

$$
\begin{array}{cl}
\lambda_1 & \texttt{+2.8228 3653 6730, +1.7738 3128 1287} \\
\lambda_2 & \texttt{-2.4864 0589 0699, +2.7553 7307 2854}
\end{array}
$$

Next, here is a folding pattern based in Figure 3.2. I made this after consulting with Noah Montgomery and Alba Malaga. Expert folders like them are able to put this thing together. I have to admit that I can't actually do it myself.

fold guide

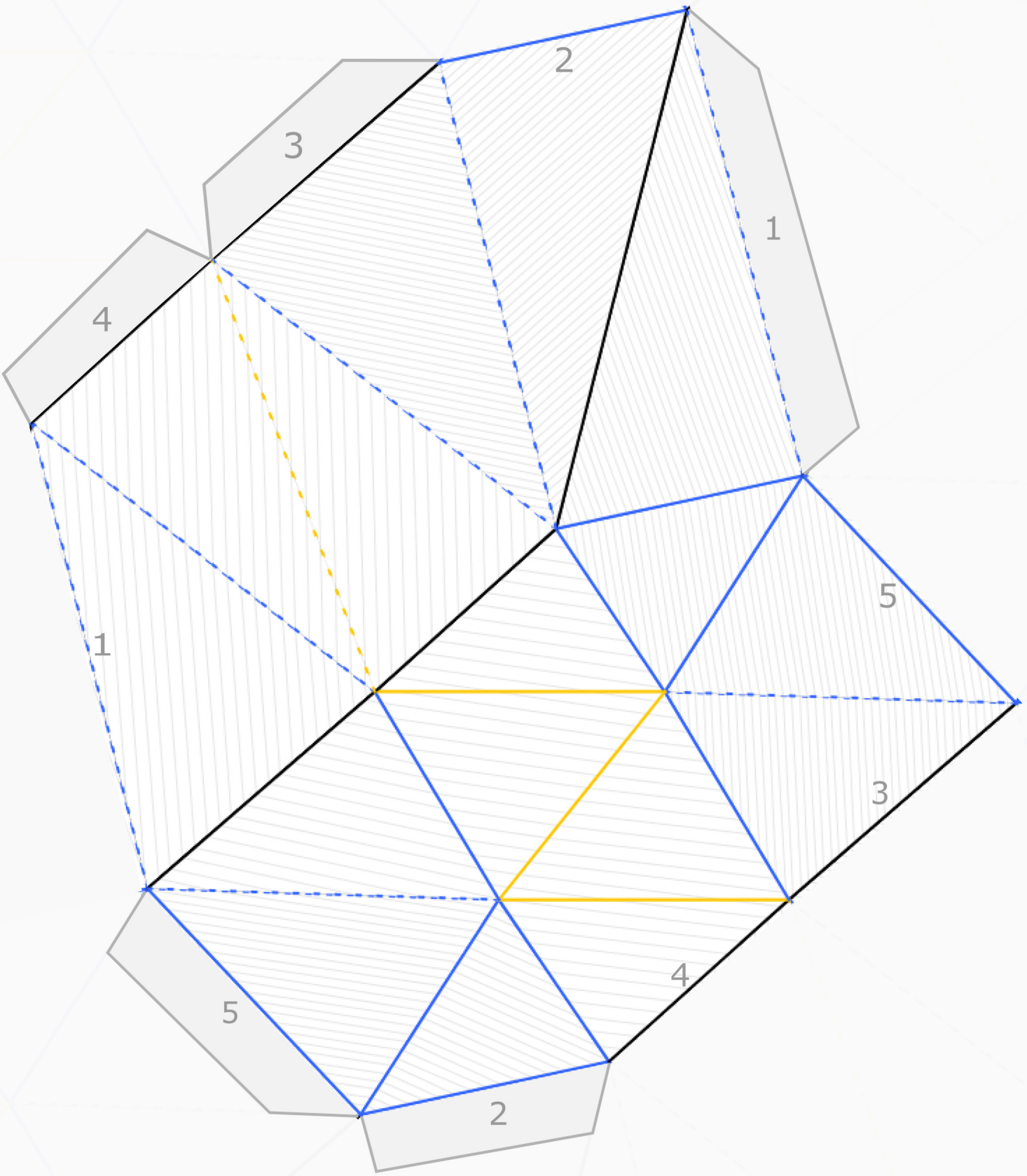
mountain

valley

gentle

moderate

sharp



Alba told me that it was important to have a file containing the edges of this pattern. So, here is a listing of all the edges, without their explicit names. I derived this list painstakingly from Equation 2.

(0.3364306460, 4.5292043541) – (1.5843170865, 4.5292043541)
(0.3364306460, 4.5292043541) – (1.1135536186, 5.2305035924)
(0.3364306460, 4.5292043541) – (−0.3679291669, 6.3930652003)
(0.3364306460, 4.5292043541) – (−1.1450521395, 5.6917659621)
(0.3364306460, 4.5292043541) – (−0.6445277505, 3.6845967037)
(0.3364306460, 4.5292043541) – (0.8705902833, 3.6376921275)
(1.5843170865, 4.5292043541) – (1.1135536186, 5.2305035924)
(1.1135536186, 5.2305035924) – (−0.3679291669, 6.3930652003)
(−0.3679291669, 6.3930652003) – (−1.1450521395, 5.6917659621)
(−1.1450521395, 5.6917659621) – (−0.6445277505, 3.6845967037)
(−0.6445277505, 3.6845967037) – (0.8705902833, 3.6376921275)
(0.8705902833, 3.6376921275) – (1.5843170865, 4.5292043541)
(−0.6445277505, 3.6845967037) – (0.2765985836, 2.7084684966)
(0.8705902833, 3.6376921275) – (0.2765985836, 2.7084684966)
(0.2765985836, 2.7084684966) – (1.3413537512, 2.9363928892)
(0.8705902833, 3.6376921275) – (1.3413537512, 2.9363928892)
(0.8705902833, 3.6376921275) – (2.1184767238, 3.6376921275)
(1.3413537512, 2.9363928892) – (2.1184767238, 3.6376921275)
(1.5843170865, 4.5292043541) – (2.1184767238, 3.6376921275)
(2.1184767238, 3.6376921275) – (3.0994351203, 4.4822997779)
(1.5843170865, 4.5292043541) – (3.0994351203, 4.4822997779)
(1.5843170865, 4.5292043541) – (2.1783087863, 5.4584279850)
(3.0994351203, 4.4822997779) – (2.1783087863, 5.4584279850)
(1.1135536186, 5.2305035924) – (2.1783087863, 5.4584279850)
(2.1783087863, 5.4584279850) – (1.6777843972, 7.4655972434)
(1.1135536186, 5.2305035924) – (1.6777843972, 7.4655972434)
(1.1135536186, 5.2305035924) – (0.6130292296, 7.2376728507)
(1.6777843972, 7.4655972434) – (0.6130292296, 7.2376728507)
(0.6130292296, 7.2376728507) – (−0.3679291669, 6.3930652003)

5 Near Flatness

5.1 The Main Result

Let T be the torus from Equation 1. We say that T is ϵ-*flat* if $\max_a |\theta_a - 2\pi| < \epsilon$. The maximum is taken over all 8 cone angles. In this chapter we prove the following result using rational arithmetic.

Lemma 5.1 (Near Flat) *T is 10^{-30}-flat.*

Let Ω be the integer torus we get by scaling T up by 10^{32}. It suffices to prove that Ω is 10^{-30}-flat. Again, in [**S**] I used Mathematica to establish 10^{-32}-flatness. The proof on [**S**] only uses 10^{-16}-flatness.

Let θ_i denote the cone angle of Ω at the (i)th vertex. Here we discuss the idea behind the rational arithmetic proof of the Near Flat Lemma. Figure 1.3 shows a plot of the intrinsic flat structure on Ω up to an error of about 10^{-16}. Around each vertex V_i we have 6 triangles $T_{i0}, ..., T_{i5}$. These triangles do not necessarily fit perfectly in the plane, but the error is so small that you cannot tell from the naked eye. Call the union of these 6 triangles a *flower*. The (i)th flower corresponds to the (i)th star in Ω, though only approximately.

For our proof below, we compute high precision versions of the flowers, then scale the coordinates up by 10^{32}, then round down all the coordinate to integers. This gives us integer models which approximate the geometry of the 8 stars in Ω. Now something very nice happens. For our scaled-up models, these big integer triangles fit together perfectly around a vertex. Put another way, for each flower F there are 6 integer vectors $V_0, ..., V_5$ such that each of the 6 triangles in F is defined by 2 cyclically consecutive vectors on the list. We then compute the mismatch between the various dot products associated to Ω and the corresponding dot products associated to the integer flowers, and this gives us the Near Flat Lemma. We list the coordinates for our flowers at the end of the chapter.

Rather than compare the angles, we compare the dot products which go into the calculation of the angles. We finish off the argument by using the Lipschitz properties of the function that converts between the relevant dot product ratio and the angle. We perform exact integer arithmetic calculations using the BigInteger class. Out of a slight laziness, and a desire for a transparent argument over a tedious exercise that anyone can perform, we will avail ourselves of the BigDecimal class in Java, which is perfectly capable of computing the ratios of 64 digit integers up to 100 digits of precision.

5.2 The Proof

Let τ_{ab} be the (b)th triangle in the (a)th star of Ω and let τ^*_{ab} be the (b)th triangle in the (a)th integer flower. Let V_{ab} and W_{ab} be the two vectors that τ_{ab} defines. By this we mean that V_{ab} points from vertex a to the first other vertex of τ_{ab} and W_{ab} points from vertex a to the second other vertex of τ_{ab}. (The torus is oriented, so we have no trouble defining "first" and "second" here.) We define V^*_{ab} and W^*_{ab} with respect to τ^*_{ab} in a similar manner. Let θ_{ab} and θ^*_{ab} respectively denote the angles of τ_{ab} and τ^*_{ab} at vertex (a).

When $r > 0$ we have

$$\theta = \phi(r), \qquad \phi(\cdot) = \arccos(\sqrt{(\cdot)}), \qquad r = \frac{(V \cdot W)(V \cdot W)}{(V \cdot V)(W \cdot W)}, \tag{3}$$

We take the positive branch of the square root function here.

We have $48 = 8 \times 6$ pairs of triangles to check. We get the following bounds which work uniformly for all these pairs:

1. $r, r^* \in I := [.00112, .77714]$ for all a, b.
2. $|r - r^*| < 1.07 \times 10^{-32}$.
3. $V \cdot W > 0$ and $V^* \cdot W^* > 0$.

Lemma 5.2 *ϕ is 15-lipschitz on I.*

Proof: We compute that

$$\left|\frac{d\phi}{dt}\right| = \frac{1}{2\sqrt{t(1-t)}}.$$

Establishing our bound is the same as establishing that

$$225 - \frac{1}{4t(1-t)} > 0, \qquad \forall t \in I$$

Establishing this bound is a routine exercise in algebra (or calculus) which we omit. ♠

Combining Lemma 5.2 with our calculations, and remembering that our flowers have 6 triangles in them, we see that Ω is ϵ-flat when

$$\epsilon = 6 \times 15 \times 1.07 \times 10^{-32} < 10^{-30}.$$

This completes the proof of the Near Flat Lemma.

5.3 The Integer Flowers

$$\begin{bmatrix} 124788644050109876602825361295674 & 0 & 0 \\ 77712297261878079581593486900079 & 70129923825301614779354253224291 & 0 \\ -70435981294870011898311359845207 & 18638608461746472826992416364452 4 & 0 \\ -14814827855674809147990484674528 6 & 11625616079216311349056991042023 2 & 0 \\ -98095839650912348417428186459704 & -84460765042234115458262663828023 & 0 \\ 53415963724939232100166950154753 & -89151222668009434223254123690615 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 84465310291672221527450441188266 & 0 & 0 \\ 69550368811175622295365728134 1642 & 10360961288458939257702412823687 1 & 0 \\ -34241316280724259280029981722340 & 10894738298538156157102604897962 1 & 0 \\ -10379168509189988223368726306398 3 & 53377701007921689940019207427 49 & 0 \\ -88338793860668172702704004953884 & -12318313380566464298633765169856 2 & 0 \\ 44045801850979783812535215764631 & -10110783497952139456112029011999 6 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 10467763544072396717998872341287 8 & 0 & 0 \\ 12034998257537687883214969911430 & 83603513440581754781321562053477 & 0 \\ -94317081172076477932017348008492 & 54413409707656384418499405421507 & 0 \\ -19163082603643258272353722077879 7 & -12813128623640095246346673704378 6 & 0 \\ -97313744864356104791519872770305 & -18254469594405733688196614246529 3 & 0 \\ 32097662001199218957367514938600 & -18556170794514120899918968774848 2 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 15158438888929791186644623305026 7 & 0 & 0 \\ 56495441368939829023723230650622 & 94715849040541184436182819641071 & 0 \\ -49223826337965360195502746428317 & 68639664650392174112843586146043 & 0 \\ -12472888956043572121402703625564 4 & -38613200962432610906628728821 49 & 0 \\ -68579908206882346371371696550992 & -91317009315546418606880471182019 & 0 \\ 56148981353553374842655339704651 & -87455689219303157516217598299871 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 19925109786397664915627628818699 2 & 0 & 0 \\ 16112081862613471590611107573640 7 & 97485840852430004244786950552576 & 0 \\ -44330167652533400164597159444753 & 12161928639913681496187205034431 6 & 0 \\ -10227774588042411966259058202240 8 & -18451785319703080875434679167189 & 0 \\ -44113235471988020751569542690915 & -11673143595475025651633741494232 2 & 0 \\ 38130279237841933250165212450584 & -97485840852430004244786950552577 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 20686355637674156862294612970694 5 & 0 & 0 \\ 36474813159413128630782636780 69 & 10882659067012618826980262906561 6 & 0 \\ -75789172382575842209671187858145 & 80117590481155307070115264260041 & 0 \\ -11699986056130538693975584715299 2 & -65757376990228984547591251604189 & 0 \\ -41210688178729544730084659294848 & -14587496747138429161770651586423 1 & 0 \\ 58216001420386574195913238485630 & -11561709115419974130709929353795 5 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 18831730499733982317415867992902 0 & 0 & 0 \\ 17841734520938330693404444524334 & 10314591543413373484327970531881 2 & 0 \\ -80328865359490685097501135729666 & 26108658194654254413402678301333 & 0 \\ -69693027059078945150953187346448 & -83662494202455064363150318793564 & 0 \\ 93593941525210768327841471388803 & -21066602504859540686475384106928 0 & 0 \\ 16328696858428971347879465873525 1 & -12700353084614034250160352227571 7 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 15158438888929791186644623305026 7 & 0 & 0 \\ 95435407535744537023790893345615 & 87455689219303157516217598299870 & 0 \\ -56239231360498057885401519560362 & 19907204678913036449385016022028 0 & 0 \\ -10571926770690518921922597707894 0 & -26076184390149010323339233495028 & 0 \\ -56495441368939829023723230650623 & -94715849040541184436182819641072 & 0 \\ 95088947520358082842723002399644 & -94715849040541184436182819641072 & 0 \end{bmatrix}$$

6 Other Pup Tents

I imagine that this section will grow as I learn more about the parameter space of pup tents. Here is the best example I have found.

$$
\begin{array}{lll}
+0.78 & -0.62 & z_0 \\
+0.25 & +0.51 & z_1 \\
+1.09 & +0.38 & z_2 \\
-0.25 & -0.51 & z_1 \\
-0.78 & +0.62 & z_0 \\
+0.64 & +0.20 & 0 \\
-1.09 & -0.38 & z_2 \\
-0.64 & -0.20 & 0
\end{array}
\qquad
\begin{aligned}
z_0 &= 0.9917\ 7247\ 8544\ 3862\ 8354\ 4208\ 7452\ 1339 \\
z_1 &= 0.9951\ 4687\ 2293\ 4807\ 1279\ 5909\ 2520\ 3831 \\
z_2 &= 0.9793\ 3367\ 3330\ 1556\ 3840\ 1007\ 6628\ 1139
\end{aligned}
\tag{4}
$$