Midterm Project for Computational Topology

Given a triangulation of a surface, decide the type of the manifold by computing its Genus

By: Jacob Stern and Tamar Yastrab

Abstract

In this project we sought to fully categorize 2-manifolds without boundary, based on a given triangulation of that topology. As input, our code uses the previous group's data structure for triangulating an unknown topology. We implemented methods that determine if the manifold is orientable, find the genus, and thus determine the type of manifold. To determine orientability, we check if all adjacent triangles are reversed along the shared edge. This process is recursively applied to every triangle in the triangulation, and if all triangles satisfy this condition, we can say that the manifold is orientable. It is a classical result that all 2-manifolds without boundary can be categorized if their genus g and orientability are determined. With the knowledge of the genus, we know the number of holes of the surface and can accurately determine the manifold.

In rigorous mathematics, a topology might classically be defined in the language of set theory. However, to gain a more intuitive grasp one can consider topology as a generalization of geometry. Features like length, size, and sides in geometry have deeper analogous properties in topology, where properties are preserved even if the original shape is stretched and bent. So long as no additional holes are added to the figure, topological figures can be molded, and for this reason it has been dubbed "rubber sheet geometry".

For our project, we will be considering 2-manifolds without boundary. Such a topology looks like a flat plane when one zooms infinitely close to the manifold. For example, a person standing on the Earth believes the ground to be flat even though it's really roughly spherical. A space without boundary has no point where one would "fall off", so a hollow tube, which is a 2-manifold, has a boundary because an object at the ends of the surface could fall off.

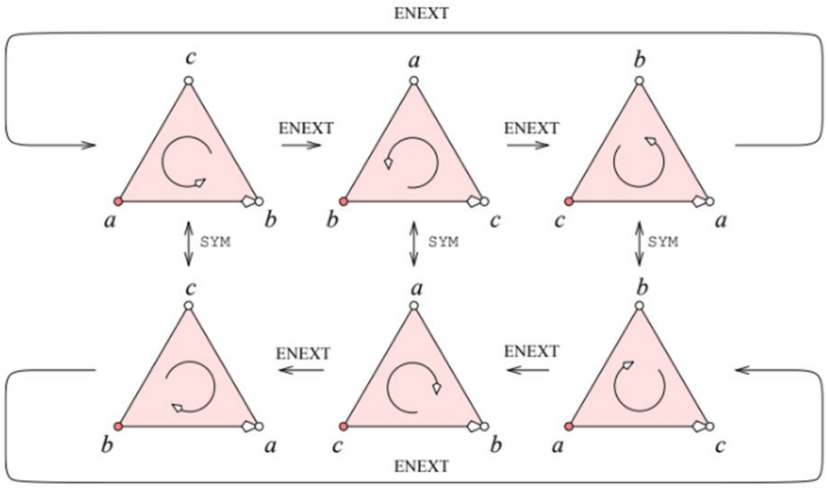
Image by Quanta Magazine

Often the qualities of 2-manifolds without boundary are easily defined by the naked eye, but identifying holes and edges becomes an interesting challenge when represented to a computer. One way to represent such a surface to a computer is through a triangulation. A triangulation is a way to cover the surface of the shape with triangles, as seen in the photo. This representation preserves the structure of the 2-manifold by simply cutting the

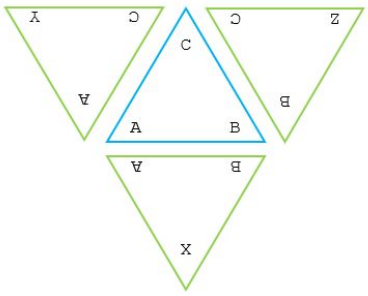
original shape into triangles. As explained in Etta and Elisheva's project, a computer can be fed triangles and orientations to work out the overall structure of the original shape. These tringluzations can be stored and manipulated by a computer where abstract topologies could not.

Our project calculates the genus of a 2-manifold without boundary given a triangulation. The genus of a topology is an important invariant that can help categorize topologies, and by determining if the surface is orientable, the genus can even categorize 2-manifolds without boundary.

Given a triangulation, our code first tests for orientability by checking the neighbors of every triangle covering the manifold and ensuring that the orientations are aligned. If a manifold is orientable, all adjacent triangles should have opposite orientations. To check this condition, we must look at a triangle vis a vis its adjacent triangles to ensure that their shared edge have opposite orientations, and we repeat this process for every triangle in the triangulation. The textbook illustrates the six possible ordered triangles as follows:



The pink circle indicates the lead vertex and follows across to the right to form the lead edge of the triangle. Each triangle is assigned a name, μ, which will be the name of the vertices starting with the leading vertex, then the vertex at the other end of the leading edge, and finally the third vertex. In addition, each triangle is given an orientation, ι, labelled 0,1,2,4,5,6 respectively. These three bit integers can inform us of the triangle's orientation by checking to see if the first bit is 0 or 1 (0=**0**00, 1=**0**01, 2=**0**10 vs. 4=**1**00, 5=**1**01, 6=**1**10). There are a few short functions written into the data structure which are very important because they allow us to move from one triangle to the next. The function ENEXT shifts the vertices of the triangle once in a clockwise motion by incrementing ι by 1 modulo 3. SYM rotates the triangle along its vertical axis by adding 4 to ι modulu 8. Notice that ENXT preserve the triangle's original orientation, whereas SYM will reverse the triangles original orientation. Now that we can move within the triangles, we can move from one triangle in the broader triangulation to the next. The function ORG returns a pointer to the original triangle, and FNEXT will return a pointer to a triangle that shares a leading edge with the original triangle.



Being able to move throughout the triangulation enables us to check if the triangles within the triangulation are collectively orientable. For a triangle to be orientable, the surrounding triangles must have opposite orientations.
Here, the center triangle has $\mu = abc$ and $\iota = 0$.
Note, each of the adjacent triangles share one of the three edges with the triangle in the center, but their orientation is reversed. For example, the triangle *bax* shares an edge with *abc*, but it is in reverse orientation. This is true for each of the adjacent triangles, which ensures that the overall shape is orientable. If any triangle in the triangulation fails this test, the manifold is not orientable.

The genus of an orientable 2-manifold without boundary can be obtained through the following formula:

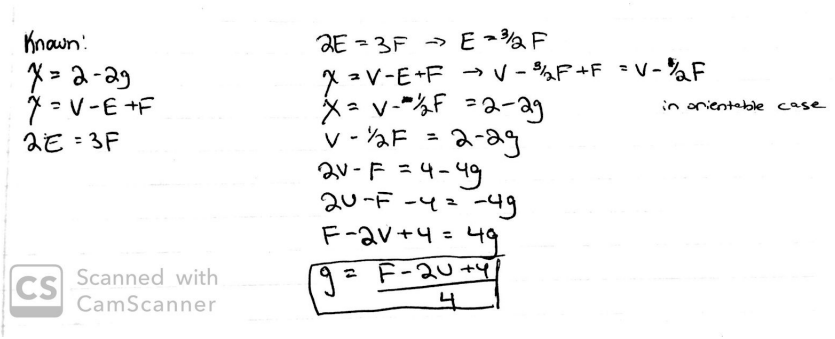
$\chi = 2 - 2g$ such that $g = \frac{2-\chi}{2}$

Where χ is the Euler characteristic of the triangulation. It is a classical result in topology that all orientable 2-manifolds without boundary are just Tori with a different number of holes, and that the genus represents the number of holes. For non-orientable 2-manifolds without boundary, we derive the formula by imagining doubling the manifold connecting it to itself, getting an orientable manifold, noting that the euler characteristic has obviously doubled. Thus, we observe: $g = 2 - \chi$.

For non-orientable 2-manifolds without boundary, the meaning of the genus is less clear. However, the genus does fully categorize non-orientable 2-manifolds without boundary. The most important takeaway from this is that with the genus and the orientability of a 2-manifold without boundary we can perfectly describe what topology it is.

Once the orientation of a shape has been determined, we employ further calculations to derive more information about the shape. To find the genus, we use a simplified formula of the Euler Characteristic. By realizing that for every edge is connected to two triangles and every triangles has three edges it is clear that, twice the number of edges is equal to three times the number of faces.

We also know that $\chi = V-E+F$. We therefore conclude the genus' value as follows:



This result is also easily understood for the non-orientable surface by adjusting χ appropriately. Therefore, our code checks for the figure's orientability and then returns the appropriate genus. From just the triangulation of a 2-manifold without boundary, we have now fully classified it.

Understanding the intuitive meaning behind the genus is the number of holes in the orientable 2-manifold without boundary, though this idea does not expand well into non-orientable surfaces. New 2-manifolds can be built by adding other 2-manifold topologies.

If we cut a 2d hole in a sphere and cut a 2d hole in a torus, attaching those manifolds by the hole would create another 2-manifold without boundary. This surface is clearly homeomorphic to the torus. By adding more tori to the sphere we can see that the number of holes in the new manifold is equal to the number of tori added to the sphere. This is another interpretation of the genus of orientable 2-manifold without boundary. This result can be easily expanded to non-orientable surfaces.

If we start with a sphere and like before cut out a circular hole, but now instead of attaching a torus, we attach a mobius strip by its boundary (which is also a circle) we will get a 2-manifold without boundary. But because the mobius strip was non orientable, the entire manifold will be unorientable. We can call this addition of a mobius strip a cross-cap. Now as before, we can add any number of mobius strips to the sphere, and get a non-orientable 2-manifold without boundary. The genus represents the number of cross-caps, or the number of mobius strips added to the original sphere.

With this information, we can now take the triangulation of any 2-manifold without boundary and fully understand its topological properties

We will now analyze how to determine orientability, genus, and determination of the manifold in code. To determine orientability, we check the three adjacent triangles, b_x, b_y, and b_z in code. The three adjacent triangles should have opposite orientations from the original triangle, and thus correspond to the three rotations of the reverse orientation. Meaning, if the original triangle's orientation was clockwise, the adjacent triangles should be the three permutations of the counterclockwise orientation. In order to determine this ι, we perform the following three operations:

```
def isOrientable(self, i):
    if not self.isVisited:
        self.isVisited = True
        x = self.isOrientable(FNEXT(SYM(self, i)))
        y = self.isOrientable(FNEXT(ENEXT(SYM(self, i))))
        z = self.isOrientable(FNEXT(ENEXT(ENEXT(SYM(self, i)))))
        return x and y and z
```

Notice that x, y, and z (corresponding to b_x, b_y, and b_z) take the mirror orientation of the original and progressively rotate, such that each rotation of the reverse orientation is considered. Each time isOrientable() is called, it returns True if all adjacent triangles have the same orientation. This function is recursive, meaning that it will recursively analyze every node, or triangle, in the larger triangulation, so that if even one node fails, a False value in x,y, or z will cause the entire function to return False.

Let's consider our case, where the original triangle has $\mu = abc$ and $\iota = 0$.

x = SYM(*abc*, 0), giving us μ = *bac* and ι = 4. Notice that the last vertex doesn't matter for the purposes of the triangulation, because it does not share an edge with the original triangle, so we can call it μ = *abx*
y = ENEXT(SYM(*abc*, 0)), giving us μ = *cya* and ι = 5. This is because we first rotated the triangle and then reversed the orientation.
z = ENEXT(ENEXT(SYM(*abc*, 0))), giving us μ = *zcb* and ι = 6 by rotating once more before reversing the orientation.
Because this function recursively moves throughout the triangulation following ι, the function should first arbitrarily choose a direction and test the other triangles against that choice.

```
def isOrientable(self, i):
    if not self.isVisited:
        self.isVisited = True  # remember that we visited this triangle
        i = 0                  # arbitrarily  choose a counterclockwise orientation
        x = self.isOrientable(FNEXT(SYM(self, i)))
        y = self.isOrientable(FNEXT(ENEXT(SYM(self, i))))
        z = self.isOrientable(FNEXT(ENEXT(ENEXT(SYM(self, i)))))
        return x and y and z   # if even one edge fails, this will return False
```

We wrote our code based on this idea, but because the current data structure does not remember ι when the triangle is entered into the triangulation, we simply asked that the user enter her vertices as an ordered triangle, and rather than move along triangles recursively by modifying ι, we check to see if the adjacent triangles' shared edge is reversed. Taking the previous example, our original μ = *abc*, and the adject triangle had μ = *bax*. We checked to see that the shared edge, in this case, *ab*, is reversed in the adjacent triangle as *ba***.** Our code also is careful to check that triangles like μ = *cya* are considered even though there's a foreign vertex in μ.

```
def isOrientable(self):
    for triangle in self.nodes.keys():            # loops through triangulation
        x = self.fnext(triangle[0:2], triangle)   # considers the three edges
        y = self.fnext(triangle[1:3], triangle)
        z = self.fnext(triangle[0:1] + triangle[2:3], triangle)
        ans = x and y and z
        if ans == False: return False
    return True


def fnext(self, given, original):
    a = given[0]
    b = given[1]
    for node in self.nodes.keys():
        if node != original:
            x = node.find(a)              # if both vertices are found, we know this
            y = node.find(b)              # triangle has a shared edge
            if x > -1 and y > -1:         #
                if x > y: return True     # if the order of the vertices are reversed
                else: return False        # this triangle is orientable
```

Our code determines the manifold with some special names given to certain surfaces, and generalizes based on the genus for the rest.

```
def genus(self):
    orientable = self.graph.isOrientable()
    if orientable: return self.manifold((self.numFaces() - 2*self.numVertices() +4)/4, orientable)
    else: return self.manifold((self.numFaces() - 2*self.numVertices() +4)/2, orientable)

def manifold(self, genus, orientable):
    if genus == 0.0 and orientable: return "orientable sphere"
    if genus == 1.0 and orientable: return "orientable torus"
    if genus == 2.0 and orientable: return "orientable double-torus"
    if genus == 0.0 and not orientable: return "npn-orientable projective plane"
    if genus == 1.0 and not orientable: return "non-orientable Klein Bottle"
    answer = str(genus) + "-hole manifold"
    return answer
```

Let's take the torus as an example. The Torus's most basic triangulation has 2 triangle faces and 1 vertex. We also know that the Torus is orientable, and so we observe:
G = (2 - 2(1)+4) / 4 = **1**. This is the correct number of holes in the Torus.

Conclusions

We were successful in fully categorizing 2-manifolds without boundary, based on a given triangulation of that topology. The two primary functions we implemented were isOrientable() and GENUS(), which determined the surface's manifold. Because a complete triangulation has a fixed number of triangles, and triangle has only three adjacent triangles, isOrientable runs in O(n) time, where n correlates to the number of triangles. GENUS() checks the orientable condition to return the genus, and this process runs in O(c) time, because given the orientation, the genus is calculated in constant time no matter the size of the triangulation. This can happen because as triangles, or nodes, are added to the data structure, the number of faces, edges, vertices, and triangles are preserved and can be called upon in constant time. Further paths for research could be improving the speed of the code, and handling more kinds of topologies, for example 2-manifolds with boundaries.

Works Cited

[1] Edelsbrunner, Herbert, and J Harer. Computational topology : an introduction. Providence, R.I: American Mathematical Society, 2010. Print.

[2] Triangleinequality. “Testing a Manifold for Orientability.” Triangleinequality, Triangleinequality, 9 Oct. 2013, triangleinequality.wordpress.com/2013/10/09/testing-a-manifold-for-orientability/amp/.