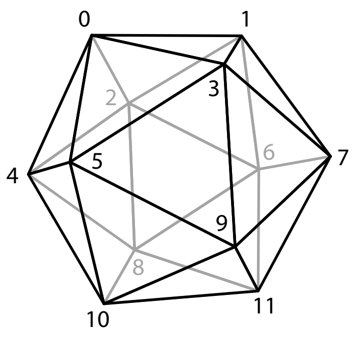
CS 465 Homework 8

out: Friday 5 November 2004
due: Friday 12 November 2004

Problem 12: Triangle meshes

1. Give as compact a representation as you can of the following triangle mesh in terms of (a) indexed triangles; (b) indexed triangle strips; (c) indexed triangle strips and fans:

   Assume the vertex positions are already known; you just need to give the indices in the right order. Be careful to keep the faces oriented outward.

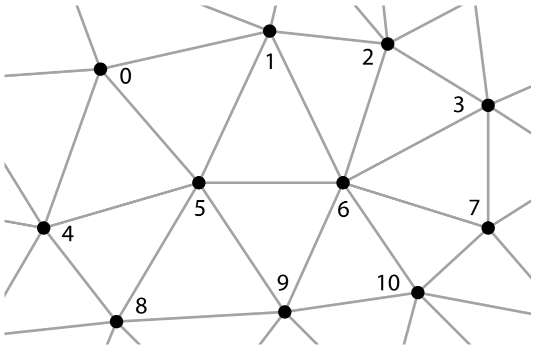
   For instance, I could represent a tetrahedron using triangles [0 1 2, 0 2 3, 0 3 1, 3 2 1]; or using triangle strips [0 1 2 3, 2 3 0 1]; or using the fan [0 1 2 3 1] and the strip [3 2 1].

I want to make a program that automatically finds triangle strips, using the greedy algorithm shown in Figure 1. The program uses a data structure consisting of two arrays of integers, each of size n_T by 3. The first, `tInd`, is the same as in the lecture slides: it stores the three vertex numbers for each triangle. The second array, `tNbr`, holds adjacency information. The integer stored in `tNbr[`k`][0]` is the number of the triangle that neighbors triangle k across the edge between the zeroth and first vertices of triangle k. Similarly, `tNbr[`k`][1]` is the neighbor across the edge between vertices 1 and 2, and `tNbr[`k`][2]` is the neighbor across the edge between vertices 2 and 0. For instance, a tetrahedron could have `tInd` = [0 1 2, 0 2 3, 0 3 1, 3 2 1] and `tNbr` = [2 3 1, 0 3 2, 1 3 0, 1 0 2].

```
mark all triangles as unvisited
**while** (not all triangles visited) {
  choose an arbitrary unvisited triangle $T$
  mark $T$ as visited
  create a triangle strip $S$ consisting of just $T$
  **while** (true) {
    find the triangle $T'$ that could be next in the strip*
    **if** ($T'$ is visited) **break**
    mark $T'$ as visited
    add $T'$ to $S$*
  }
  output $S$
}
```

Figure 1: The greedy triangle strip building algorithm

2. Suppose I'm working on the mesh shown here:



   and I have built the strip [8 9 5 6]. If the program is going to continue that strip, what is the next vertices it has to add and why? What would be the vertex after that?

3. Write pseudocode that uses the `tInd`/`tNbr` data structure to implement each of the two starred steps in Figure 1. Assume that S is a list of vertex numbers stored in a reasonable data structure that allows appending, querying the length, etc., and that you know the triangle number of the most recently added triangle. Note that adding a triangle to the strip amounts to appending one vertex to S.

   You can assume the mesh is a manifold—it's OK if you crash on a nonmanifold input mesh.