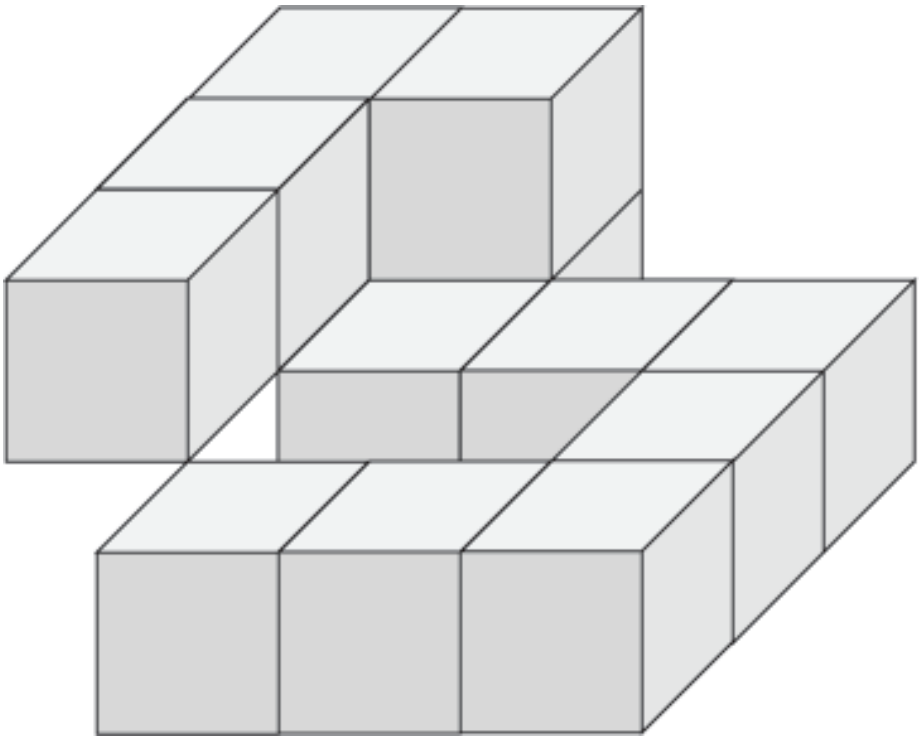
Surface Model

The frontier of a 1- or 0-region of voxels splits in general into frontiers of a finite number of 2-regions; we will consider only 2-regions here. (In terms of the grid cell model, we only consider 6-regions rather than 18- or 26-regions.)

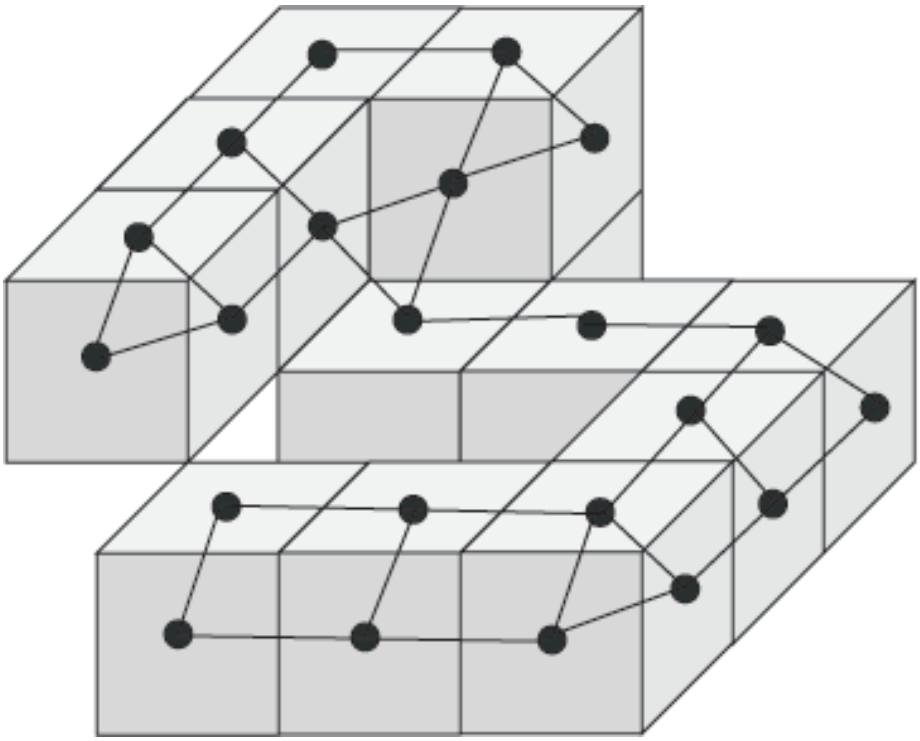
The frontier of a 2-region of voxels consists of 2-cells (faces), 1-cells (edges), and 0-cells (vertices). Each frontier edge is incident with at least two frontier faces. Note that it is also possible that one frontier edge is incident with four frontier



faces, or one frontier vertex with six frontier edges. The shown example is a 2-region.

Frontier Graph

Consider the frontier faces to be the nodes of an undirected graph $[F, A]$ (we call it *frontier graph* assuming a frontier of a 2-region) which are connected by an edge iff the frontier faces share a frontier edge.



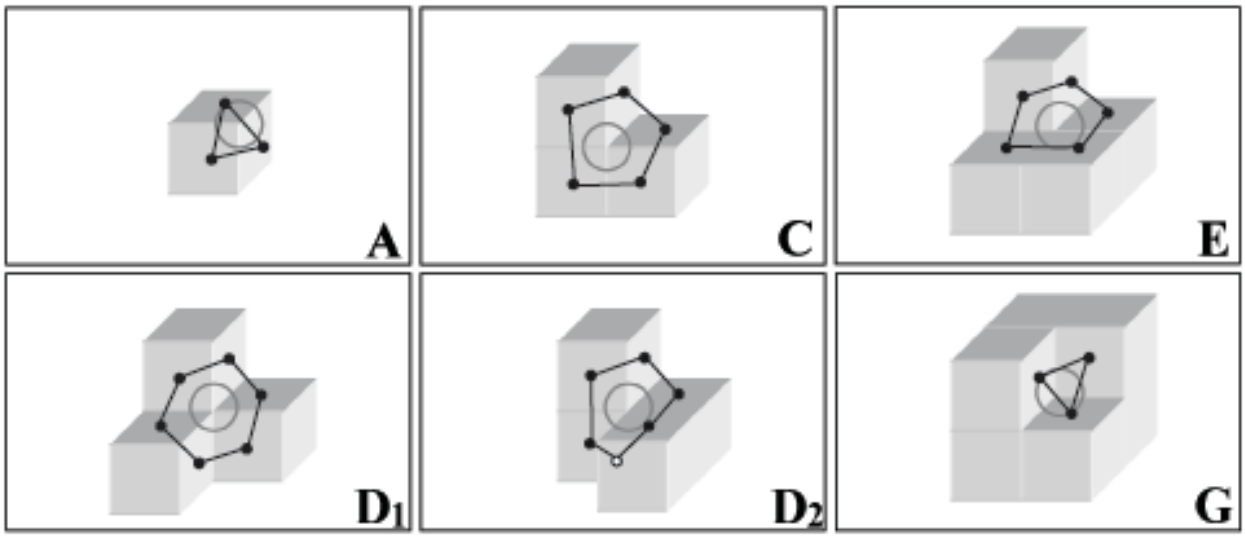
In this figure, only those nodes are shown where midpoints of frontier faces are visible, and only those edges where both endnodes are shown.

Theorem 1 (K. Voss, 1993) *All nodes in a frontier graph* $[F, A]$ *are of degree 4.*

Faces of Frontier Graphs

We assume local cyclic orders for this graph, defined by clockwise orientations (looking from the outside onto the frontier of the 2-region).

A frontier graph of a simply-connected 2-region is planar, and its cycles are of length 3, 4, 5, or 6.



These are all possible angles of a frontier of a 2-region.

Cycles in the frontier graph, which are not 'around' one angle, are always of length 4.

Conclusion: we have uniformity of the degree of nodes (i.e., always four edges for each face), but not uniformity of cycle length.

FILL Algorithm

The FILL procedure discussed in Lecture 03, can also be applied to visit all nodes (i.e., all frontier faces) of a frontier graph.

For each voxel (x, y, z) we consider a 6-tuple representing its six faces (in a fixed order, such as top face first (which is in the plane $Z = z + 0.5$), then the one in the plane $X = x - 0.5$, then the one in the plane $Y = y + 0.5$, then the one in the plane $X = x + 0.5$, then the one in the plane $Y = y - 0.5$, and finally the bottom face (which is in the plane $Z = z - 0.5$). The algorithm is as follows:

(1) We start at one frontier face f of one voxel p with a new label L (e.g., write this label into the position of this face at voxel p) and put f into a stack.

(2) If the stack is empty, then stop.

(3) Pop f out of the stack, label with L all frontier faces that are edge-adjacent to f and have not yet been labeled, and put these frontier faces into the stack.

(4) Go to Step (3).

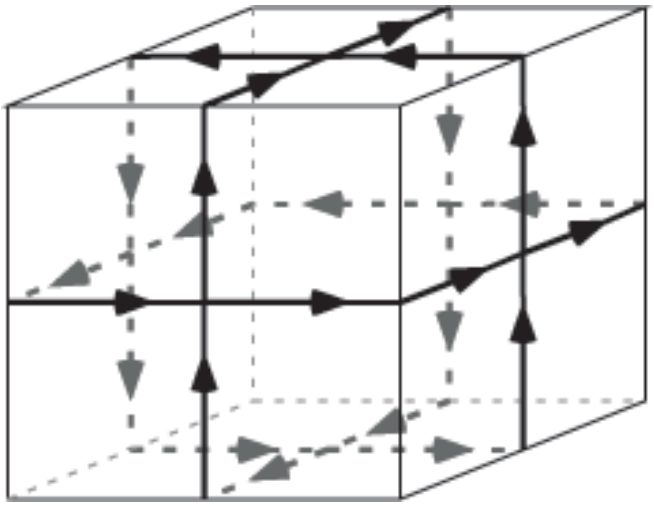
This algorithm does not attempt to minimize the number of accesses to frontier faces to determine whether they have already been labeled; each frontier face will be accessed four times, including the first visit, when the face is labeled.

In Lecture 25 we discussed that a frontier graph can be visited in form of a Hamilton path if the 2-region is a simple arc or curve. However, this is not possible anymore for the general case of arbitrary 2-regions.

The following algorithm requires **at most two visits to each face**, including the first visit.

Artzy-Herman Algorithm (1981)

We consider three circuits in the frontier graph that represents the faces of a single 3-cell.



There are three circuits, each of which is parallel to one of the coordinate planes. Each face has two *outgoing* and two *incoming edges* defined by the orientation of these circuits.

We assume a closed 2-region M of voxels in the 3D grid.

Let $[F, A]$ be the frontier graph in which each node represents a frontier face of M.

Two nodes f_1 and f_2 are adjacent in this graph (we write $f_1 A f_2$) iff the corresponding faces are 1-adjacent.

Each $f \in F$ is incident with one 3-cell $I(f)$ in M and with one 3-cell $O(f)$ in $\overline{M}$.

$f_1 A f_2$ iff

(1) $O(f_1) = O(f_2)$ (so that $I(f_1)$ and $I(f_2)$ are 1-adjacent) or

(2) $O(f_1)$ and $O(f_2)$ are 2-adjacent (so that $I(f_1)$ and $I(f_2)$ are also 2-adjacent) or

(3) $O(f_1)$ and $O(f_2)$ are both 2-adjacent to the same 3-cell in $\overline{M}$ (i.e., $I(f_1) = I(f_2)$).

Assume that the figure on the previous pages illustrates a voxel $I(f)$. If we start at any face f in the frontier of M, the two outgoing edges of f on $I(f)$ point to two frontier faces called the *out-faces* of f, and the two incoming edges are outgoing edges of two *in-faces* of f.

The Artzy-Herman algorithm for tracing all faces of a 3D frontier:

1. Create a list F of faces and push f_0 into F; create a queue Q of faces, and push f_0 into Q; create a list L of labeled faces, and push f_0 twice into L.
2. While the queue Q is not empty:
   (a) pop face f out of Q (and delete it from Q),
   (b) for both out-faces g of f, do the following:
      i. if g is on L, delete it from L,
      ii. otherwise, push g into F, push g into Q, and push g into L.

This frontier tracing algorithm makes use of the directed graph structure of the shown circuits.

We accumulate all of the frontier faces of M in the list F. The queue Q implies a breadth-first access order; using a stack instead implies a depth-first access order.

We can return to a face at most twice; the second copy of f_0 on L is removed when we enter Step 2.b.i for the first time.

Example

Assume we only have one voxel in M, and consider the figure on page 5. We have six frontier faces: t (top), l (left), r (right), c (close), d (distant), and b (bottom). We start with face t.

Initialization: In Step 1 we create list $F = [t]$, queue $Q = [t]$, and list $L = [t, t]$.

Loop

In Step 2.a we pop out $f = t$ and have $Q = \emptyset$.

In Step 2.b we identify l and d as being the out-faces of $f = t$.

First visit of l: Face l is not on L, we cannot delete it there; so we obtain $F = [l, t]$, $Q = [l]$, and $L = [l, t, t]$.

First visit of d: Face d is not on L; we obtain $F = [d, l, t]$, $Q = [d, l]$, and $L = [d, l, t, t]$.

Loop

In Step 2.a we pop out $f = d$ and have $Q = [l]$.

In Step 2.b we identify l and b as out-faces of $f = d$.

Second visit of l: Face l is on L; we delete it there and have $L = [d, t, t]$.

First visit of b: Face b is not on L; we obtain $F = [b, d, l, t]$, $Q = [b, l]$, and $L = [b, d, t, t]$.

Loop

In Step 2.a we pop out $f = b$ and have $Q = [l]$.

In Step 2.b we identify c and r as out-faces of $f = b$.

First visit of c: Face c is not on L; we obtain $F = [c, b, d, l, t]$, $Q = [c, l]$, and $L = [c, b, d, t, t]$.

First visit of r: Face r is not on L; we obtain $F = [r, c, b, d, l, t]$, $Q = [r, c, l]$, and $L = [r, c, b, d, t, t]$.

Loop

In Step 2.a we pop out $f = r$ and have $Q = [c, l]$.

In Step 2.b we identify d and t as out-faces of $f = r$.

Second visit of d: Face d is on L; we delete it there and have $L = [r, c, b, t, t]$.

First visit of t: Face t is on L; we delete it there and have $L = [r, c, b, t]$.

Loop

In Step 2.a we pop out $f = c$ and have $Q = [l]$.

In Step 2.b we identify t and r as out-faces of $f = c$.

Second visit of t: Face t is on L; we delete it there and have $L = [r, c, b]$.

Second visit of r: Face r is on L; we delete it there and have $L = [c, b]$.

Loop

In Step 2.a we pop out $f = l$ and have $Q = \emptyset$.

In Step 2.b we identify b and c as out-faces of $f = l$.

Second visit of b: Face b is on L; we delete it there and have $L = [c]$.

Second visit of c: Face c is on L; we delete it there and have $L = \emptyset$.

Final Result

We stop here and have $F = [r, c, b, d, l, t]$ — all the faces of the frontier.

Efficiency and Time Complexity

It is sufficient to represent only three faces for each voxel: faces in planes $X = x + 0.5$, $Y = y + 0.5$, and $Z = z + 0.5$.

It is possible to implement list L in a way such that deletion and insertion only takes a constant time (based on coordinates and labeling of faces of voxels).

It is also possible to identify both out-faces of a given frontier face in constant time (based on coordinates and membership of voxels in the given 2-region).

Thus, if n is the number of frontier faces, the algorithm runs in $\mathcal{O}(n)$ time, visiting each frontier face twice.

Coursework

Related material in textbook: Section 8.4.1.

A.26. [6 marks] Implement a frontier tracing algorithm (either the non-optimized FILL algorithm, or the Artzy-Herman algorithm).

As input pictures, generate digitized spheres and cubes (Gauss digitization) at some grid resolution (say, at least $64 \times 64 \times 64$).

Visualize the traced frontiers. (Hint: you may use standard 3D visualization software for this.) For example, you should be able to generate a picture as follows: