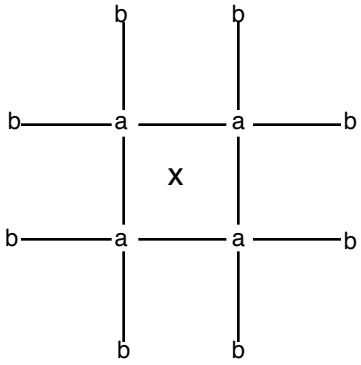
Quadrilateral Recursive Interpolating Subdivision Scheme

Lambrecht, Bram; and Sammis, Ian

The most common case for a quadrilateral mesh is one in which every vertex has valence four. Let's start by devising a reasonable scheme for subdividing that sort of environment, then try to extend it do handle different valences and boundary edges.

We'll use the simplest subdivision scheme that we can imagine, and at each stage add exactly one new vertex in the center of each quadrilateral.

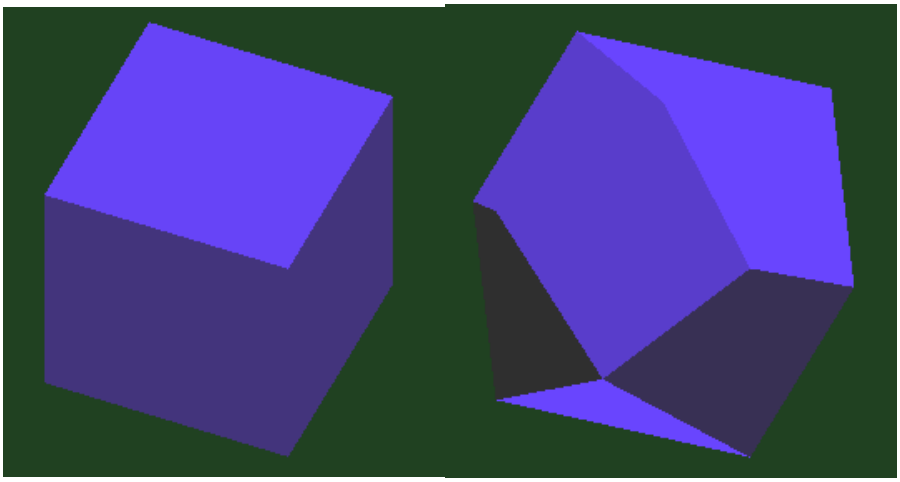


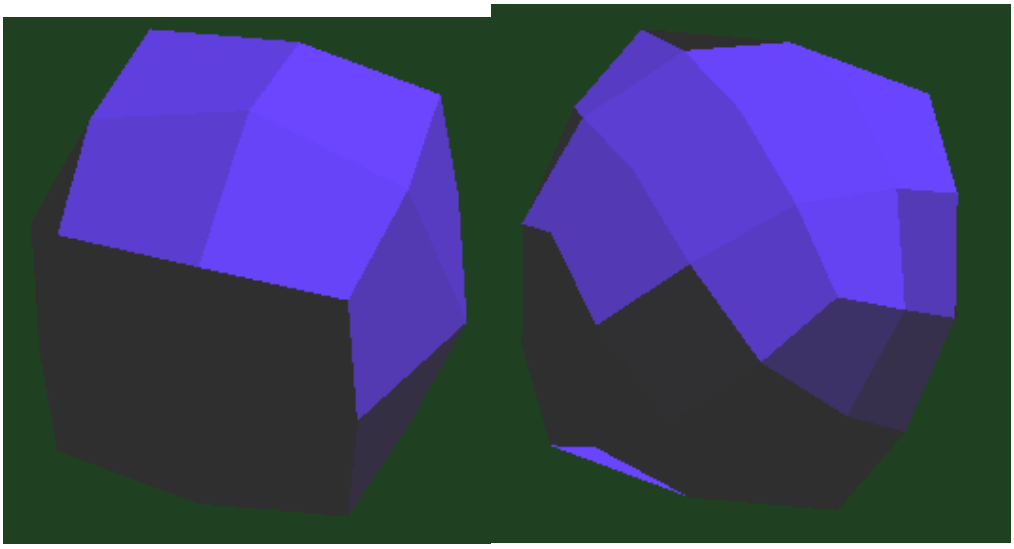
If you imagine the degenerate case where one dimension is much larger than the other, we feel that the scheme should morally reduce to one-dimensional cubic interpolation, with weights -1/16 on the "outward" points and 9/16 on the "inward" ones. If we divvy those weights equally amongst the *a*s and *b*s, we find that each *a* should have weight 5/16 and each *b* weight -1/32.

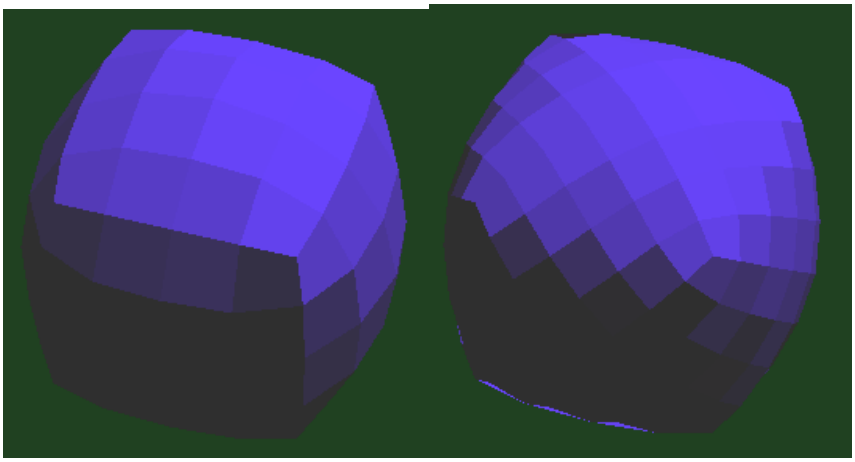
Since new edges cross exactly one edge of previous iteration quadrilaterals, and we deal with only quadrilateral meshes, the number of irregular valence ($\neq 4$) vertices stays constant throughout the subdivision. Thus, most vertices are handled by the regular case subdivision. However, the first subdivision step sets the general shape of the limit surface and deals with the highest percentage of irregular valence vertices. Thus, a good scheme must be devised to handle these vertices.

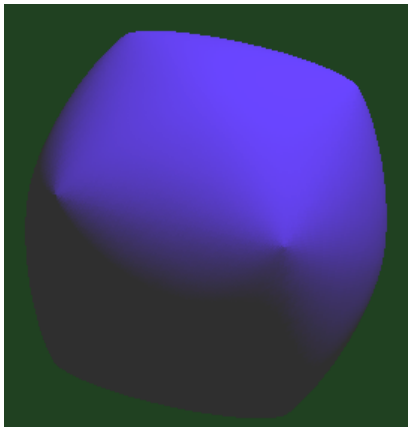
Our implementation takes the following approach. The new vertex calculated for each quadrilateral is the weighted sum of the existing quadrilateral's vertices and each vertex's nearest neighbors. A vertex of the existing quadrilateral is given a weight of $\frac{1}{4}+\epsilon$, and each of its nearest neighbors is given a weight of $-\epsilon/(k-2)$, where k is the valence of the vertex. For the regular case, $\epsilon=1/16$. However, it is not immediately clear what ϵ should be for other valences. As a first guess, we continue to use $\epsilon=1/16$.

Here's a cube, after repeated subdivision:







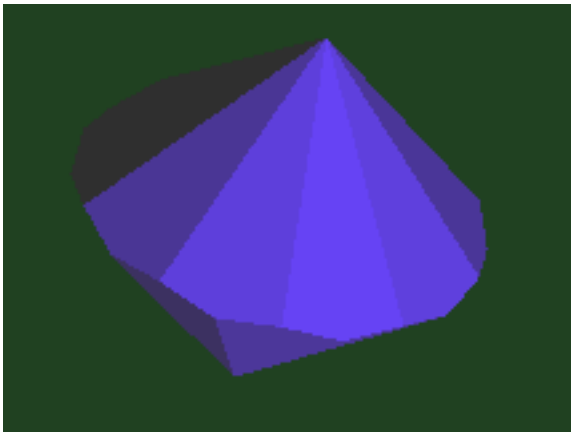


After much more subdivision:

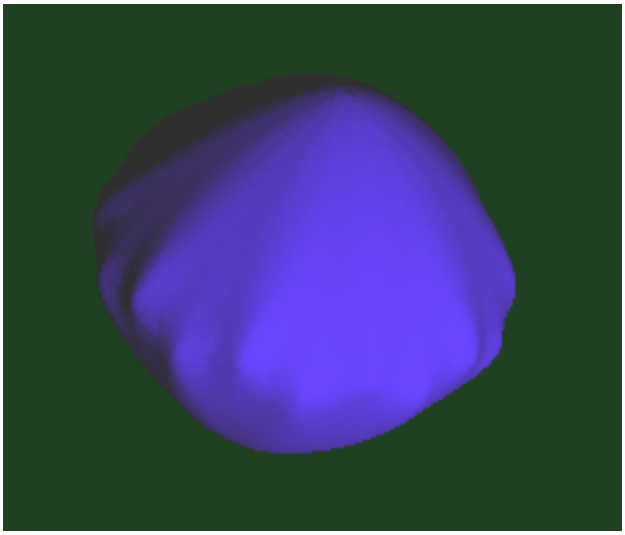
It isn't a sphere, obviously, but it's at least plausibly smooth.

The corners are a bit ugly. Perhaps varying ε to be valence-dependent would be preferable?

Let's see how the algorithm treats high-valence nodes. We constructed a figure like a top, with high valence nodes at top and bottom:

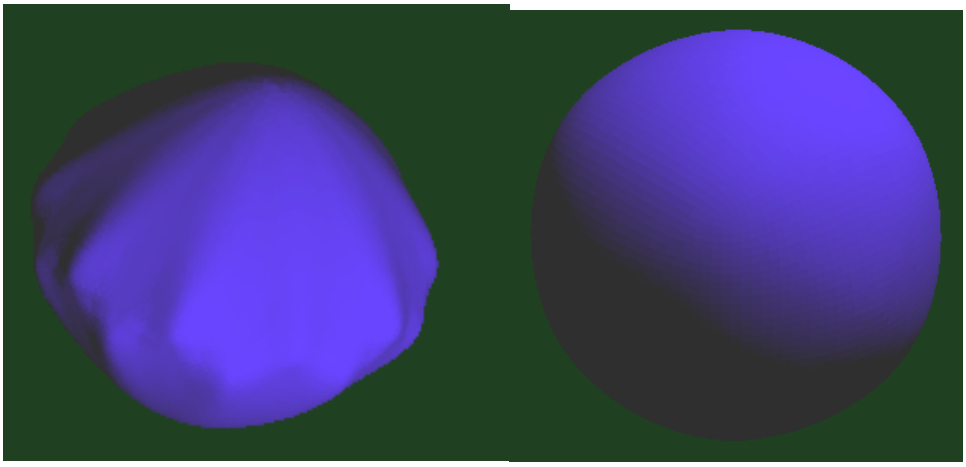


This figure has valence 10 vertices at top and bottom.



The high-valence nodes are more or less acceptable here—it's the valence 3 nodes that continue to look oddly pointed. Let's try an *ad hoc* change to our method: we'll allow a different ε to be used in the valence 3 case.

We implement a slider that allows us to modify the value of ε used on valence 3 nodes. We note that a slightly higher ϵ than our standard 1/16 gives a better visual appearance than the standard value: (1st picture the 20-sided figure; 2nd the cube. Note that ε is slightly higher in the second figure than in the 1st).



(Implementation was in Objective C, in Apple's Cocoa environment.)

(Shapes are hard-coded!)