

EECS 598-1: Supplement to PA3

Igor Guskov

March 12, 2002

1 Description of data structures

In this assignment we assume closed control meshes: that is there are no boundaries.

The edge-based mesh structures of the first programming assignment are used as the control mesh: the faces of these meshes do not have to be triangles but can be any polygons. The control mesh corresponds to level -1 of the hierarchy – this convention simplifies indexing for patches on higher levels. The level 0 of the subdivision mesh is obtained by introducing new vertices on faces and edges of the coarsest level control mesh – each control face and edge gets a new vertex. If the original face had n vertices then the zeroth level subdivision will produce n quads on that face – each of these quads will be associated to a directed edge in such a way that the origin of that directed edge is a corner of the quad. Thus each edge stores pointers to two patches, as shown in Figure 1. The quads will be refined during

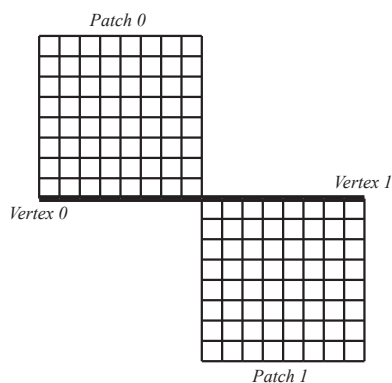


Figure 1: *Each edge is associated with two patches*

further subdivision and will be a square grid of $2^j \times 2^j$ points where j is the

level. The actual grid has a maximal level J and is allocated as an array of vertices that has $2^J \times 2^J$ vertices. This array can be accessed via $PatchT :: at(j, k_u, k_v)$ where j is a level index, (k_u, k_v) are indices within that level so that $0 \leq k_u \leq 2^j$ and $0 \leq k_v \leq 2^j$. The particular orientation of the grid is shown in Figure 2. In fact, $PatchT :: at(j, k_u, k_v)$ subsamples the finest

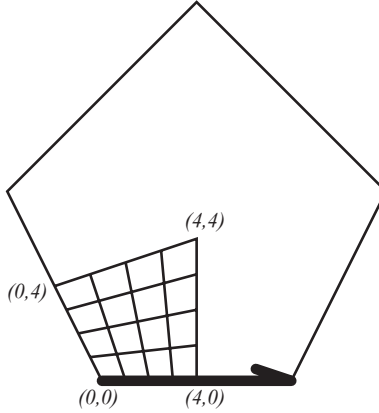


Figure 2: A patch associated to the bold directed edge and its indexing (k_u, k_v) on level 2.

level patch, and accesses a vertex in the square grid at $(2^{J-j}k_u, 2^{J-j}k_v)$.

2 Subdivision

The subdivision is implemented separately for the first step that takes an arbitrary closed polygonal mesh and produces quads at level 0 (see $MeshT :: SubdivideBase$), and the further steps that refine the quad mesh (see $MeshT :: SubdivideLevel$). In our description, a patch will mean a regular quad patch of the type described in the previous section, so that every control face with n corners will have exactly n patches.

The difficulties during subdivision arise from the fact that the patches are overlapping on the boundary. Thus a separate step is performed after each subdivision step that synchronizes all the vertices on the patch boundaries. This operation takes all the vertices that should coincide, averages their values and assigns the average value back to all the contributing vertices. Look into the procedure $MeshT :: SynchronizePatches$ for details.

3 Masks and synchronization

We should use special masks for certain vertex computations so that after the synchronization step the subdivision produces correct values.

In particular, the masks to be modified are the edge and vertex masks for new points on patch boundaries.

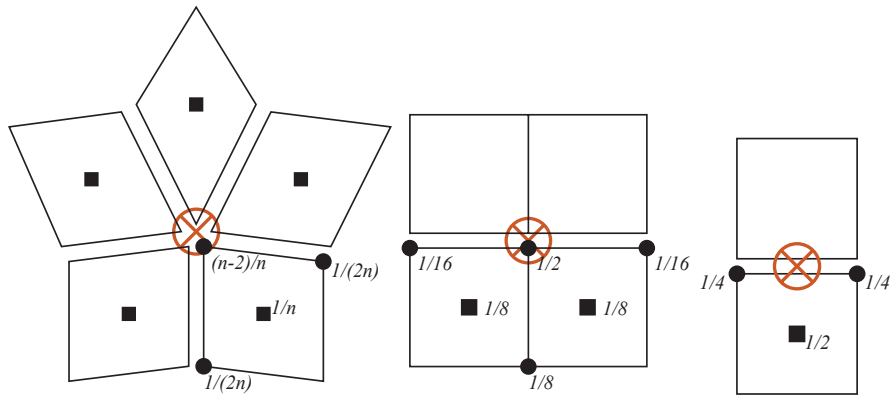


Figure 3: *Patch boundary masks: vertex patch corner, vertex patch side, edge patch side. The red circled cross denotes the vertex position being evaluated, black circles denote previous level vertices, black squares are face vertices on the current level.*

4 Miscellaneous

All the values on all the levels are stored in the same square grid of points so that when a vertex point gets updated to the value on the next level, one needs to store its old value so that other vertex points can get updated as well. If your computations proceed row by row within each patch it is enough to store one full row of the old values plus one additional old value on the current row.