CS417 Homework 5
background notes

April 5, 2003

This handout is to provide some background material that is needed for the problems in the homework but is not written down clearly elsewhere.

Subdivision

To make the interpretation of the subdivision masks more concrete, here is the one-step procedure that would be used to construct the fine mesh M_f from the coarse mesh M_c:

Function smooth-subdivide(Mesh M_c):
- Create empty mesh M_f
- For each vertex v_c in M_c:
  - Find all vertices adjacent to v_c
  - Let $n \leftarrow$ the valence of vertex v_c
  - Let $p \leftarrow$ weighted average of the positions of v_c and its neighbors
    - using weight β/n for each neighbor and and $(1-\beta)$ for v_c
  - Add a new vertex v_f to M_f with position p
- For each edge e_c in M_c:
  - Let $v_c^1, v_c^2 \leftarrow$ the endpoints of e_c
  - Let $v_c^3, v_c^4 \leftarrow$ the opposite vertices in the two triangles that share edge e_c
  - Let $p \leftarrow$ weighted average of the positions of $v_c^1, \ldots, v_c^4$
    - weighting v_c^1 and v_c^2 by 3 and v_c^3 and v_c^4 by 1
  - Add a new vertex v_f to M_f with position p
- For each triangle t_c in M_c:
  - Add four new triangles $t_f^1, \ldots, t_f^4$ to M_f using the vertices
    - that came from the vertices and edges of t_c
- Return M_f

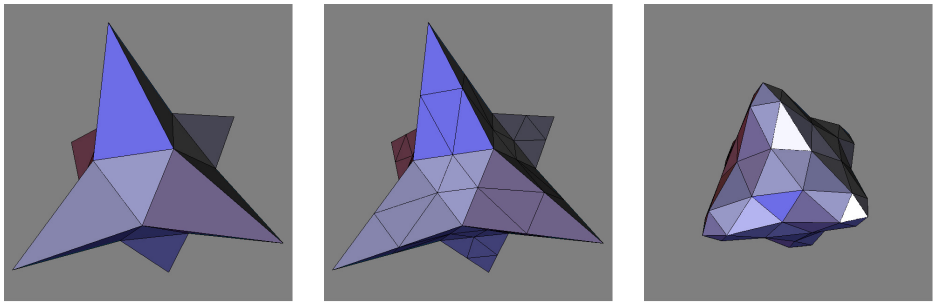

Figure 1: Computing a subdivision step by first refining, then smoothing.

And here is the two-pass procedure you are asked to work with:

Function subdivide(Mesh M_c):
    Create empty mesh M_f
    Copy all vertices from M_c to M_f.
    For each edge e_c in M_c:
        Let $v_c^1, v_c^2 \leftarrow$ the endpoints of e_c
        Let $p \leftarrow$ average of positions of v_c^1 and v_c^2
        Add a new vertex v_f to M_f with position p
    For each triangle t_c in M_c:
        Add four new triangles $t_f^1, \ldots, t_f^4$ to M_f using the vertices
            that came from the vertices and edges of t_c
    Return M_f
Function smooth(Mesh M):
    Create empty mesh M'
    For each vertex v in M:
        Find all vertices adjacent to v
        Let $p \leftarrow$ weighted average of the positions of v and its neighbors
            using weights that you define in your solution so this part
        Add a new vertex v' to M' with position p
    Copy all triangles from M to M' using the corresponding vertices
    Return M'

Figure 1 shows the two separate stages for a simple shape.

The goal is to define smoothing weights so that smooth-subdivide(M) computes the same result as smooth(subdivide(M)). It's not obvious that this is possible, since the single smoothing mask has to match the rules for two different subdivision masks. It is possible, however, for the particular masks that implement Loop subdivision.

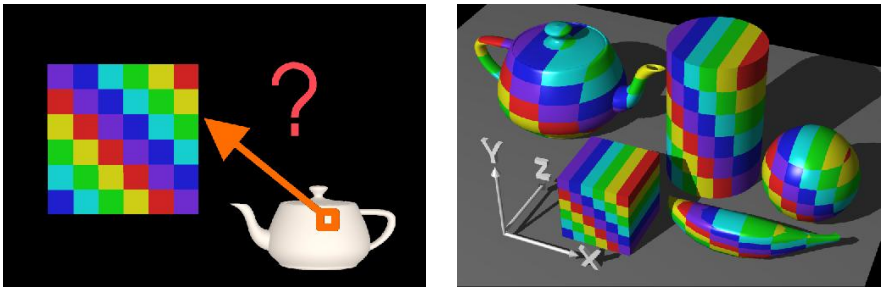

Figure 2: Defining texture coordinates by projecting into the x-y plane.

Texture mapping

The first part of the texture mapping problem has to do with defining texture coordinates for a 3D surface using the method of projection onto an intermediate shape. The primary usefulness of this method of defining texture coordinates is when you are texturing a surface that does not naturally "come with" a parameterization, and in that sense the problem is artificial, since the surface in the homework is already parameterized by the equations that define it, and it could be texture mapped successfully using that parameterization.

The first goal of defining texture coordinates on a surface is to make sure the mapping between the surface and the texture space is one-to-one everywhere that we care about putting texture on the surface. This is because in places where the mapping is many-to-one—that is, it maps a range of surface points to one texture point—it is not possible to completely control the texture on the surface, no matter what we put in the texture map. Similarly, if the mapping folds over on itself then near the fold the mapping is almost many-to-one, in the sense that there is some direction we can move on the surface that will have very little effect on the texture coordinates.

When the mapping function is smooth and continuous, as it is in this problem, these problematic points can be found by noting where the derivative of the mapping is singular. In the case of a mapping defined by projection onto a surface, the singularities are where the projection direction is parallel to the surface. Figure 2 illustrates this for the case of projecting onto the x-y plane (that is, throwing away the z coordinate). Note that when the surface is parallel to the z axis all that shows up on the surface is stripes, despite the fact that the texture map is a checkerboard-like pattern.

The previous paragraph gives both analytical and geometric criteria for points where the mapping will fail. It is possible to solve the problem using the analytical criterion, by writing out the derivative matrix (that is, the Jacobian) of the texture

coordinates (u, v) with respect to the parametric coordinates (s, t) and setting the determinant to zero. However, I find it much simpler for this problem to use the geometric criterion instead.

The second part of this problem also has to do with the derivative of a mapping to texture coordinates, but in this case the derivative is that of texture coordinates (u, v) with respect to screen coordinates (x, y). This derivative, a 2 by 2 matrix I will call J, is important when antialiasing textures because it gives us an approximation to the area in texture space that is covered by a particular pixel (this is the "footprint" of the pixel). In the setup described in the problem, the camera is sitting above an infinite checkerboard, which recedes to the horizon with the squares appearing smaller and smaller in the image. This means that the components of J get larger as y approaches zero (the horizon) and v increases (moving farther from the camera).

The partial derivatives that make up J tell us about what happens when we move slightly along the two axes in image space. The geometric interpretation of J, then, is that the two columns give you the vectors in (u, v) space that project to the axial unit vectors in the image. Even more concretely, if we assume that a pixel is a very small square, then it will project (approximately) to a little parallelogram in texture space, and the columns of J are the edges of that parallelogram. If we happen to be in a position where J is a diagonal matrix, we can also interpret the columns as the major and minor axes of the elliptical footprint of a circular pixel area.

The problem refers to the image coordinates as x and y because I am thinking of the projection plane as actually sitting in the scene at $z = -1$. But it may be clearer to think of the image plane coordinates as being separate from the world coordinates. If you like, go ahead and call the image coordinates (x', y'), the world coordinates (x, y, z), and the texture coordinates (u, v), and think of them as three separate spaces with mappings defined between them. Note that x' and y' each range from -1 to 1 across the image.