LSU EE 7700-1 **Homework 2** **Due: 19 February 2009**

As noted in class, the lighting of large triangles does not appear realistic when the light is closer to the interior of the triangle than a vertex. A realistic image would show a pool of light within the triangle, but all our code shows is nearly uniform lighting (based on the lighted color of the vertices).

Consider triangle $V_0V_1V_2$ illuminated by a light at L. Let P denote the point on the plane defined by $V_0V_1V_2$ which is closest to L. (Point P is the brightest spot on the triangle's plane.) The *first-order Vaidyanathan tessellation* of $V_0V_1V_2$ is either the set of three triangles PV_1V_2, V_0PV_2, and V_0V_1P, if P is inside $V_0V_1V_2$, or just $V_0V_1V_2$ (the original triangle) otherwise.

A first-order Vaidyanathan tessellation will partly solve the lighting problem. (Note: Dr. Vaidyanathan is not actually famous for this tessellation, he just suggested it to me.)

Problem 1: Modify hw2.cc so that when variable `opt-split-triangles` is `true` the first-order Vaidyanathan tessellation is applied to all triangles. (Applying means splitting the triangle only if the light is within it.)

- The 's' key is already set up to toggle the state of `opt-split-triangles`.
- Finding P is easy with a little knowledge of geometry and the pre-defined geometry code. The `sample-code` routine in the assignment file shows how to use the geometry classes and functions.
- The changes must be within the rendering pipeline code, not at the point where triangles are inserted into the vertex list.
- The changes must be at an appropriate place in the rendering pipeline code. The modified code must still operate by applying a small operation to a large number of elements. (That is, don't reorganize the code to process an entire triangle, from transformation to rasterization, in one step.)

Problem 2: Consider the performance implications of the first-order Vaidyanathan tessellation. There are two aspects to performance: determining whether a triangle should be tessellated, call this the *tessellation test*, and any additional work in processing the increased number of triangles.

(*a*) Estimate the performance impact of the tessellation test on the rendering pipeline up to, but not including, rasterization. Quantify performance by an operation count obtained by hand (number of floating-point operations, etc.).

(*b*) Compare your performance estimate to the render time provided by the frame buffer simulator. (Be sure to not run the -debug version of the code.) Comment on any differences.

(*c*) Suggest a way of reducing the computation needed by using a less compute-intensive method of deciding to not tessellate a triangle.

Problem 3: Assuming that the rasterization code has no way of determining whether a triangle was tessellated, estimate the performance impact of tessellating a triangle.

(*a*) Specify performance overhead of rasterization in terms of scene characteristics including the number of tessellated triangles plus something else. (That is, the overhead is not as simple as $3n_t$, where n_t is the number of tessellated triangles.)