

Assignment 4

CMSC 740, Spring 2002

Due: 12:30pm Thursday, May 9, 2002

This assignment involves extending Assignment 3 to accomplish better lighting and shading.

(a) Our AVW building model currently has really big quads and that is one of the reasons why the lighting is unsatisfactory. Build a quadtree decomposition of each quadrilateral by recursively subdividing it about its center. The recursive subdivision should terminate once the longest side of the quadrilateral is less than 100 units long. Store the quad-tree decomposition in a data structure associated with each polygon. (5)

(b) The reason the radiosity method is so expensive is because it solves the $n \times n$ system of linear equations which take higher-order inter-reflections among n patches into account. However, if we only consider the direct lighting from the visible light sources, we will get a reasonable lighting for our scene at a fraction of the cost for the full radiosity implementation. For this step you can use one of the following two methods:

- **Method A:** Shoot out $10K$ light rays from the center of each area light source in random directions. Use uniformly random values for θ and ϕ and then determine the direction to shoot the ray using the polar coordinates. For each light ray, find which leaf-level quadrilateral (in the quadtree decomposition) it intersects first and evaluate the illumination equation (ambient + diffuse) at the center of that quadrilateral. You will have to do this using hierarchical ray-shooting: start from the root of the quad-tree for each polygon and recurse down only if the ray intersects the parent.
- **Method B:** Render the scene with the light source as the viewpoint and the image plane a slight distance away from the viewpoint. Render each leaf-level quad in a distinct color using this method and read back the framebuffer to identify the visible quads. Use this list of visible quads to distribute the light source energy. Instead of using the hemi-cube method you can use the faster (but approximate) single-plane method. Refer [Recker *et al.* 90] pointer on the assignment web-page.

Add the contributions of multiple light rays falling on the same quadrilateral and store this as the *illuminated color* of the quadrilateral. At the end of the light-distribution phase above you can normalize the color by $\max(\max(R, G), B)$, where the max is computed over all the *illuminated colors*. Note the time it takes your code for the light distribution phase in your README file. (6)

(c) Implement a level-of-detail-based scheme where you use higher-detail quadrilaterals when the user gets close to them. Implement a metric that refines or coarsens each quadrilateral based on the distance to the viewer as well as the projected size of the quadrilateral (which can be approximated by the projected size of one of the quadrilateral diagonals). Use the *illuminated color* of each quadrilateral for display. To debug this part of the assignment you might wish to try using `GL_LINE_LOOP` instead of filled quadrilaterals. After debugging you should display the model with illuminated color per quad and no OpenGL lighting. (6)

(d) For each vertex, add the average illumination from its neighboring leaf-level quadrilaterals that are its siblings. Aliasing is a problem with both of the methods in part (b) above. This is caused because of insufficient sampling missing small quads which are then left unlighted. One way to fix this is to perform low-pass filtering over the quad hierarchy. In this scheme the *display color*, D , of each quad is computed as a geometrically-weighted sum of its ancestors. If for a quad j , the *illuminated color* is denoted by I_j , and its parent is represented by $P(j)$, then: $D_j = \frac{I_j}{2} + \frac{I_{P(j)}}{4} + \frac{I_{P(P(j))}}{8} + \dots$ This will antialias dark patches at the cost of some blurring of shadow boundaries. Display the quads with vertex colors as computed above with Gouraud shading and no OpenGL lighting. (3)