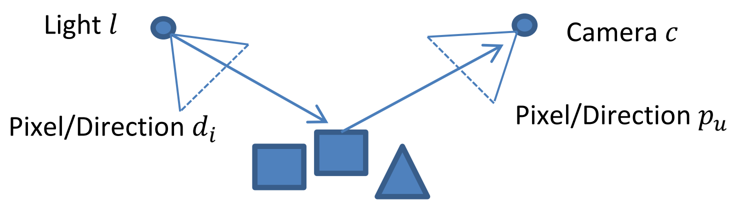
A2—Light Transport & Helmholtz Reciprocity

CS 43400, Spring 2013

Due on Wednesday February 27 at 7:00am

Summary: implement a program that computes the (synthetic) light transport of a scene and uses Helmholtz Reciprocity to swap the positions of the camera and light source.

1. **Create/define a rendering engine that produces basic diffuse scenes and optional non-diffuse effects, such as the reflections from the previous assignment.**
   a. Setup this project as an extension of your previous assignment or write a new rendering engine able to render a 3D scene and produce at least a simple diffuse (e.g., Lambertian) rendering --- you may choose the specifics of this rendering engine, but more details later in the assignment.
2. **Create sampled light transport matrix**
   a. You are to create a sampled light transport matrix T, using the aforementioned rendering engine, such that $c = Tl$ where c is a camera image and l is a light pattern.
   b. Choose a location for the single light source l and the single camera c; for example, they should be approximately 45-60 degrees from each other and the camera seeing 'one side of the scene' and the light illuminating the 'other side of the scene'.
   c. Render the light source as an array of small directional lights emanating from the light source position l and towards the scene, then bouncing off the scene and towards the camera. You can define the light directions d_{ij} by imagining a small plane (or curved surface) in front of the light position. Similarly can be done for the camera, with pixels p_{uv}. The parameterization should be roughly M by N steps for both the camera and light source planes. A directional light source can be defined as a ray from l and through d_{ij}. For example, in the following <u>2D case</u>, you see a light ray from l and through d_i. This light ray hits the scene somewhere and then bounces to the camera at pixel p_u (or p_{uv} in the full 3D case).



   d. To produce the light directions/pixels, you can use OpenGL spot lights with a very narrow cone or write a simple GPU program such mimics such a narrow spot light behavior. Roughly the light ray footprint on the scene surface should be such that it does not overlap with the light ray footprint of the adjacent light rays. This only needs to be approximate.

   e. To produce the camera pixels/directions, use the rendered image resulting from using the aforementioned narrow light ray. The camera pixels associated with the current light ray are those pixels whose value are greater than a small threshold (assuming black is the background/ambient color.
   f. Thus, as described in class, the camera image resulting from individually illuminating each light pixel corresponds to one column of the light transport matrix. For the basic requirement, the camera image can be grayscale. As mentioned later, supporting color is extra credit and simply entails concatenating the 3 color channels together in the image vector.
   g. NOTE: while you may implement a sparse-matrix or sparse-image data structure, it is sufficient to work at a low resolution such that the matrix T fits in memory. You may assume $M = N = 128$ (so 128x128 pixel camera/light images).

3. **Visualize Light Transport**
   a. The first test is to produce a <u>transport visualization</u> of the matrix T. The transport matrix is $16384x16384$ in size. Please conservatively subsample down to $4096x4096$ (e.g., compute the max of every group of adjacent $4x4$ pixels and make it the down-sampled value). Then convert the $4096x4096$ matrix to a standard grayscale image format normalized to the range $[0, 255]$. Please output either .jpg, .tif or any other standard image format.
4. **Rendering with Light Transport**
   a. Now, use the transport matrix to create novel scenes.
      i. <u>First</u>, use $l = [1]$ (a vector of ones) to create a camera image c. Please save the camera image to a standard image file format.
      ii. <u>Second</u>, use l as a checkerboard pattern (e.g., every 4x4 group of pixels is either black or white as in a checkerboard). Please save the camera image to a standard image file format.
      iii. <u>Third</u>, setup your program so that we can run it with our illumination image, and then save the camera image to a standard image file format.
   b. Next, please transpose the light transport matrix T and <u>repeat the same above output images</u>. The light source and camera should have effectively swapped position.
5. **Extra Credit**
   a. Support color: methodology is as described above
   b. Accelerate the creation process by, for example, parallelizing which light rays are used simultaneously in the same camera image.
   c. Make your rendering engine more complex than a simple diffuse engine: please note that the visualization of your transport matrix should reflect this.
6. **Turn in**
   a. Transport visualization image
   b. Output images for 2 light patterns (first and second)

c. Output images for 2 light patterns after transposing T.
d. A program for which we can specify the illumination image of size 128x128 stored as JPG or TIFF. It may be via a command line argument ("myprogram myilum.jpg") or via the GUI. In all cases, it should be very obvious. To produce the output using the transpose of the matrix, the command line argument should be similar to "myprogram myilum.jpg –transpose". If using the GUI, it should be clear as well.
e. Source files
f. Executable for program. It will be used for part 'd'. If you expect a command line argument and it is not given, please give an error indicating so. If you want us to put the filename using the GUI make sure it is obvious. Do not expect us to know what other magical parameters to use. The produced camera image should be put in the current directory with the name "output.jpg" or similar.
g. Extra credit as described above, with appropriate extra image files
h. Turn in a single zipped archive with all your files via Blackboard