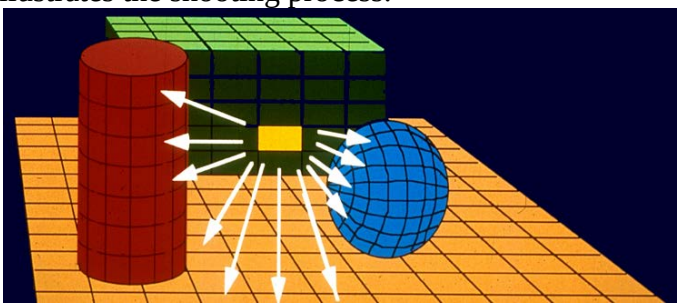
COP 3503 - Programming Assignment #3
Light Transport Computation using Shooting Algorithm
Total: *6 points.*
Assigned: Jan 31, 2012 (Tuesday)
Due: February 09, 2012 (Thursday) at 11:55 PM WebCourses time

Objective

Create a Graph Data Structure, use indirect binary heap Priority Queue data structure to efficiently selecting and updating graph vertices according to priority.

Light Transport Computation by Shooting: One of the methods of realistically lighting a synthetic scene is by shooting light in a scene. In this method, surfaces of the synthetic scene are divided into patches. The shooting algorithm then proceeds iteratively by choosing the brightest patch in the scene and shooting light from that patch to all other patches. In the beginning emitting surfaces are brightest surfaces. But as shooting progresses, some of the non-emitting surfaces receiving light become bright, and light is shot from them. The processing continues till equilibrium is reached (i.e. brightness of surfaces do not change anymore). Synthetic scenes can have thousands of surface patches. Not every patch in the scene is visible to all patches in a scene. So a surface patch emits towards only a few other patches. It is important that at the begin of every shooting process, the brightest patch in the scene is found efficiently. So, a priority queue is often used to find the brightest of all patches.
The figure below illustrates the shooting process.



The pseudo code below outlines the shooting algorithm.

```
--(1)--Create a weighted graph with surface patches as vertices,
visibility between patches as edges, and formFactor as weights of
the edges.
//Initialize
for (i=0; i< N; i++)
  patch[i].totalPower=patch[i].unshotPower=patch[i].emissionPower
--(1)--organize unshotPower in a priority queue---
// Shooting: Iterative computation
while (!converged) do
{
  --(2)--find patch k such that its unshotPower is the maximum---
```

```
  //Shoot unshotPower to all adjacent patches
  for every adjacent patch i of k
  {
      patch[i].totalPower += patch[i].reflectance*
                formFactor(k->i)* patch[i].unshotPower
      patch[i].unshotPower += patch[i].reflectance*
                formFactor(k->i)* patch[k].unshotPower
      //--3–Restore priority queue property
  }
  patch[k].unshotPower=0.0
  // --4–Restore priority queue property as in –3-
}
```

Problem:

In this assignment, you are required to organize unshotPower in a priority queue (binary max heap) such that you are able to,

- find in O(1) time the patch index that has max unshotPower,
- restore in log N time the heap property after every update of unshotPower.

You will assume the following:

N is the number of surface patches. (input)

Each patch has a patchId from 1 to N and has the following properties:

- reflectance: a fraction between 0 and 1 (input)
- emissionPower: 0 for most surface except for a few emitters. (input)
- number of visible patches (input)
- list of visible patches (edges) and form-factor (weight) (assembled from input).
- totalPower (computed during iteration)
- unshotPower (computed during iteration)

M is the number of iterations required for Light distribution to converge. (Convergence checking requires more involved checks inside the iteration look. For this assignment we will assume that M number of iterations is sufficient.)

Instruction

- Use indirect binary max heap to implement the priority queue. Write three functions: buildIndirectHeap, increasePatchValue and decreasePatchValue
  - buildIndirectHeap: Works on the unshotPower values of the patches (may or may not accept the graph as input)
  - increasePatchValue/decreasePatchValue: Takes patchId and the newUnshotPower as input, updates the value of the patch, and reestablishes the maxheap property.
- EmissionPower: It is the power leaving the surface patch of the emitters. Assume it to be a random value less than 100. Assume that there are L << N number of emitting paches in the scene.

  If you want to create sample data you may initialize the emission as follows.

  for (i = 0; i <L; i++){

Randomly choose a patch index between 1 to N. Let it be "k".
Randomly choose a fraction "f" between 0..100
and set patch[N].emission = f

- Reflectance: It is a fraction of incident light reflected from a surface patch. Reflectance must be ≥ 0 and ≤ 1. It is a material property and is generally assigned by the scene modeler.
  If you want to create sample data, you may assign it by drawing a random number between 0 and 1.
- formFactor(k->i): It is the fraction of light emitted from patch "k" that reaches patch "i". It is a geometric factor and is computed using geometric relation between patches. Because of its definitions the following relation must be true:
  $\sum_{i=0}^{N} \text{formFactor}(k \rightarrow i) \leq 1.0$.

  For creating sample formfactor data, you may choose to do a random assignment, but guarantee that the given relation is satisfied. In a real-scene objects obstruct other objects. So not all patches receive light emitted from a patch. So you must find out all those patches that will receive light from a patch "k". Here, you will choose a random number between 1 to N to decide on the number of patches that will receive light from patch "k", and also choose each of those patches randomly. Basically you will do something like the following:

```
for every patch k
        choose number of visible patches j : a random number between 1.. N
        sumf = 0.0;
        for m = 1 to j{
                Randomly choose a patch index between 0 to N-1.  Let it be "i".
                Randomly choose a fraction "f" between 0..1
                set formFactor(k->i) = f
                sumf += f
        }
        for every visible patch i  from patch k
                formFactor(k->i) /= sumf
}
```

  Note: If you are creating sample input data then
  - Use call "random(seed)" after reading in the "seed" from input file and before setting up reflectance, emissionPower, formFactor etc...
  - Use "nextFloat()" to get a random float number, "nextInt(n)" to get a random integer between 0 and n-1. If you wish to get a random number between 1 to n then use "nextInt(n)+1".

Input

The program must accept input from file **Assignment03.txt** with following information:

1st line: <N: the number of patches> <M: the number of iterations>
Followed by multiple inputs for each patch:
<patchId> <emissionPower> <reflectance> <# visible patches>
< visiblePatchId> <formfactor> // one line per visible patche

Output

- For every patch:
  - Print its emission power, and reflectance (in one line)
- For every patch:
  - Print total power after completion of the iteration. (in a line)
  - Print the list (patchId of visible patch, formfactor). (in a line)

Deliverables

You must submit the source code (The .java files, not the .class files) for your programs, and report over WebCourses by 11:55 PM on Thursday, February 09, 2012. You must upload your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will not be accepted.
The package must contain:
- Assignment03.java, the main java files and any additional for your assignment.
- Assignment03.pdf: the writing component of your assignment. This document must contain three sections.

1. Pseudo-code for indirect Binary Heap construction, indirect Binary Heap Update
   a. buildIndirectHeap
   b. increasePatchValue (called every time the patch unshotPower is increased.)
   c. decreasePatchValue (called every time the patch unshotPower is set to zero.)
2. Status of the programming Assignment: Complete, incomplete,.
3. Output for the sample input.

Restrictions

- Follow posted assignment submission guideline.
- Your program must accept the input file (**Assignment03.txt**) from the same directory as where the Java source file is placed. NOTE: Do not code any absolute directory path.

Grading Policy: Total 6 points

You program must compile and run. (0 credit for code not compiling or not running.)
This submission will be graded from a total of 6 points using the following criteria:

- Read and create the graph, setup and initialize data (emissionPower, reflectance, form-factor, totalPower, unshotPower) as per the instruction. (1 point)
- Correctly for the shooting algorithm (includes adjacency list traversal). (1 point)
- Create indirect heap
  - build heap with unshotPower data (1 points)
  - update the heap order after every change in unshotPower (1 points)
- Successful completion of iteration till convergence. (0.5 points)
- Correctly print totalPower. (0.5 points)
- Report: (1.0 points)