Math 155B – Topics in Computer Graphics – Spring 2017

Project #3– Implement distributed ray tracing

Due date: Friday, May 12, 9:00pm.

Hand in: (a) A PDF file with images from you project, and a short description of your work. (b) On the computer lab computers, save high quality bitmap files of your scene: with only basic ray tracing and with distributed ray tracing. (c) An individual grading session with your professor or TA.

Overview: For this assignment, you will modify the **RayTrace** code to include distributed ray tracing. Your program should support the following:

1. Your scene should support at least two forms of distributed ray tracing. You should choose two from the following list:
    a. Stochastic supersampling,
    b. Depth of field,
    c. Motion blur,
    d. Extended lights/soft shadows.
2. Your scene should use **both transmission and reflection**, to a trace depth of four or five. (Suggestion: do the initial development and debugging at a trace depth of three, for faster performance.)
3. You should use jittering and stochastic supersampling whenever appropriate.
4. Create images with a copy of your scene twice: First, rendered with only basic ray tracing and without distributed ray tracing, and second, rendered with distributed ray tracing. Be sure to save high quality bitmap (.bmp) files images of these. You should also include them in the PDF, but be aware that the PDF format will compress the images, and may make it harder to see the special effects of distributed ray tracing.
5. Your scene is recommended to be midway in complexity between the simple scene of "RayTrace" and the scene on the cover of the textbook ("RayTrace2"). **It does not need to be more any more complex than is needed to illustrate the ray tracing effects.** You may use a scene similar to one of the two supplied scenes, or you may make up your own scene, or incorporate your Bezier curves from the previous project, or you may can set up a billiard ball image as shown in some the textbook images, etc. (ask Professor Buss for help in setting up checkerboard texture maps). If you create a scene with more than a couple dozen objects consider learning how to use the kd-tree version of the RayTrace software, which will probably run faster.
6. Keep the keyboard controls of the existing RayTrace software to allow changing the viewpoint. (If you wish, you may disable the OpenGL rendering option: you will possibly find this annoying more than helpful.)

Suggestions:

1. Download the latest version of the RayTrace software from

    http://www.math.ucsd.edu/~sbuss/CourseWeb/Math155B_2017Spring/Project3/RayTrace_155B_S17.zip

This version is better than the version available at the textbook's webpage! (Which currently is broken, but will get fixed.)

2. For instructions on using the software: Please see: the appendix in the textbook, and the web page at http://www.math.ucsd.edu/~sbuss/MathCG/RayTrace/raytrace_3.0_intro.html. We will also discuss this in class.
3. If you have difficulties with the code, please talk to Professor Buss or TA Kuchihotla as soon as possible. Any bugs, or other difficulties, should be reported to Professor Buss, as the software may be updated upon request. (It is hoped to post a new release of the RayTrace software by the end of the course.)
4. When you download the RayTrace software,
    a. Open the "Solution" file (extension .sln") in the RayTrace folder.
    b. This has a total of eight projects. Make sure that one of **RayTrace**, **RayTrace2** or **RayTraceKd** is the "startup project". Do this by right clicking on the project name.
    c. Find out how to switch between "Debug" and Release". Debug mode is best for development and debugging. Release mode will run 20 times faster. (!) The ray tracing will finish quicker if you keep the size of the window small --- so there are few pixels to calculate.
    d. **RayTrace** makes a simple scene with a transparent sphere, a checkboard terture, and some triangles. **RayTrace2** makes the scene on the cover of the textbook. **RayTraceKd** supports kd-tree acceleration, and can either draw the scene on the front of the textbook, or load scenes from .obj or .nff files. (No textures or material properties in these files, unfortunately.) You may use any of these as the basis for your project. but **RayTrace2** may be the best version for you, as it has examples of all the features of the ray tracer, and its scene can be simplified down to a scene you wish to use for your project.
5. The program starts by drawing the scene in OpenGL, not raytraced. Press the "g" command or the space bar to initiate raytracing. Depending on how large your window is, how many objects there are, and how many rays are being spawned by the distributed ray tracing, it might take a few seconds to a few minutes or even longer to generate the ray traced image. (So be patient!) Again, debug using a very small window initially.
6. **For stochastic supersampling:** Note that **MainView->CalcPixelPosition(** *i*, *j* **)** can take floats as arguments: this lets you get sub-pixel locations easily.
7. **For Jittered eye positions:** The first principle is that you should set up the camera position **MainView** once and do not change the position of the viewer or the screen pixels afterwards. To jitter the eye position, you should do the following.
    a. First, get the position of the current pixel by calling **MainView->CalcPixelPosition(** i, j **);** Let's call the returned pixel position p. (Again, note that i and j may be floating point numbers.
    b. Second, get the position of the viewer. This is the position you gave to **MainView** when you initialized it, or you may get it again by calling **MainView->GetCameraPosition()**. Let's call this position **c**. ("**c**" for "camera").
    c. Third, get the "right" and "up" directions, which I will call **u** and **v**, by calling the routines **MainView->GetPixeldU()** and **MainView->GetPixeldV()**. These values are the vector pointing rightward from a pixel in the screen to its immediate neighboring pixels, and the vector pointing upward from a pixel to the pixel above. You might wish to normalize these vectors (depends on how you use them in the next step). These **u** and **v** vectors are fixed, so they can be obtained ahead of time rather than recomputed for each ray from the eye.
    d. Choosing appropriating, jittered values α and β let the ray start at $\mathbf{C} := \mathbf{c} + \alpha\mathbf{u} + \beta\mathbf{v}$ and have direction **p-C**. You should normalize this direction vector.
8. **For extended lights:** You will need to write your own code to determine target positions on extended lights. You can choose the points by using stochastic sampling of an angle theta and a

radius *r*. To get an equal area (unbiased) sampling make the probability of choosing a particular value for *r* proportional to *r*. We will discuss this in class lecture.

9. **For motion blur,** update the position and orientations of the moving objects each time you start tracing a new ray from the eye. (If you use the kd-tree representation, please note that the kd-tree does not accommodate moving objects.)

Hand in procedure: One-one grading PLUS submit a PDF file PLUS save good quality .bmp files of your scene rendered with only basic raytracing and with distributed raytracing..

(1) As usual, create a directory called **Project3** in the **Documents** folder your networked file system on the PC computers. (not the desktop). Place there your source files and your Visual C++ solution and project files. Do not modify these after turning in the programs, so as to leave the file dates unmodified. Your program should compile and run in your work directory with the version of Visual C++ installed on the computer lab computers. If you use another version of C++ at home for development, it is your responsibility to convert your project files to work on the computer lab computers.

(2) Create a one or two page PDF file (preferably one page, **at most** three pages!) describing your project. For this, include a short description of your project, and at least one image. This will be uploaded for Professor Buss, so he can see all projects from the class. By default, the PDF files will be made available for other students to see in this class and possibly future classes, so include your name, but not your PID. (Please let me know if you prefer that your report not be included.)

(3) Create two bitmap files as instructed above. Save these in the PC computers.

There will be piazza instructions on how to submit the PDF files by upload to dropbox.

Grading: Grading is an individual session with Srivastava or Professor Buss, and on evaluation of your PDF submission. Please do not modify your files after the due date.