CS4247
Graphics Rendering Techniques

Semester 2, 2018/2019

Assignment 3

Progressive Refinement Radiosity

School *of* Computing
National University *of* Singapore

Dates

- Release Date
  - 29 March 2019, Friday

- **Submission Deadline**
  - **19 April 2019, Friday, 11:59 PM**

  - Late submissions will NOT be accepted
  - The submission folder in the IVLE Workbin will automatically close at the deadline

Assignment Overview

- You are provided with an incomplete C/C++ program that implements the **progressive refinement radiosity algorithm**

- Task #1
  - Complete the program
  - Generate a radiosity solution for the sample input scene
- Task #2
  - Create a new input scene model
  - Generate a radiosity solution for the new scene

Learning Objectives

- Implementing the Progressive Refinement Radiosity Algorithm
- After completing the assignment, you should have learned
  - How to use the Hemicube algorithm to estimate form factors:
    - How to compute delta form factor for each pixel on the hemicube
    - How to set up OpenGL views to project the scene onto the faces of the hemicube
    - How to use the item buffering technique to identify the patch that occupies a pixel
  - How the progressive refinement radiosity algorithm works:
    - How to “shoot” light power from a shooter patch to the gatherer patches, and update the radiosity values of these patches
    - How to update the unshot power of a shooter patch with the new power received by its child gatherer patches
    - How to terminate the progressive refinement radiosity computation

What Are Provided (1)

- Download the file **`cs4247_assign3_2019_todo.zip`** from the **`Assignments`** IVLE Workbin folder
- The ZIP file contains a Visual Studio 2008 solution file **`assign3.sln`**. The solution has three C/C++ projects
  - QuadsViewer
  - RadiositySolver
  - RadiosityViewer
- **Tip:** In Visual Studio, to make a project as the default project to be built and run, you can right-click on the project name in the **Solution Explorer**, and select “Set as StartUp Project”

What Are Provided (2)

- QuadsViewer is a completed OpenGL application that lets you preview the input scene model and check the subdivision of the input quads into smaller “shooter” quads and even-smaller “gatherer” quads
  - Gatherer quads are obtained by subdividing shooter quads
  - Press “m” to cycle through display of the input quads, the shooter quads, and the gatherer quads
  - In **`quadsviewer.cpp`**, the size of the shooter and gatherer quads are controlled by the constants **`maxShooterQuadEdgeLength`** and **`maxGathererQuadEdgeLength`** respectively
    - **`radiositysolver.cpp`** also has the same constants for the same purpose

What Are Provided (3)

- RadiositySolver is the incomplete program that
  - Reads the input scene model
  - Subdivides the input quads into "shooter" and "gatherer" quads
    - In **`radiositysolver.cpp`**, the size of the shooter and gatherer quads are controlled by the constants **`maxShooterQuadEdgeLength`** and **`maxGathererQuadEdgeLength`** respectively
  - **Computes a radiosity solution for the scene**
    - **This step is the only part not implemented yet**
  - Computes vertex radiosities from the patch radiosities
    - This step can take very long to run, so you should first test your radiosity algorithm implementation with a model with not too many gatherer quads
  - Outputs the model with radiosity solution

What Are Provided (4)

- RadiosityViewer is a completed OpenGL application that lets you view the model output by **RadiositySolver**
  - Reads in a model with radiosity solution
  - Performs simple tonemapping to map radiosity values to displayable color values (i.e. to R, G, B values from 0.0 to 1.0)
  - Renders the polygons using the tonemapped radiosity values as vertex colors
  - You can try the viewer on the given sample model file **`cornell_box.out`**
    - Note that the sample **`cornell_box.out`** has been "watermarked" with some bright and dark patches — your radiosity solution for **`cornell_box.in`** should not have those

Task #1

- Complete only the source file **`radiositysolver.cpp`**, which is part of the **RadiositySolver** project
  - Complete the code at places marked "**`WRITE YOUR CODE HERE`**"
  - You can add additional functions to the file
  - Use good coding style and document your code adequately (otherwise marks deducted)
  - Study the files **`quadmodel.{h,cpp}`** to see how the scene model and its subdivided quads are represented
  - You can make use of helper functions found in **`common.h`** and **`vector3.h`**

Task #1 (continue)

- Test your program on the provided sample input scene model **`cornell_box.in`**
  - Name your output file **`cornell_box_my.out`**
- View **`cornell_box_my.out`** in **RadiosityViewer** (in non-wireframe mode) and capture three different snapshots
  - On Windows, you can use the **Snipping Tool** to take a snapshot of the window and save the image
    - Or you can press **Alt + Prnt Scrn** to capture a snapshot of the current active window, and then save the image in a image editing software tool (e.g. Paint)
  - Use the default window size
  - Save your snapshots to image files
    - **`cornell_box_1.png`**
`cornell_box_2.png`
`cornell_box_3.png`

Task #1 — RadiositySolver Explained

- **RadiositySolver** uses to the Progressive Refinement Radiosity algorithm to compute patch radiosity of each gatherer quad
  - Pre-compute the delta form factors on the top and side faces of the hemicube
    - Top hemicube face has same pixel resolution as the default window size (always a square)
  - Progressive Refinement Radiosity loop
    - Find the shooter quad with the greatest unshot power
    - Use a hemicube to compute form factors from the shooter quad to each of the gatherer quads
    - Update the radiosity of each of the gatherer quads
    - Update the unshot power of each shooter patch with the new power received by its child gatherer patches
    - Terminate loop if max iterations is reached or the greatest unshot power is below a threshold

Task #2

- Create a new scene model
  - Study the sample scene model **`cornell_box.in`** to find out the input model file format
  - Name your new scene **`new_model.in`**
  - Run your **RadiositySolver** program on **`new_model.in`** to output **`new_model.out`**
  - View **`new_model.out`** in **RadiosityViewer** and capture <u>three different snapshots</u>
    - Use the default window size
    - Save your snapshots to image files
      - **`new_model_1.png`**
`new_model_2.png`
`new_model_3.png`

What to Submit

- Only the following **10** files
  - Task #1
    - `radiositysolver.cpp`
    - `cornell_box_my.out`
    - `cornell_box_1.png`, `cornell_box_2.png`, `cornell_box_3.png`
  - Task #2
    - `new_model.in`
    - `new_model.out`
    - `new_model_1.png`, `new_model_2.png`, `new_model_3.png`

How to Submit

- Package <u>only the required</u> files in a single ZIP file
- Name your ZIP file **`<matric_no.>_assign3.zip`**
  - e.g. **`A0123456X_assign3.zip`**
- Upload ZIP file to IVLE Workbin folder "**<u>`Assignment 3 Submission`</u>**"
  - Folder will close at the deadline
  - You may upload your ZIP file multiple times, but we take the latest
  - Please delete your old submissions
  - Your filename may be automatically appended with a number. Don't worry

Other Requirements

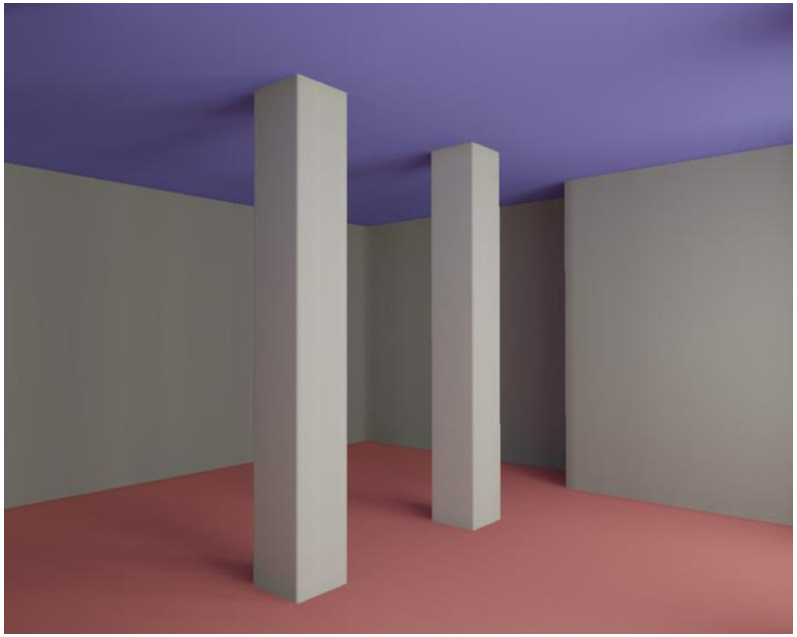
- Programming languages and APIs
  - C / C++
  - OpenGL (won't accept other graphics APIs)
  - GLUT
  - No other third-party APIs are allowed
- Platform
  - You can develop your program on other OS platforms and IDE
  - However, the final submitted version must be able to compile in Microsoft Visual Studio 2008 and later

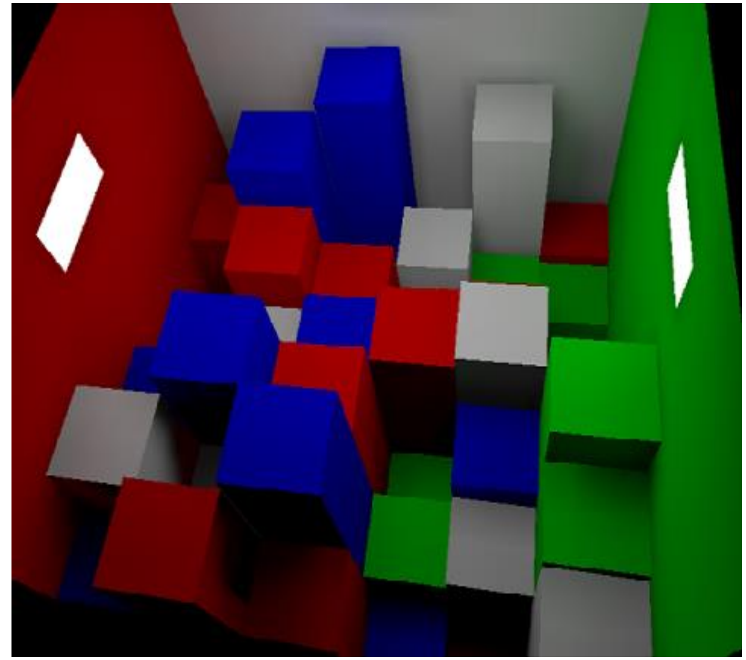
Grading

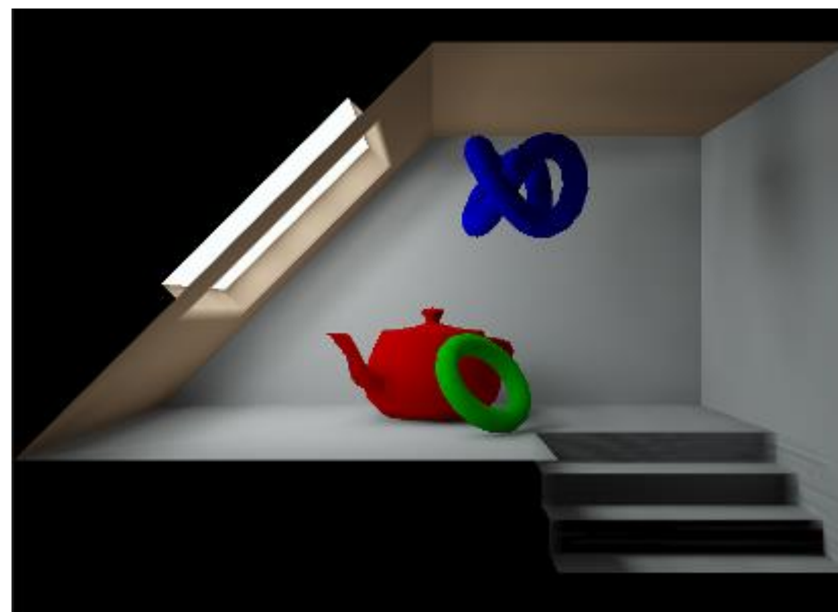
- Maximum marks is **100**
- Constitutes **10%** of total marks for CS4247

- Marks allocation
  - **Task #1 – 80 marks**
    - Correctness & Coding Style
  - **Task #2 – 20 marks**
    - Task completion (10 marks)
    - Aesthetics and complexity (10 marks)

Past Submissions — Task #2







The End