

# **CS4247**

# **Graphics Rendering Techniques**

Semester 2, 2023/2024

## **Assignment 4**

## **Progressive Refinement Radiosity**

**School of Computing**  
**National University of Singapore**

# Dates

- Release Date

- 8 April 2024, Monday

- Submission Deadline

- 26 April 2024, Friday, 11:59 PM

- Late submissions will NOT be accepted
    - The submission folder 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) – For Windows

- Download the file `cs4247_2324S2_assign4_todo_(win-vs2017).zip` from Canvas > CS4247 > Files > Assignments folder
- The ZIP file contains a Visual Studio 2017 solution file `assign4.sln`. The solution has **three** C/C++ projects
  - QuadsViewer
  - RadiositySolver
  - RadiosityViewer
- **Tip:** In Visual Studio, to make a project 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 (1) – For macOS

- Download the file `cs4247_2324S2_assign4_todo_(mac-xcode).zip` from Canvas > CS4247 > Files > Assignments folder
- The ZIP file contains the Xcode project `assign4.xcodeproj`, which has the following three C/C++ targets
  - QuadsViewer
  - RadiositySolver
  - RadiosityViewer

# What Are Provided (2)

- **QuadsViewer** is a completed (old-style) 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
  - The default input scene model file is **model.in**
  - **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 **quadviewer.cpp**, the size of the shooter and gatherer quads are controlled by the values of **maxShooterQuadEdgeLength** and **maxGathererQuadEdgeLength** respectively
    - These two values are from the input scene model file **model.in**
    - **radiositysolver.cpp** also uses these two values for the same purpose

# What Are Provided (3)

- **RadiositySolver** is the incomplete program that
  - Reads the input scene model from **model.in**
  - Subdivides the input quads into “shooter” and “gatherer” quads
    - In **radiositysolver.cpp**, the size of the shooter and gatherer quads are controlled by the values of **maxShooterQuadEdgeLength** and **maxGathererQuadEdgeLength** respectively
      - These two values are from the input scene model file **model.in**
  - Computes a radiosity solution for the scene
    - This step is the part to be completed
  - 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 to the file **model.out**



# What Are Provided (4)

- **RadiosityViewer** is a completed (old-style) OpenGL application that lets you view the model output by **RadiositySolver**
  - Reads in a model with radiosity solution from file **model.out**
  - 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**
    - First, copy **cornell\_box.out** to the file **model.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 `quadmodel1.{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**
  - **Before** running your RadiositySolver, copy **cornell\_box.in** to the file **model.in**
  - **After** running your RadiositySolver, copy the output file **model.out** to **cornell\_box\_my.out**
- View your radiosity solution **model.out** using **RadiosityViewer** (in non-wireframe mode) and capture three different snapshots
  - On Windows 10/11, you can use the **Snip & Sketch** tool (press Win + Shift + “S”) to take a snapshot of the window and save the image
  - Use the default window size
  - Save the **three** 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 face** (already done) and **side faces** of the hemicube
    - Note that 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

# 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 three different snapshots
    - 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

- Put only the required files in a single ZIP file
- Name your ZIP file *your-student-number\_assign4.zip*
  - e.g. A0123456X\_assign4.zip
- Submit your ZIP file to Canvas > CS4247 > Assignments > Assignment 4
  - Assignment submission will close at the deadline
  - You may upload your ZIP file multiple times, but we take the latest

# Other Requirements

## ■ Programming languages and APIs

- C / C++
- OpenGL 1.1 (old-style OpenGL; no shader)
- GLU & GLUT / FreeGLUT
- No other third-party APIs are allowed

## ■ Platform

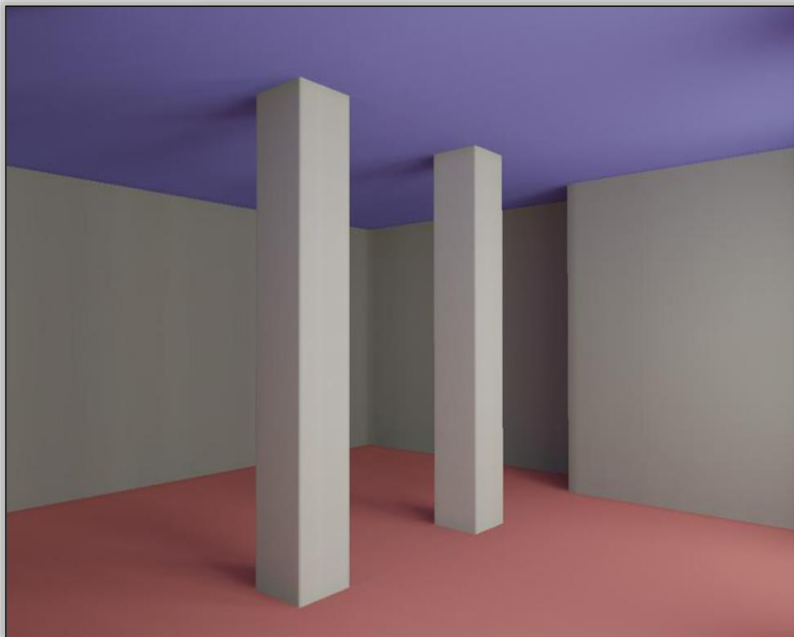
- You can develop your program on any OS and IDE
- However, the final submitted version must be compilable/buildable in Microsoft Visual Studio 2017

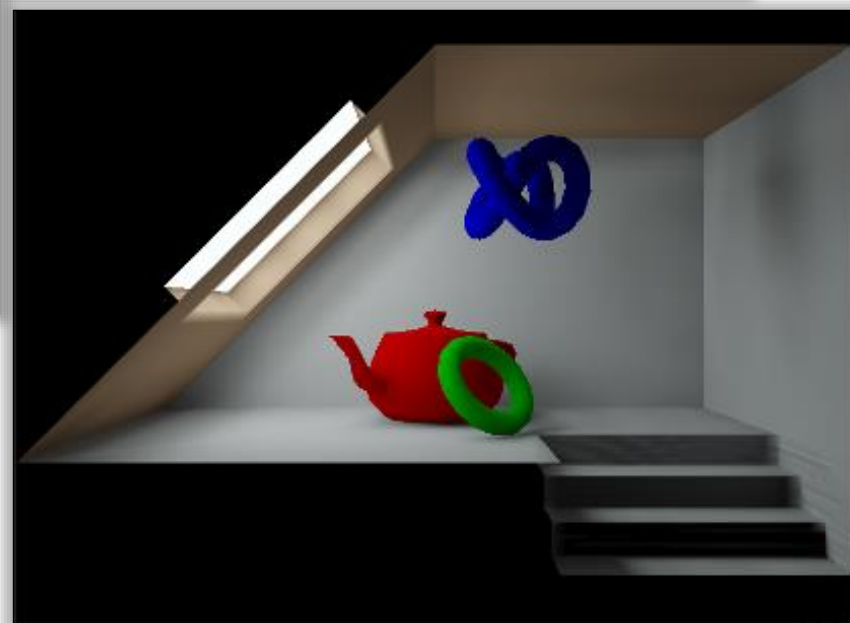
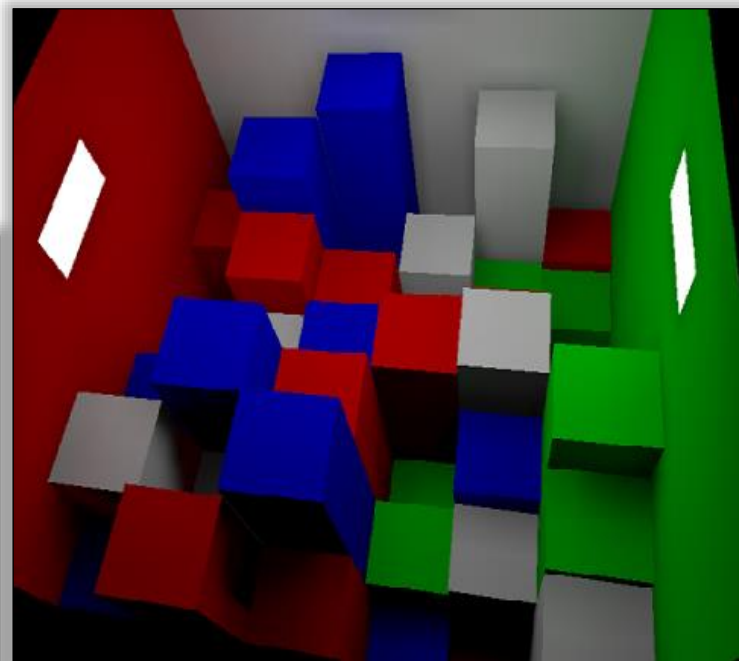


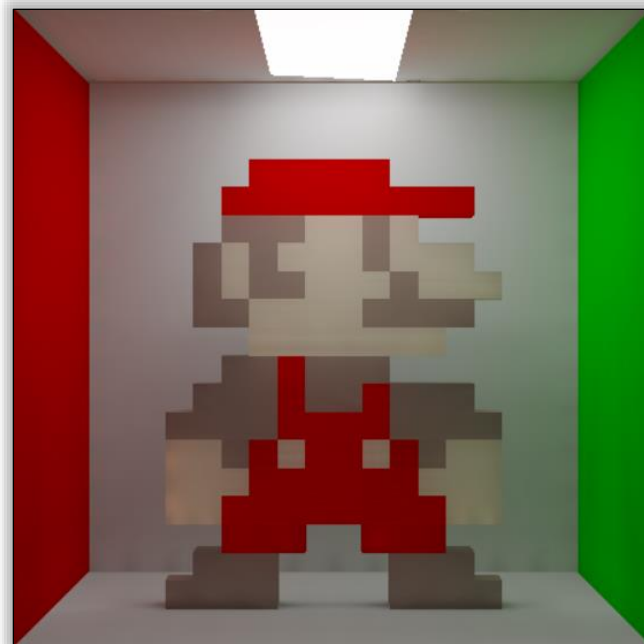
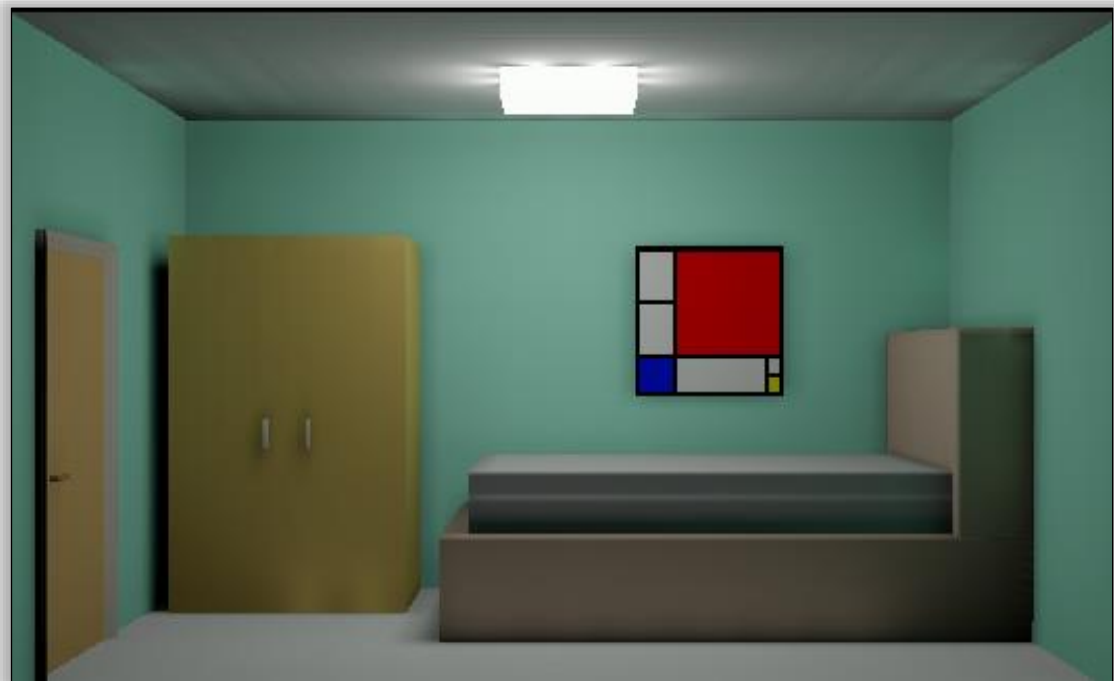
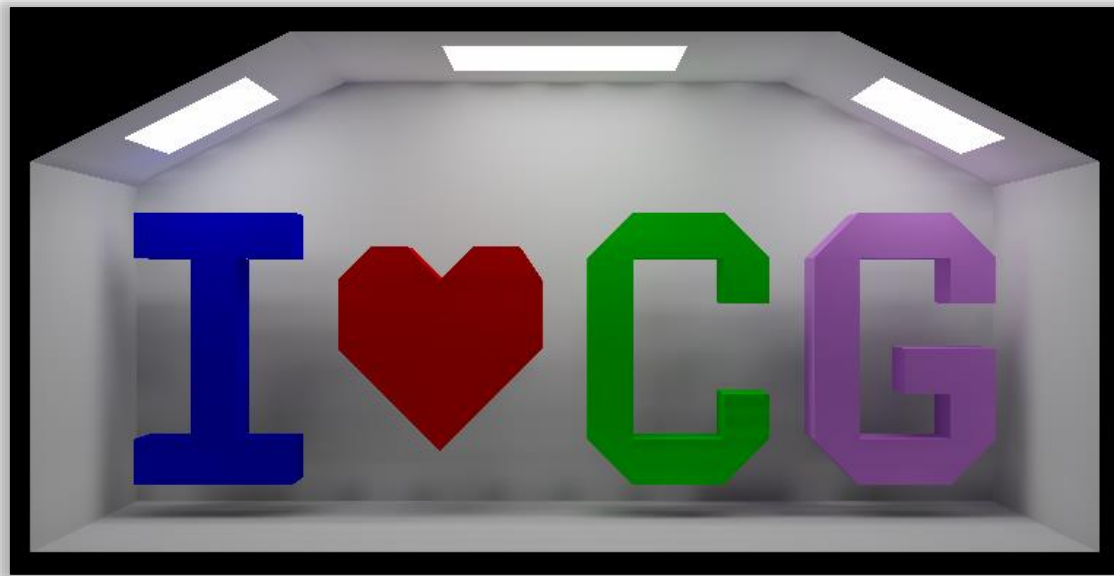
# Grading

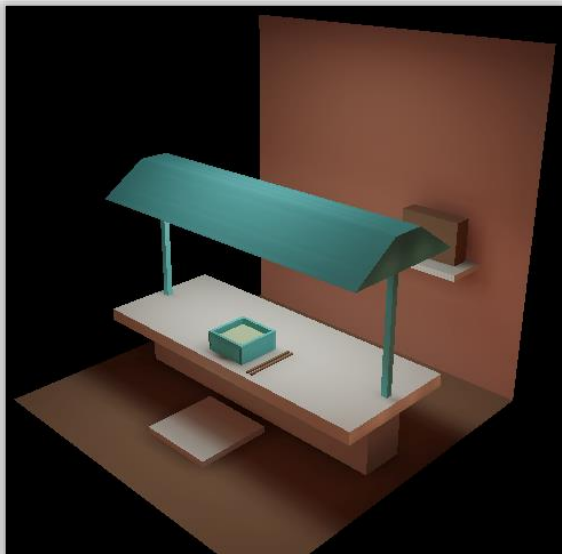
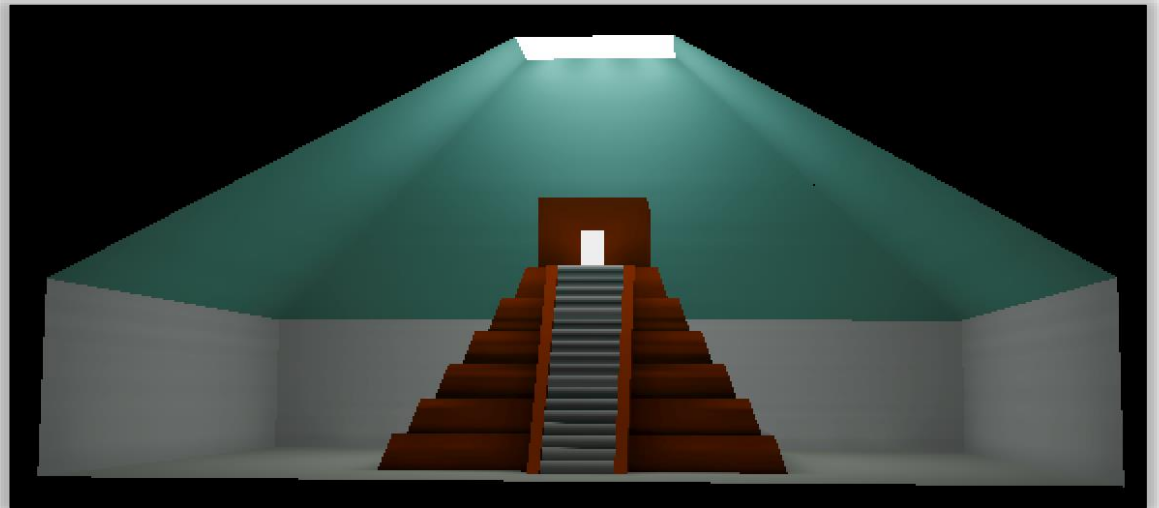
- Maximum marks is 100
- Constitutes 9% of total marks for CS4247
- Marks allocation
  - Task 1 – 80 marks
    - Correctness & Coding Style
  - Task 2 – 20 marks
    - Task completion (10 marks)
    - Aesthetics and complexity (10 marks)

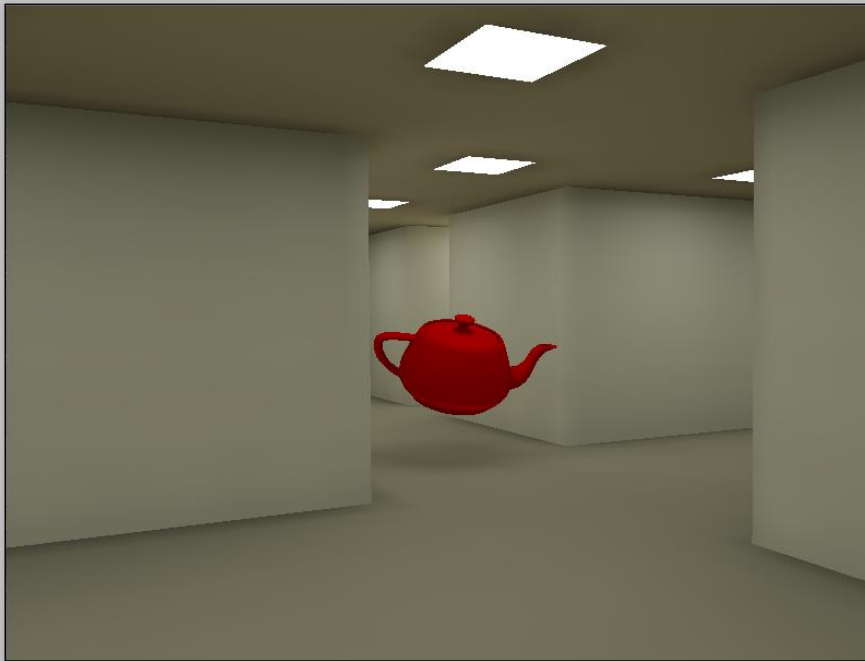
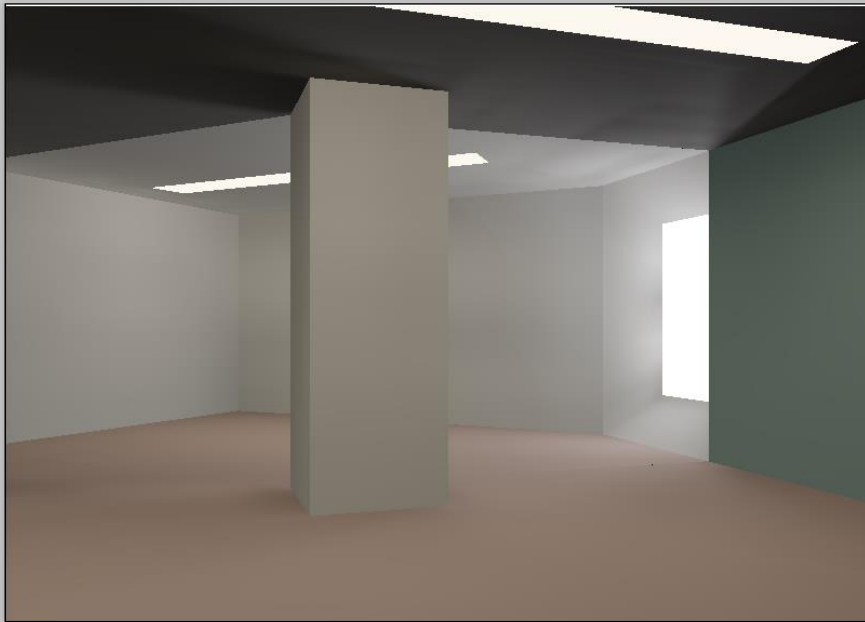
# Past Submissions for Task 2

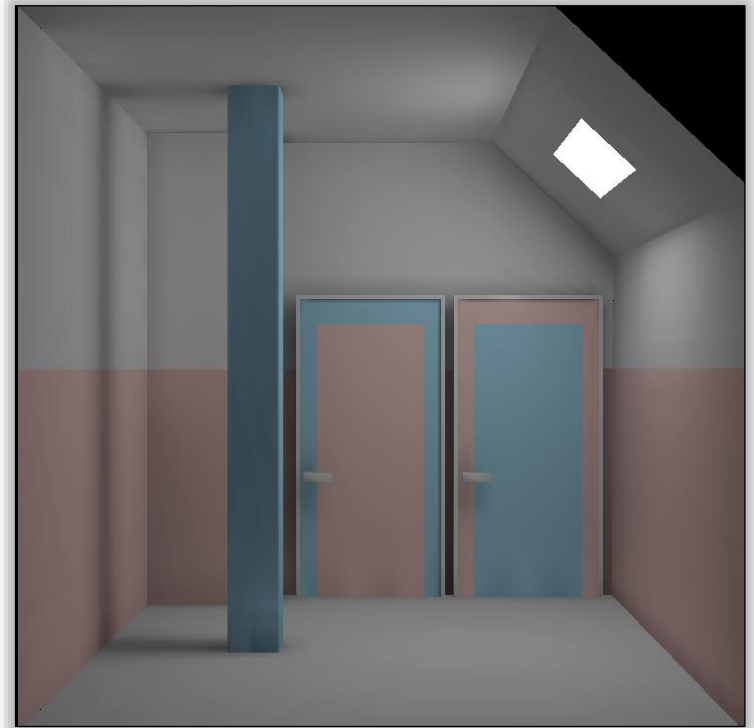


















**The End**