A5 Project Final Report
Title: Water Cooling
Name: Ziyang Shan
Student ID: 20605869
User ID: z7shan

Final Project:

Purpose :

To implement robust ray tracer and render a high-quality 2K image as desktop background.

Extend A4 and implement astonishing visual effects/algorithms like reflection, refraction.

Create a dream scene had in mind.

Statement :

As one who has a fine arts background, the ray tracing scene I have in mind is a Nvidia RTX 2080ti diving into water. This should appeal to people with a PC build background. For general audience, it is still interesting to see an uncanny scene which has electronic parts in water. There will be one light source shooting from above the water to the graphics card with a lot of water bubbles around, as well as the bottom of the water which will create a glare effect. See images at last page for better impression.

It will require me to do fair amount of modeling to make a detailed realistic model of Nvidia RTX 2080ti. Reflection, refraction and photon maps will be implement to create nice look of water. Texture mapping and bump mapping will be helpful to polish details. Before the process I will implement essential features namely refraction, reflection of water in order to render a believable image, and multi-threading to achieve a better work flow.

I will learn more of script modeling, interesting ray tracing algorithms and optimization tricks along the way.

Technical Outline :

To model a believeable graphics card I need **additional primitives** like cylinder, **CSG** and of course **texture mapping**.

To create glare effect at the bottom of the water which is a caustic effect, I need **photon mapping** and **bump mapping** for water surface.

To simulate realist water bubbles I need **reflection** and **refraction**. Also since I need a lot of small bubbles in the scene, I need **spatial division** to help me speed up intersection test.

Finally, I need **multi-threading** to speed things up.

Bibliography :

Constructive Solid Geometry
Peter Shirley: pg.13 10.7 Constructive Solid Geometry

Recursive Ray Tracing:
McConnell: 8.2, 8.3. Recursive Ray Tracing
Watt: 12.1. Recursive Ray Tracing
Bram de Greve, Stanford: Reflections and Refractions in Ray Tracing pg.2 - 3

Refraction:
Peter Shirley: pg.13 10.7 Refraction
Graphics Codex: Direct Illumination; Transmittion

Photon Mapping:
Course Slides: Photon Mapping
A Practical Guide to Global Illumination using Photon Maps, Stanford: pg. 14 - 18 https://graphics.stanford.edu/course
00/course8.pdf

Communication :

Input: lua script with extra gramma implemented. More on this later.

Interaction: None

Output: .png image file

Modules :

I separate my rendering procedure from my shading procedure, such that:

RayTracer.cpp contains concurrent procedure for rendering every pixels. It also contains procedure to pre-compute photon map.

RenderPixel.cpp contains shading procedure for rendering a single pixel

Intersect.cpp contains all function regarding ray intersection.

ConstructiveNode.cpp is a subclass of SceneNode and used to perform CSG

Material.cpp/PhongMaterial.cpp contains all information regarding material along with normal mapping and texture mapping.

New syntax in lua :

gr.material(..., shininess, opacity) has an extra double type [0, 1] parameter at the very end. This will define the opacity of the material. Opacity is been multiplied to the color shading. When opacity==0, no color from the object itself is returned.

gr.construct('name', 'mode') returns a ConstructiveNode and is used to perform CSG. "mode" option is either one the three "union", "difference", and "intersect". ConstructiveNode only accepts two children.

For "difference" whichever child is added first is the solid shape and the other one is the subtraction. All implementation works except "intersect" mode.

For "union" the surfaces inside the CSG should not exist (i.e. they will not be seen when both children are transparent)

I don't need to perform intersect in my scene so I did not implement it. However, I know that "intersect" is the converse of "difference", and I made my code modulized such that "intersect" mode can be easily implement if needed.

node:set_normalmap('filename') looks for a png image in ./Assets and loads as a normal map for the given node. Syntax enforces the node has to have already set_material previously. The node then uses the normal map to as it's normal at given point of intersection.

node:set_texture('filename') looks for a png image in ./Assets and loads as a texture for the given node. Unfortunately, my scene does not require to map texture onto uneven surface, hence the implementation cannot garentee mapped result when performing onto curved surface although my implementation still takes the idea of u, v coordinate mapping.

gr.nh_cylinder('name', {x, y, z}, radius, height) returns a non-hierarchical cylinder primitive with 'name', XYZ position and its radius and height.

Worth Mention Algorithms :

Concurrency My implementation uses global_x and global_y as designated pixel position. Every threads in each iteration takes global_x and global_y as their next job and increament those two. This method garentees average distribution of work on all threads (i.e. no thread would finish earilier than others just because they got the relative easy jobs).

Super Sampling is my extra feature in A4. it is implemented by shooting 4 rays to four corners of the single pixel and store the average of four returned shading colors in a global 2D color_chart. It easy to see that the work is only width+1 X height+1 hence the run-time stays the same.

Photon Mapping I performed photon mapping on one surface that is the ground to achieve caustics. Lights are point light and are mainly distorted by the water surface with a normal map in my scene.

Different to what is suggested in Ä Practical Guide to Global Illumination using Photon Maps, Stanford; I collect nearby photons using squares instead of circles. This allows me to just use a 2D array to store a mapping for a surface. The mapping is pre-computed before shading and uses multi-threading for efficiency.

Totally 36000 photons are been casted, and the patch is a 2.5 units square. Both data is pushed to global to support concurrency.

Constructed Solid Geometry is mentioned in the upper section under **gr.construct('name', 'mode')**

Refraction uses formulas in Graphics Codex: Direct Illumination, Transmittion. It endorses total internal reflection.

Code Style and Debug :

COMPILE_OPTION.hpp contains macros to toggle almost all shading features, change shadow ray max recursion, and number of threads.

All important features are guarded by its own option so it can be easilly toggled.

It also has option like LIGHT_TURN_ON that makes debug more easier.

DEBUG.hpp contains useful macros for debug purposes, like

DASSERT does assertions,

DPRINT prints variables,

DPRINTVEC prints vectors,

and DCOUNT counts occurances.

Known Issues :

Spatial division is not implemented.

Texture mapping only works for flat surface.

Photon Mapping is only performed on one surface.

"intersect" mode is not implemented in CSG.

Objectives:

Full UserID: 20605869 **Student ID: z7shan**

___ 1: Photon Mapping for Caustics

___ 2: A detailed graphics card model.

___ 3: Additional Primitive: cylinder

___ 4: Constructive Solid Geometry (CSG).

___ 5: Reflection

___ 6: Refraction

___ 7: Texture Mapping.

___ 8: Bump Mapping.

___ 9: Spatial Division.

___ 10: Multi-threading.

A4 extra objective: Super Sampling.
Extra reference images next page.

Figure 1: Overall Impression

Figure 2: Example of Glare effect at bottom of the pool

Figure 3: Photo of Nvidia RTX 2080ti