Ray Tracing A Messy Kitchen Counter

Joanne Hong (20470862)

December 4, 2017

The purpose of this project was to display a visually pleasing image using various ray tracing techniques we learned in class. This project will build on top of my work in assignment 4. Note that for assignment 4, I did not implement any extensions.

1 Topics

- Photorealism via Ray tracing:
  - Mirror reflections
  - Refractions
  - Phong shading
  - Antialising
  - Texture mapping
  - Bump mapping
  - Soft shadows
  - Constructive solid geometry
- Animation of gaseous particles

2 Statement

The scene I generated contains:

- A metal kettle and kitchen knife blade: portrays mirror reflections
- A wine cup and bottle: portrays refractions
- A brick wall: portrays bump mapping
- A marble countertop: portrays texture mapping
- A piece of cheese: portrays constructive solid geometry

Phong shading, antialising and soft shadows should be evident on all objects.

I also created animation of steam evaporating from the kettle. The motion of the individual particles will be modelled and different images will be generated and recorded to convey particle motion. Refer to the technical outline for more details.

3 Technical Outline/Implementation

3.1 Mirror reflections and refractions

The reflections and refractions were implemented as recursive secondary rays. Reflective rays reflect at the same angle as their incident rays. The computation for reflection rays can be found in subroutine `generate_reflectio_rays` in `Project.cpp`. For refractive rays, their angles are dependent on the refraction index of the mediums. According to this Wiki article(https://en.wikipedia.org/wiki/Refractive_index), the refractive index of air is around 1 and glass around 1.62. I used this information for my project. Moreover, the computation for this can be found in `generate_refraction_rays` in `Project.cpp`. The depth of the reflections can be specified with `RECURSE` constant in `Project.cpp`.

3.2 Phong Shading

For Phong shading, I first preprocessed the vertex normals of my `Mesh` instances. This was accomplished using a blender tool that generates the appropriate `.obj` files for me. Here, the vertex normals are labelled with `vn`. I edited `Mesh.cpp` appropriately to store the vertex normals in `std::vector`. Then, at runtime, the vertex normals of each triangle in a mesh were interpolated with techniques used in class for Gouraud shading. In particular, I used barycentric coefficients to linearly combine the vertex normals. As a result, the shading looks a lot smoother and less faceted. Note that barycentric coefficients were solved using Cramer's rule [3].

3.3 Antialiasing

For antialising, I used the method of adaptive supersampling. I averaged the radiance of surrounding pixels when they contain different objects (or faces if the same objects) to avoid the "jagged" look. In `Project.cpp`, I globally keep track of `int nodes[x][y]` and `int faces[x][y]` which are node ID and face numbers of all the pixels `(x,y)` seen so far.

3.4 Bump and Texture Mapping

For bump and texture mapping, I used the `lodepng` library to load the image as an array of unsigned chars representing their rgb components. The code for this can be found in `PhongMaterial.cpp`. In order to be able to specify the

image to be uploaded, I extended the `Lua` command for `material` to accept file names (for bump and texture mapping separately). These changes can be found in `scene_lua.cpp`.

For texture mapping, the rgb files are directly used as the colour of the pixel (see `get_texture` in `Project.cpp`). For bump mapping, the image used is interperated as a normal map, and the rgb values are used as the normals (see `recompute_normal_for_bump_mapping`).

The texture I used for the countertop is in `Assets/marble.png` and the normal map is in `Assets/brick.png`.

3.5 Soft shadows

For soft shadows, I generated distributed rays at primary ray intersections to simulate an area light source instead of a point light source. For my ray tracer, the area of light is a line and so the generated image may look like it's lit under a fluorescent light. The number of shadow rays and the linear distance between them are specified in `NUM_SHADOW_RAYS` and `SHADOW_SHIFT` respectively.

3.6 Constructive Geometry

For constructive geometry (CSG), I extended my intersection subroutine implemented in assignment 4 with a "difference" operator (the extension can be found in subroutine `find_node_with_smallest_t` in `Project.cpp`). The difference operator essentially keeps track of the entering and exiting intersection pairs for each primary ray on objects of interest and "subtracts" these intervals. Futhermore, the interval of children of CSG objects are subtracted from the parents.

The CSG object in my project is a piece of swiss cheese wit holes. A group of sphere objects are subtracted from a single cube object to achieve this effect.

3.7 Particle System Animation

For the particle system animation, the following classes were created:

- `Particle`: encapsulates velocity, acceleration and lifespan of a single particle
- `ParticleSystem`: encapsulates logic for adding new particles (at random locations) and deleting dead particles and updating all the particles in the system
- `Particles`: a subclass of `Primitive` to distinguish between other `GeometryNode` instances in `Lua` file

Each particle starts at a same position, but different starting velocities (upward and left or right) and a downward (in y direction) acceleration due to gravity [2]. After each ray tracing interval, velocities are updated according to acceleration and the positions are updated. If the lifespan of a particle expires, it is removed from the pool by `ParticleSystem`. New particles are added at each time interval as well. Also, the lifespan of the individual particles determine their transparency.
Note that to create different images for the animation, in the `scene_lua.cpp` file, I call `Project_Render` procedure `T` times (`T` is specified in the command line argument). For each iteration `i`, a file `project_[i].png` is generated. At the end, a linux command line `convert` is used to create a gif out of these images.

3.8 Feature Enable

To enable and disable some features, I've added flags at the top off `Project.cpp` file:

- `SUPER_SAMPLING`
- `PHONG`
- `CSG`

3.9 Development Process

For development, I created a private repo in University of Waterloo's gitlab and pushed commits regularly. This allowed me to roll back to past versions to find bugs.

3.10 Improvements

There is a lot of room for improvements in efficiency. There was some optimization done with bounding volumes (assignment 4) and adaptive antialising, but additions like distributed rays for soft shadows slow down the ray tracing process proportionally. Maybe an 11th objective, spaital subdivision would have been appropriate since my scene is relatively easy to divide into subdivisions.

I have also wanted to implement radiosity. The approximated ambient lighting isn't enough make the photo look realistic.

4 Manual

4.1 Comprehensive Guide to Running the Program

As usual, run `premake4 gmake` and then `make` in the `project` directory. Then, move to the `Assets folder` and run `../Project [n = number of keyframes]`. This will create n images with the name `project_[i].png` where $i \leq n$. It will

also create a gif file in the same directory if $n > 1$. Note that the lua file used for the final image is in `Assets/project.lua`.

5 Bibliography

1 Glassner, A. (ed) An Introduction to Ray Tracing. Academic Press New York, N.Y. 1989

2 W. T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", Computer Graphics, vol. 17, no. 3, pp 359-376, 1983

3 Ericson, C. Real-Time Collision Detection. CRC Press, 2004

6 Acknowledgments

Thanks to Stephen Mann and Gegory Philbrick for helping me with my proposal. Also thanks to user John Calsbeek for his summary of Christer Ericson's Real-Time Collision Detection on Stack Overflow (https://gamedev.stackexchange.com/questions/23743/whats-the-most-efficient-way-to-find-barycentric-coordinates) - it showed me the most efficient way to compute barycentric coordinates.

7 Objectives

Name: Joanne Hong

User ID: j46hong

Student ID: 20470862

___ 1: Mirror reflections on at least one object (`Assets/reflection.png` and `Assets/no_reflection.png`)

___ 2: Refraction on at least one object (`Assets/refraction.png` and `Assets/no_refraction.png`)

___ 3: Phong shading on objects (`Assets/phong.png` and `Assets/no_phong.png`)

___ 4: Adaptive supersampling for antialiasing (`Assets/antialiasing.png` and `Assets/no_antialiasing`)

___ 5: Texture mapping on at least one object (`Assets/texture_mapping.png` and `Assets/no_texture_mapping`)

___ 6: Bump mapping on at least one object (`Assets/bump_mapping.png`)

___ 7: Soft shadows (`Assets/soft_shadows.png` and `Assets/no_soft_shadows.png`)

___ 8: Constructive solid geometry used for at least one object (`Assets/csg.png` and `Assets/no_csg.png`)

___ 9: Unique scene is portrayed in the image (`screenshot.png`)

___ 10: One animation of "fuzzy" or "gaseous" object by modelling with particle systems (`Assets/project.gif`)