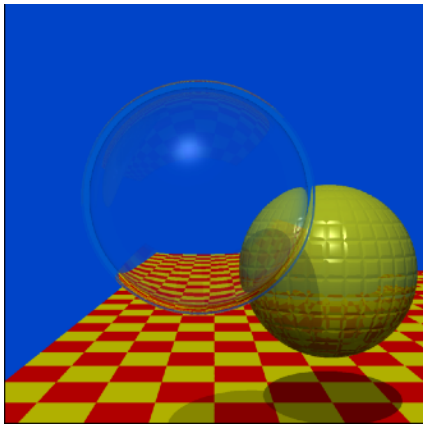
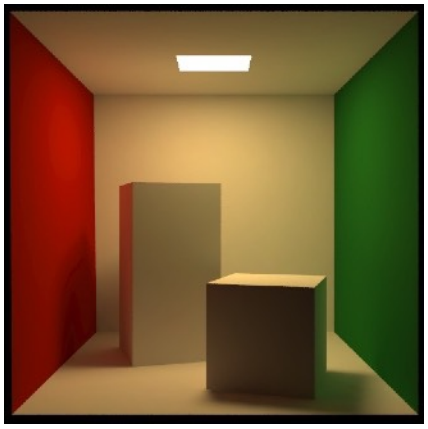
Ray Tracing

Assignment #4

CPSC 453 • Fall 2016 • University of Calgary


Overview & Objectives

Ray tracing is a classic computer graphics technique used to synthesize perspective images of virtual three-dimensional scenes. The core algorithm is also simple enough that it can be implemented within the scope of a few days' work, even with little or no infrastructure.

It is finally time for you to enter the world of 3D! This will be your first real experience with rendering images of three-dimensional scenes in CPSC 453. The goal of this assignment is to learn and appreciate the core concepts behind synthesis of perspective 3D images. You will implement a basic ray tracing system to simulate the optics of a pinhole camera. You will also gain experience using shading models that approximate ambient, diffuse, and specular light reflection on surfaces, which give objects the appearance of having different materials.

This assignment consists of a written component and a programming component. The programming part is described first, but **note that written answers to the assignment questions are due ahead of the program submission.**

Due Dates

Written component	**Wednesday, November 9 at 11:59 PM**
Programming component	**Monday, November 21 at 11:59 PM**

Programming Component

There are a total of four main parts to this programming assignment with an additional requirement to render, save, and submit final images of three scenes. This component is worth a total of 20 points, with the point distribution as indicated in each part, and an additional possibility of earning a "bonus" designation. Since the bonus is a binary state, it will only be awarded to submissions that do an exemplary job of meeting the requirements.

You will not need to use the OpenGL rendering pipeline, or write OpenGL shader code, to complete this assignment. Instead, you will be writing your ray tracing code in C++ to be run on the CPU. The OpenGL Mathematics library (GLM, developed independently of the Khronos Group) can provide you with C++ geometric vector data types and associated operations, similar to those found in the OpenGL Shading Language, that you may find helpful for this assignment. A C++ class to help you display partial or full images to the screen and save them to disk will also be provided.

In addition to your source code, you will need to submit a final rendered image of each of three scenes, and instructions on how to use your program to generate these images. Note that there are no specific deliverables associated with each part in this assignment. The parts provide a suggestion for the progression of your implementation and an allocation of points.

Part I: Ray Generation (3 points)

Write a program that creates an OpenGL rendering window and generates a view ray for each pixel of your window. You will probably find it most convenient to represent your ray as a position and a direction vector. If you express your scene geometry in the camera reference frame, then your viewing rays will conveniently have zero vectors for their position of origin. Provide a means in your code to set the field of view (or equivalently, "focal length" and sensor/film size) of your virtual pinhole camera—you will find this capability useful later on.

Notes & Hints:

- Obviously, you will still not be able to see anything after completing this part. However, it may help for debugging if you rendered each pixel's ray direction as an RGB colour. You are probably on the right track if you see an image with smooth radial gradients.

Part II: Ray Intersection (6 points)

This part forms the core of your ray tracing program. For every view ray you generated in Part I, you will need to test it against each of the objects in your scene and find the closest intersection, if any. That surface is what you see when looking through that image pixel.

Add functions to your program to find intersections between your rays and sphere, plane, and triangle objects. How you represent these objects in your program is entirely up to you. As you complete each intersection test, you may wish to create a simple object of that type to verify that it works before proceeding to implement the next test. After you've completed the intersection tests, you can be begin to render preliminary versions of the provided scenes.

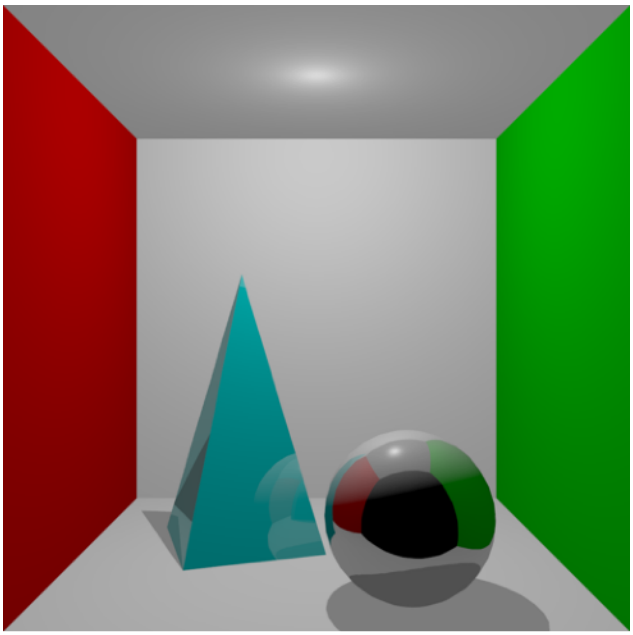
In addition to your own scene (Part V), you will be expected to render two scenes with your ray tracing program: our variation of the famous "Cornell Box", and a second scene with various types of objects in it, both depicted below. The geometries of these scenes are specified in accompanying text files designed to be both human- and machine-readable. You may choose to write a simple text parser to read these files and render them, or simply cut and paste the values directly into your code for each scene.

Notes & Hints:

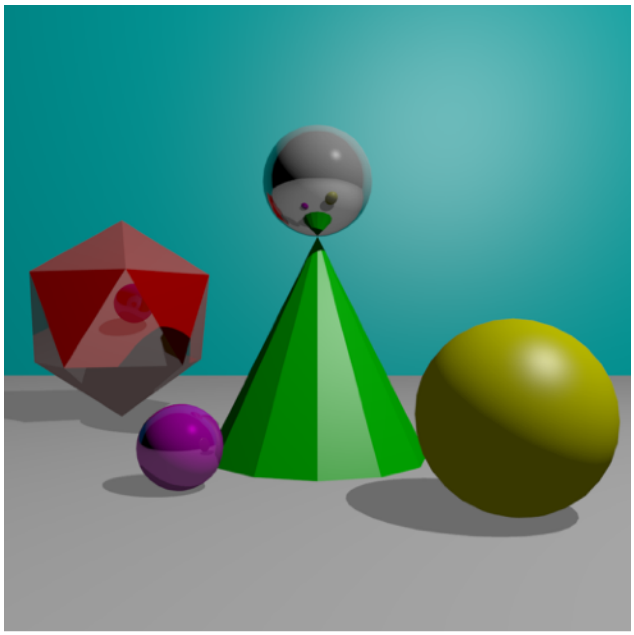
- At this stage, it is would be straightforward and beneficial to verify that Parts I and II are both working correctly. Even though your image of the scene will not yet have lighting, materials, or shading, it should be rendered in correct perspective. If you assign a unique colour to each object and set each pixel to the colour of the object that its ray hits, you should be able to generate images like Figure 4.11 in your textbook.

Part III: Shading (4 points)

Shading is what makes your scene come alive! Although this is not the most challenging part, it is perhaps the most important part of the assignment.



Scene 1



Scene 2

First add a light source to your scene. Remember, without light there is nothing! For the purposes of this assignment, a light is just an infinitely small point that emits white light. The position of the single light is indicated in each of the provided test scenes. You may certainly add support for more than one light if you wish.

Implement a shading model that includes diffuse, specular, and ambient components to assign colours to each of your ray-object intersections. You may use Equation 4.3 in Marschner & Shirley as a starting point, but feel free to explore more sophisticated models if you'd like. No material properties are specified in the provided scenes, so you will need to define diffuse, specular, and ambient colours/intensities, as well as the specular exponent, for each object in the scenes. Try to make the colours and materials of your objects look as similar to the images on the previous page as possible, but do not worry about matching the appearances exactly. Also keep in mind that you will not have shadows or reflections until you finish Part IV.

Part IV: Shadows & Reflections (4 points)

The final elements to add, which will give your scenes that uniquely "ray-traced" appearance, are shadows and reflections. Shadows can be rendered by tracing a ray from your intersection/shading point to light sources in your scene. If the shadow ray is blocked by another object, then your surface will not receive the diffuse and specular contributions from that light.

Reflections are computed by reflecting the view or eye ray about the surface normal and recursively tracing the reflected ray within your program. Surfaces are often not 100% reflective, and you can achieve a partial reflection by mixing the colour obtained from the reflected ray with the shaded colour of the object itself. Because these are technically pure specular reflections, it is often most visually correct to keep the amount of ray-tracing reflection consistent with the object's specular shading parameters (e.g. modulating the reflected colour by the material's specular colour).

Set reflective properties of the objects in the two provided test scenes so that your rendered images look similar to the ones shown. Note that no object is purely reflective.

Notes & Hints:

- When tracing shadow rays, you may need to be careful with self-occlusions, or rays intersecting the same object you are shading. Think carefully about when you may need to test for occlusion against the object you are shading, and when you may not.
- Some view rays may be reflected many times, or at worse, lead to an infinite recursion! Typically, a ray tracer will limit the number of "bounces" a ray makes before giving up that trace. It depends on the scene, but a recursion depth limit of 10 is very reasonable.

Part V: Artistry & Final Renderings (3 points)

Finally, here is your chance to be creative! Design and render a scene of your own that contains at least one of each geometric element type (sphere, plane, triangle). You may model a real-world object, create something entirely abstract, parse and render a 3D model file, or whatever else you can think of. Your scene can be something very simple too, but we will not give you credit for this part if we don't think you spent at least 30 minutes' effort here.

A bonus designation will be given to the best scenes rendered by programs submitted in this assignment. We will award at least one bonus for artistic merit and one for technical merit.

With your program now complete, take this opportunity to do final, high quality renderings of each of the two scenes provided with the assignment and your own scene. **Save the three final images to file and submit them with your assignment.**

Bonus Part: Depth of Field or Real-Time Ray Tracing

There are two possibilities for earning a bonus in this assignment. Each is described in the paragraphs below. If you're so compelled, you may complete both to earn a "mega-bonus"!

Remember that a simultaneous advantage and disadvantage of a pinhole camera is that every element of the scene is always in focus. Sometimes we do want to have the realism or artistic flair that an image with a defocused background or foreground offers. The first opportunity to earn a bonus is to generate synthetic photographs of your scenes by tracing rays through a virtual lens. Create and render at least one additional scene, perhaps similar to that shown below, to showcase your depth of field effect. Make sure that you design your virtual lens and/or scene so that at least one object is in focus. Try varying your virtual aperture to obtain different levels of defocusing.

Ray tracing is what one may call an "embarrassingly parallel" problem. The colour of each image pixel can be computed independently from those of other pixels, but unless you are doing something special, you are using only one CPU core to render your images in this assignment! The second opportunity for the bonus is to flex those OpenGL muscles you've developed and utilize the 576 cores on the GPU. Implement the ray tracing algorithm in the

OpenGL shading pipeline, rather than in C++. With the size of the test scenes provided, it should not be a problem to render the scenes interactively, complete with shadows and reflections. Provide a means to interactively move the camera around, or animate a motion path for the camera, to demonstrate that your scenes render in real time.

Submission

We encourage you to learn the course material by discussing concepts with your peers or studying other sources of information. However, all work you submit for this assignment must be your own, or explicitly provided to you for this assignment. *Submitting source code you did not author yourself is plagiarism!* If you wish to use other template or support code for this assignment, please obtain permission from the instructors first. Cite any sources of code you used to a large extent for inspiration, but did not copy, in completing this assignment.

Please upload your source file(s) to the appropriate drop box on the course Desire2Learn site, and indicate any late days used here. Include a "readme" file that briefly explains specific instructions for compiling your program, if needed, and **how to run your program to generate each of your three images**. If your program does not compile and run on the graphics lab computers in MS 239, *the onus is on you to ensure that your submission runs on your TA's grading environment for your platform!* Broken submissions will be returned for repair at a minimum penalty of one late day, and may not be accepted if the problem is severe. Ensure that you upload any supporting files (*e.g.* makefiles, shaders, data files) needed to compile and run your program, but please do not upload compiled binaries or object files.

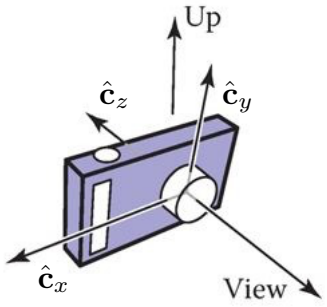
Your program must also conform to the OpenGL 3.2+ Core Profile, meaning that you should not be using any functions deprecated in the OpenGL API, to receive credit for this part of the assignment. We highly recommend using the official OpenGL 4 reference pages as your definitive guide, located at: https://www.opengl.org/sdk/docs/man/.

Assignment Questions

The written component consists of four questions that correspond to respective parts of the programming assignment, and is worth 10 points in total. When answering the questions, show enough of your work, or provide sufficient explanation, to convince the instructors that you arrived at your answer by means of your own.

Question 1: Ray Generation (2 points)

Consider a reference frame with its origin coincident with the optical centre of your virtual pinhole camera (*i.e.* the point where the "pinhole" is). The $\mathbf{c}_x$ basis vector points to the right of the camera, the $\mathbf{c}_y$ basis vector points up with respect to the camera, and the $\mathbf{c}_z$ basis vector points away from the scene in the direction of the optical axis. This is illustrated in the diagram to the right.



A. If you were to render a square image measuring 100×100 pixels with a field of view of 60° into your scene, what would be the *normalized* direction vector for view ray of the upper-left pixel, expressed in the basis described above?

B. Is it possible to crop the image from 1A to form a result that looks as if you had rendered your scene with a 30° field of view? If so, describe precisely where and how much to crop the image. If not, briefly explain why it would not be possible.

Question 2: Ray Intersection (4 points)

As you are ray tracing an image by simulating the camera described in Question 1, you may generate a ray with a parametric equation,

$$\mathbf{r}(t) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + t \begin{bmatrix} 0 \\ 1 \\ -4 \end{bmatrix}.$$

This ray, expressed in the camera frame, has a zero vector origin and a direction of $\mathbf{d} = \mathbf{c}_y - 4\mathbf{c}_z$.

A. Find the intersection point(s) between the above ray and a unit sphere with a centre at (0, 0, -3), expressed in the camera frame. If you found more than one point, which point is nearest to the camera?

B. Now consider a triangle with positions of its three corners at (2, -1, -3), (2, 3, -5), and (-1, -1, -3). What is the normal vector of the plane this triangle rests on?

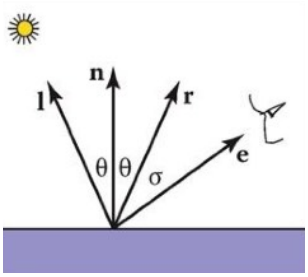
C. Find the position of intersection between the above ray and the plane in 2B.

D. What are the barycentric coordinates of the intersection point from 2C, with respect to the three points of the triangle described in 2B? Does the camera ray described at the beginning of this question intersect this triangle?

Question 3: Shading (2 points)

A. Consider a surface material that has a purely Lambertian (diffuse), white reflection. At what incident light angle, measured from the surface normal, would you see light reflected from the surface at exactly half of the incident intensity?

B. Recall that specular highlights on materials can be approximated using the Phong lighting equation, $c = c_l \max(0, \mathbf{e} \cdot \mathbf{r})^p$, as written in Equation 10.5 and in Figure 10.5 of Marschner & Shirley, reproduced on the right. If your surface material had a Phong exponent of $p = 8$, at what viewing angle (*i.e.* direction of the **e** vector), measured from the surface normal, would you see a specular reflection of exactly half of the incident light intensity?



Question 4: Shadows & Reflection (2 points)

A. Briefly describe the key difference(s) between a view/camera ray and a shadow ray in terms of how they should be treated in your ray tracing system (*i.e.* your code for this assignment, and specifically the parts that deal with intersection testing or shading).

B. Briefly describe the key difference(s) between a view/camera ray and a reflected ray in terms of how they should be treated in your ray tracing system.

You may submit your answers in digital form (typed or scanned) in the appropriate drop box on the course Desire2Learn site, or as hard copy in the assignment boxes on the second floor of Math Sciences. Remember that late days may not be used for the written component!