How to debug a ray tracer
and how to accelerate it
The rendering equation

\[ L_o(x, \bar{\omega}_r) = L_e(x, \bar{\omega}_r) + \int_{\Omega} L_i(x, \bar{\omega}_i) f(x, \bar{\omega}_i, \bar{\omega}_r)(\bar{\omega}_i \cdot \bar{n}) d\bar{\omega}_i \]

For a specific wavelength and time
Phong reflectance model

A ray tracer
A ray tracer
A ray tracer

Calculate Phong
A ray tracer

Calculate Phong
A ray tracer

Calculate Phong

Calculate Phong
An alternate approach
Yet another alternate approach
Hybrid approaches
But let’s start with this.
Things you will need to implement

- Shoot rays
- Intersect rays with objects
- Calculate values according to illumination model
Ray-sphere intersection

https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm
Ray-sphere intersection

\[ R(t) = R_0 + t \cdot R_d, \quad t > 0 \]
Ray-sphere intersection

\[ R(t) = R_0 + t \cdot R_d , \quad t > 0 \]

\[
x = X_0 + X_d \cdot t \\
y = Y_0 + Y_d \cdot t \\
z = Z_0 + Z_d \cdot t
\]

https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm
Ray-sphere intersection

\[ R(t) = R_0 + t \times R_d , \ t > 0 \]

\[
\begin{align*}
x &= X_0 + X_d \times t \\
y &= Y_0 + Y_d \times t \\
z &= Z_0 + Z_d \times t
\end{align*}
\]

\[ S = \{(x, y, z) \mid (x - X_c)^2 + (y - Y_c)^2 + (z - Z_c)^2 = S_r^2\} \]
Ray-sphere intersection

\[ x = X_0 + X_d \times t \]
\[ y = Y_0 + Y_d \times t \]
\[ z = Z_0 + Z_d \times t \]

\[ S = \{(x, y, z) \mid (x - X_c)^2 + (y - Y_c)^2 + (z - Z_c)^2 = S_r^2\} \]

\[ (X_0 + X_d \times t - X_c)^2 + (Y_0 + Y_d \times t - Y_c)^2 + (Z_0 + Z_d \times t - Z_c)^2 = S_r^2 \]
Ray-sphere intersection

\[ x = X_0 + X_d \cdot t \]
\[ y = Y_0 + Y_d \cdot t \]
\[ z = Z_0 + Z_d \cdot t \]

\[ S = \{(x, y, z) \mid (x - X_c)^2 + (y - Y_c)^2 + (z - Z_c)^2 = S_r^2 \} \]

\[(X_0 + X_d \cdot t - X_c)^2 + (Y_0 + Y_d \cdot t - Y_c)^2 + (Z_0 + Z_d \cdot t - Z_c)^2 = S_r^2\]

A quadratic equation: \[A \cdot t^2 + B \cdot t + C = 0\]
Debugging:

(2) Primary ray visualization: Provide code that will produce an image showing line segments indicating the paths of primary rays starting at the camera eye point and stopping at the first surface intersections. This option could be implemented by writing a .scn file with a representation for each ray (e.g., add "line" commands to the scene). Or, it could be implemented by extending rayview.cpp to show rays emanating from the camera provided in the .scn file as the scene is viewed interactively. You should restrict the number of rays displayed so that they are clearly visible. Commands for this feature do not need to be included in the runme file, but a description of your process should be included in the writeup.

What’s wrong here?
(2) Primary ray visualization: Provide code that will produce an image showing line segments indicating the paths of primary rays starting at the camera eye point and stopping at the first surface intersections. This option could be implemented by writing a .scn file with a representation for each ray (e.g., add "line" commands to the scene). Or, it could be implemented by extending rayview.cpp to show rays emanating from the camera provided in the .scn file as the scene is viewed interactively. You should restrict the number of rays displayed so that they are clearly visible. Commands for this feature do not need to be included in the runme file, but a description of your process should be included in the writeup.
Debugging:

(2) Primary ray visualization: Provide code that will produce an image showing line segments indicating the paths of primary rays starting at the camera eye point and stopping at the first surface intersections. This option could be implemented by writing a .scn file with a representation for each ray (e.g., add "line" commands to the scene). Or, it could be implemented by extending rayview.cpp to show rays emanating from the camera provided in the .scn file as the scene is viewed interactively. You should restrict the number of rays displayed so that they are clearly visible. Commands for this feature do not need to be included in the runme file, but a description of your process should be included in the writeup.
Common pitfalls
Common pitfalls
Common pitfalls
Acceleration
Acceleration

Test bounding box
Acceleration

Cache last intersection
Acceleration

Front to back
Acceleration

Use data structures
BSP Tree

http://en.wikipedia.org/wiki/Binary_space_partitioning
BSP Tree

http://en.wikipedia.org/wiki/Binary_space_partitioning