# How to debug a ray tracer

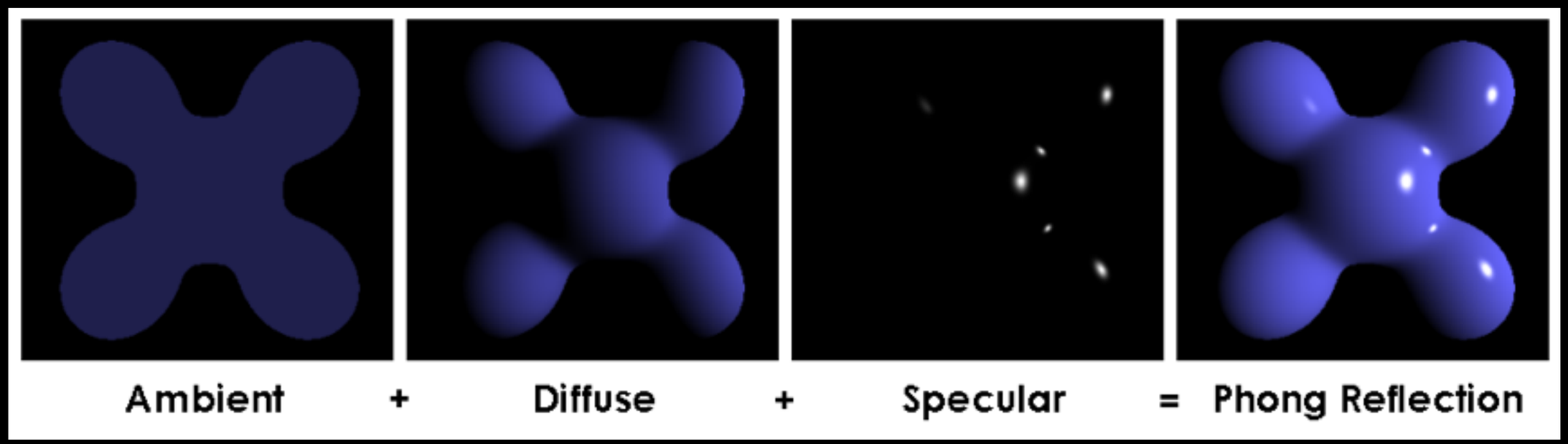
and how to accelerate it

# The rendering equation

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

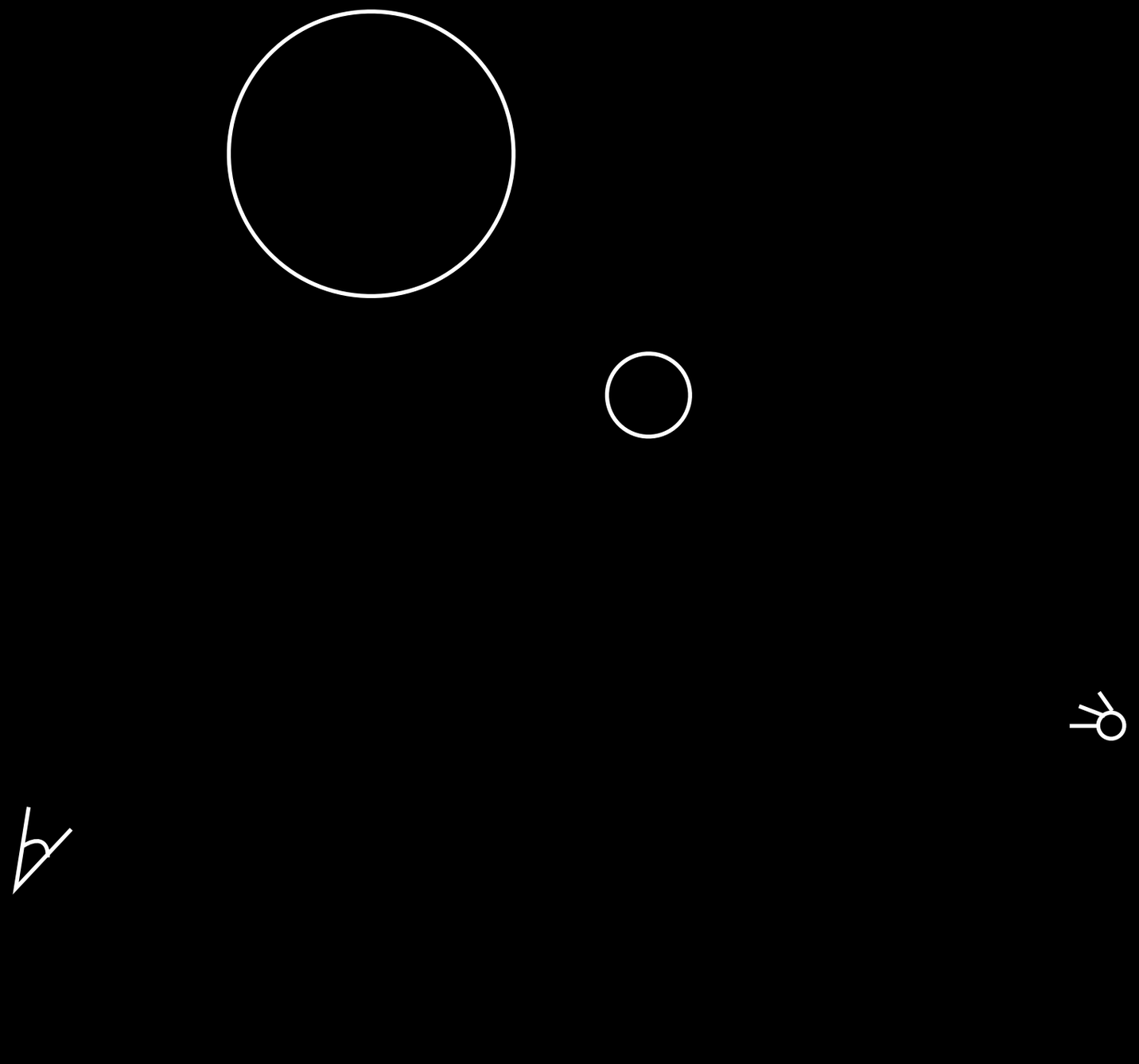
For a specific wavelength and time

# Phong reflectance model

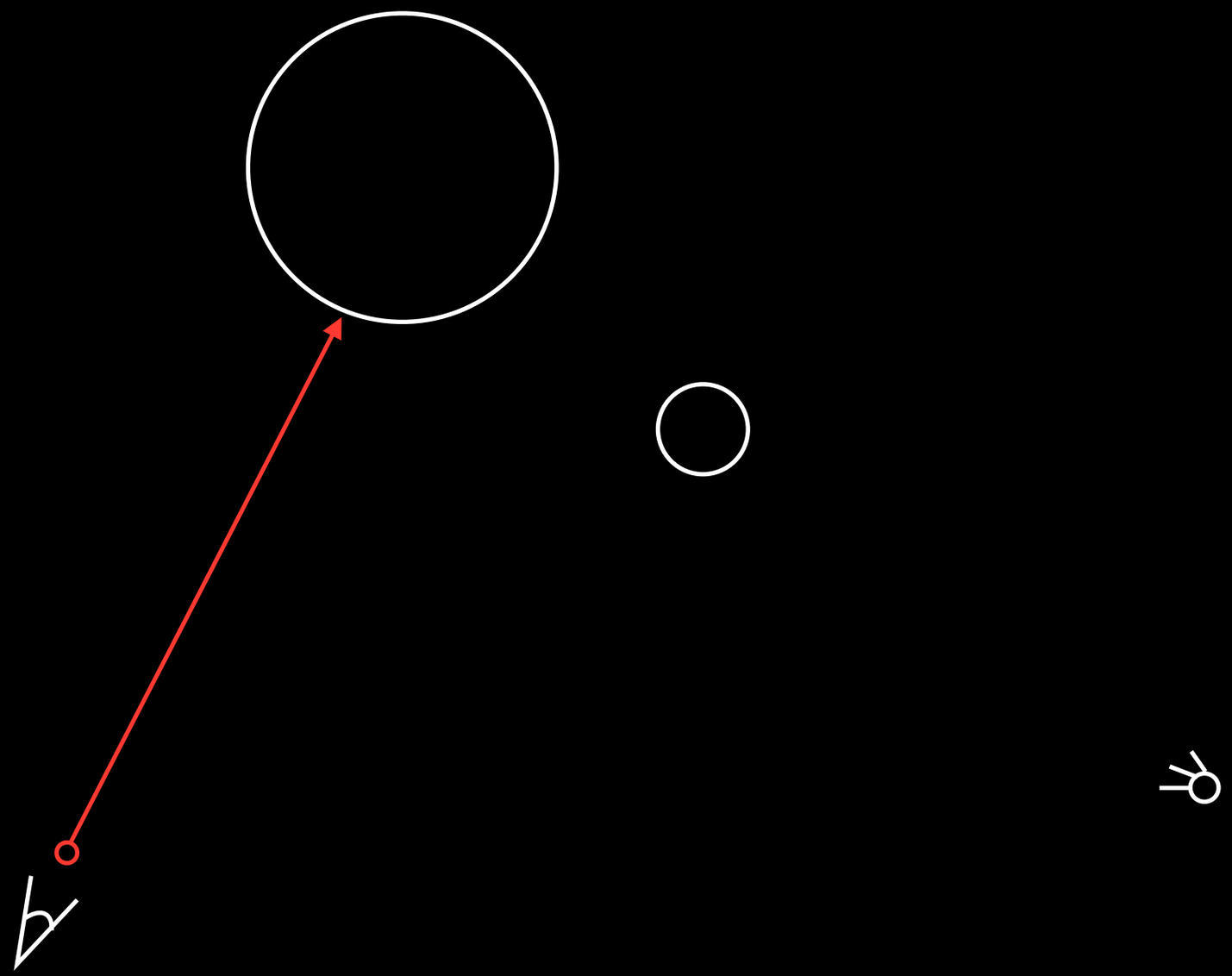



http://en.wikipedia.org/wiki/Phong_reflection_model

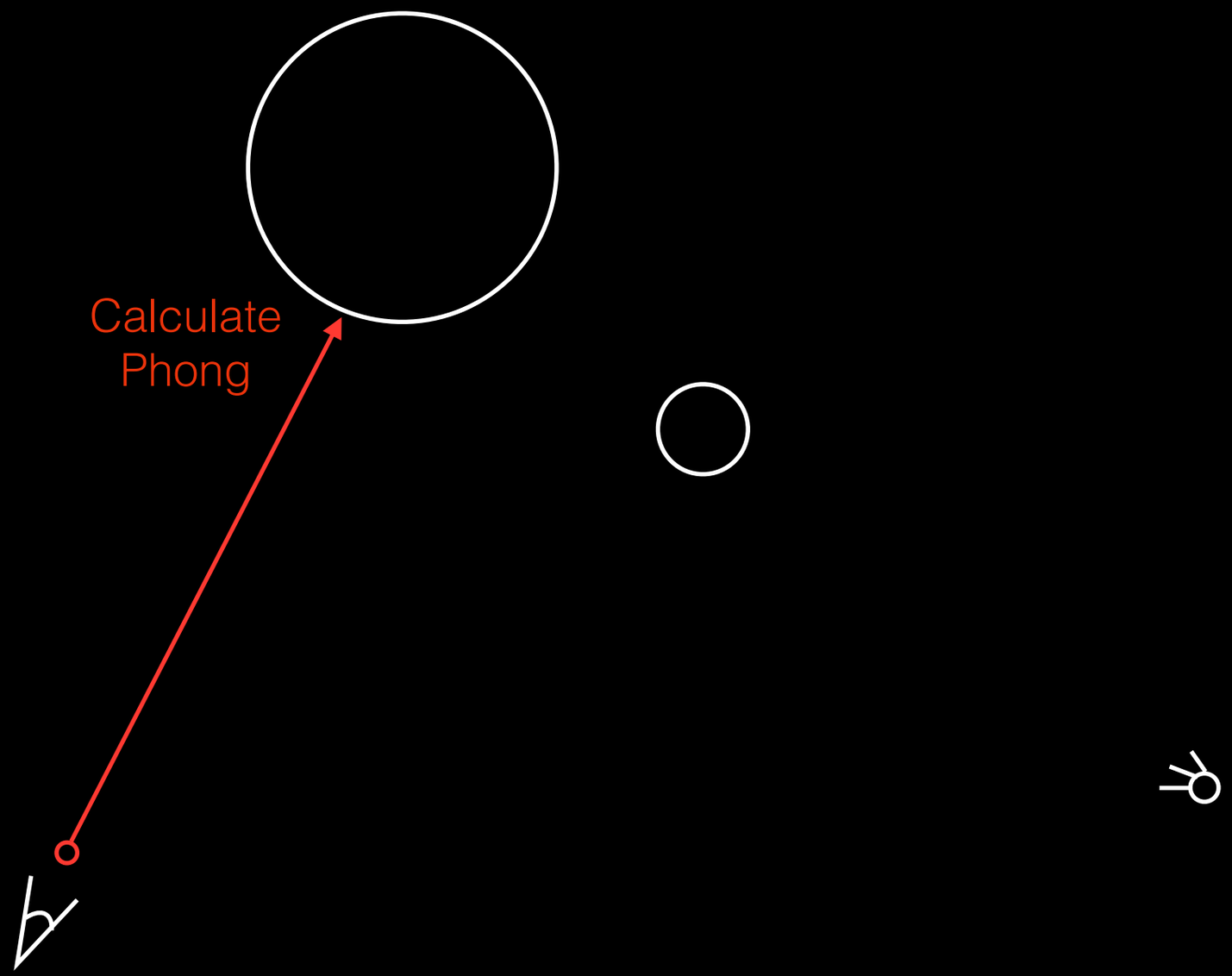
# A ray tracer



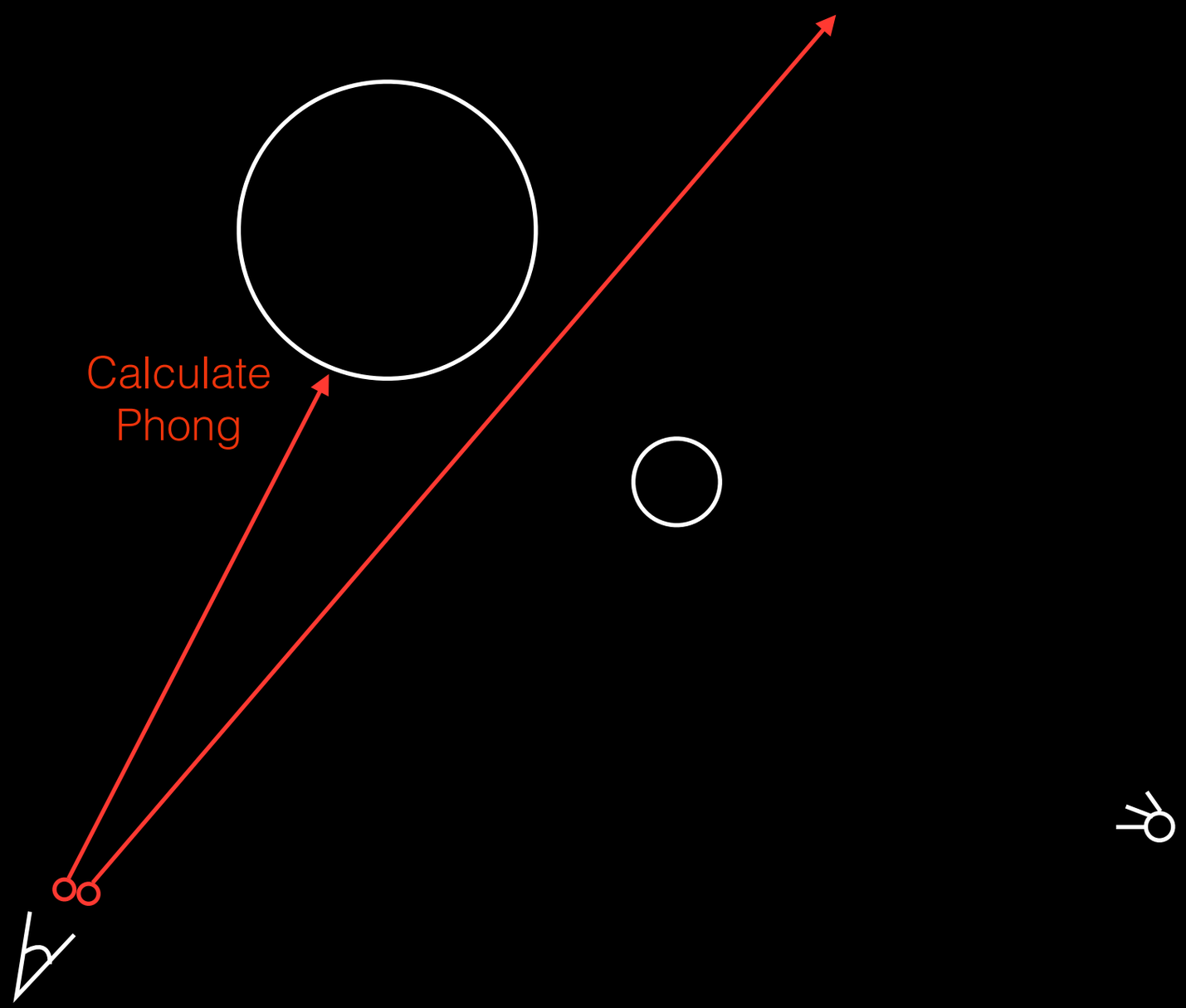
# A ray tracer



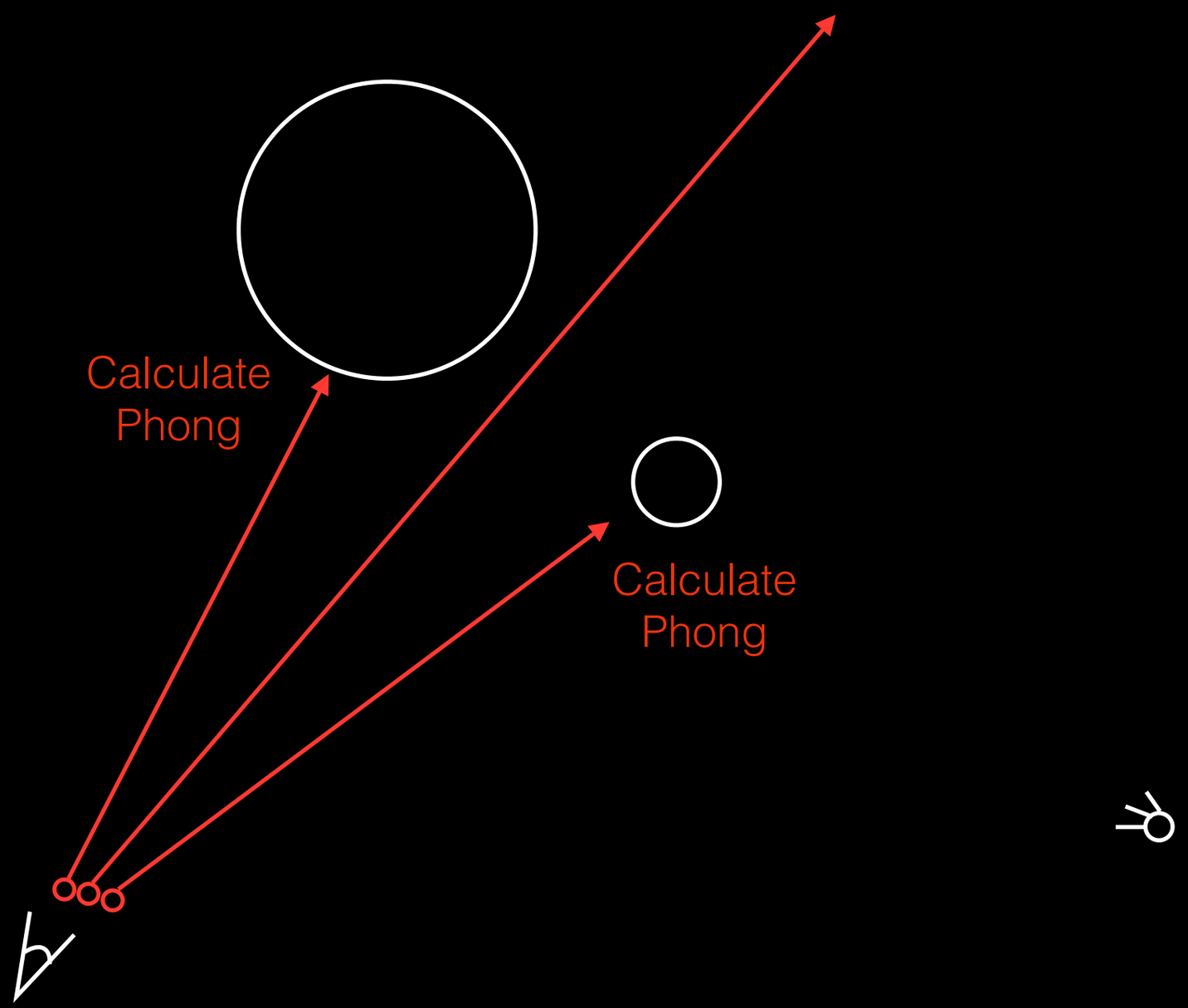
# A ray tracer



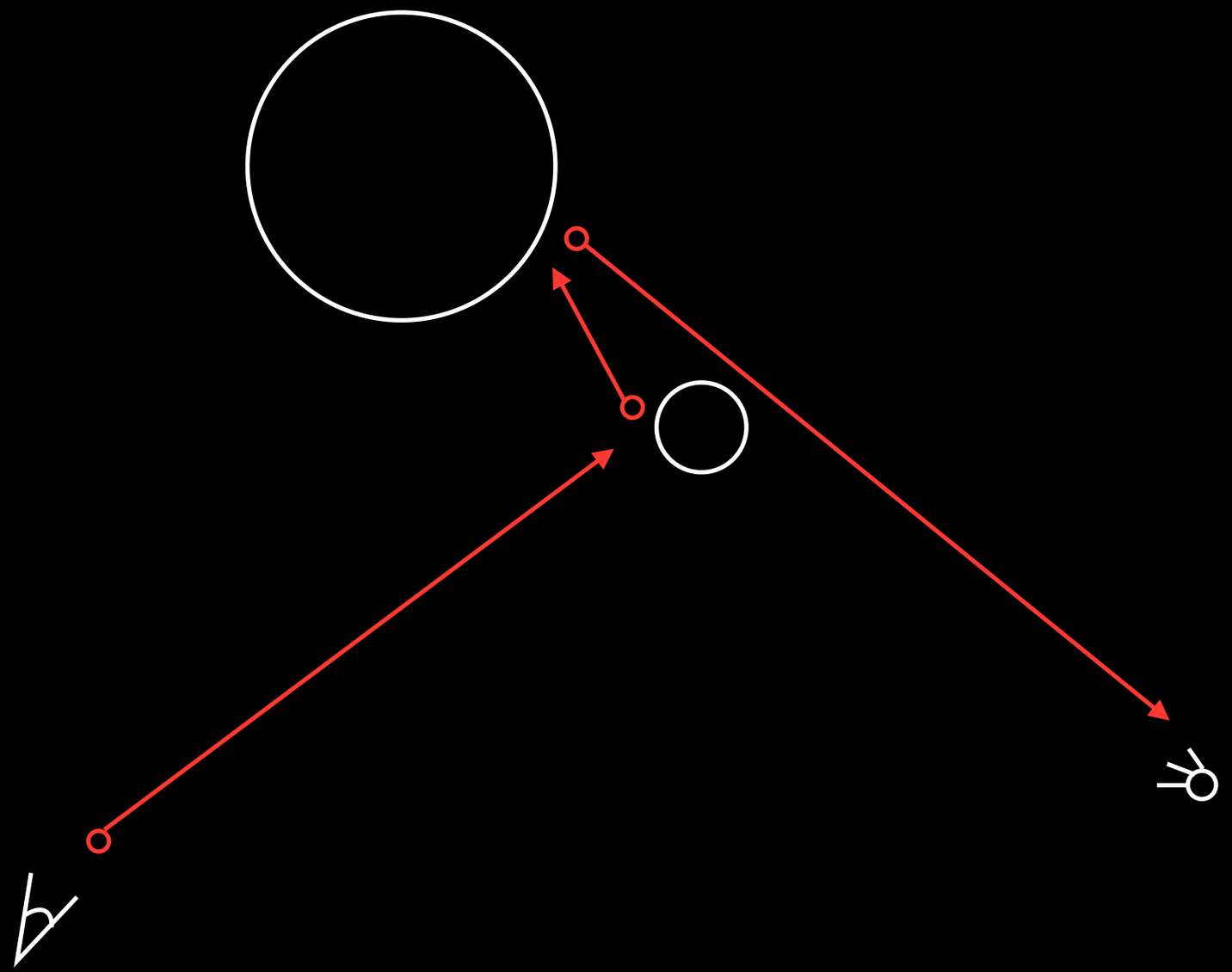
# A ray tracer



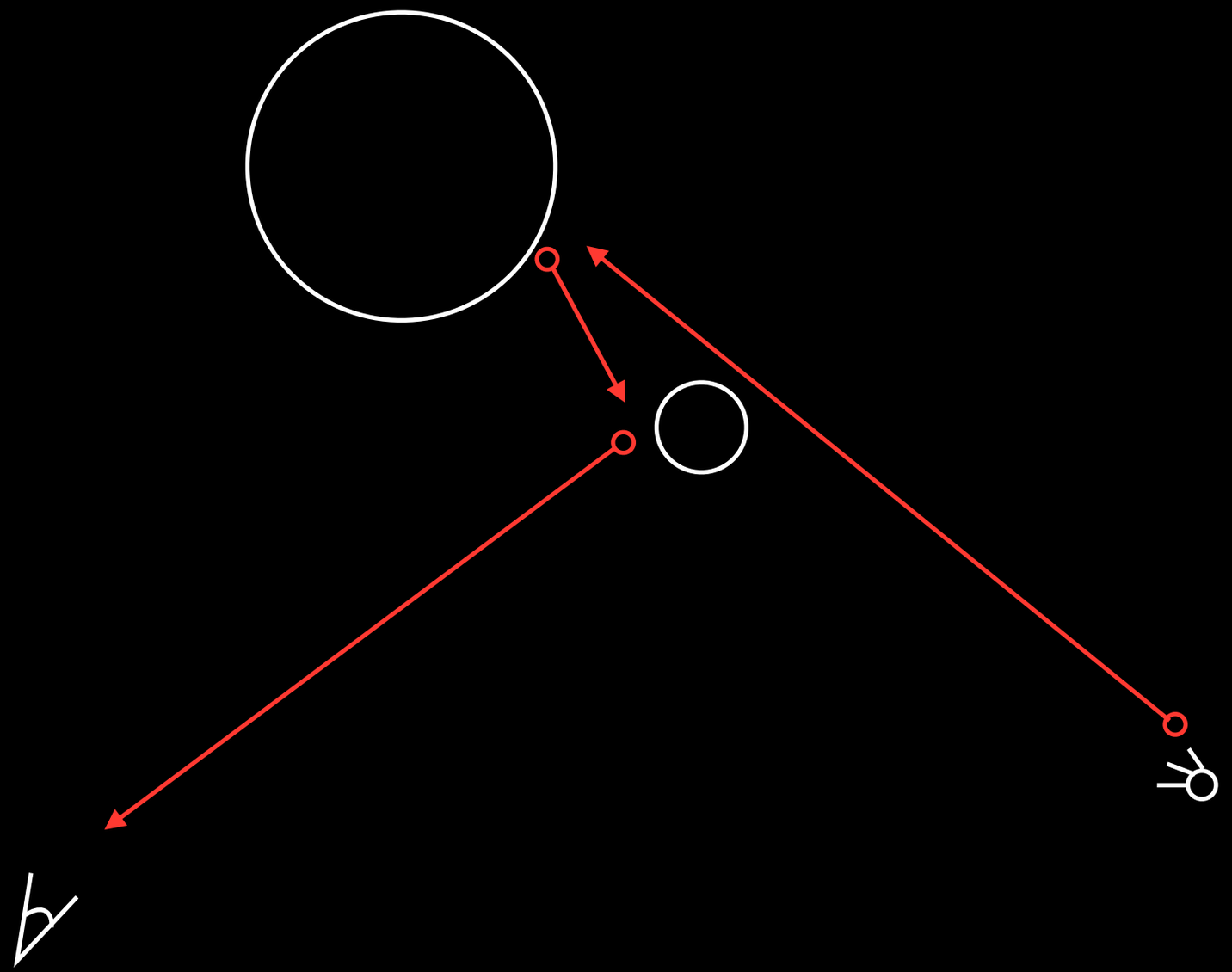
# A ray tracer



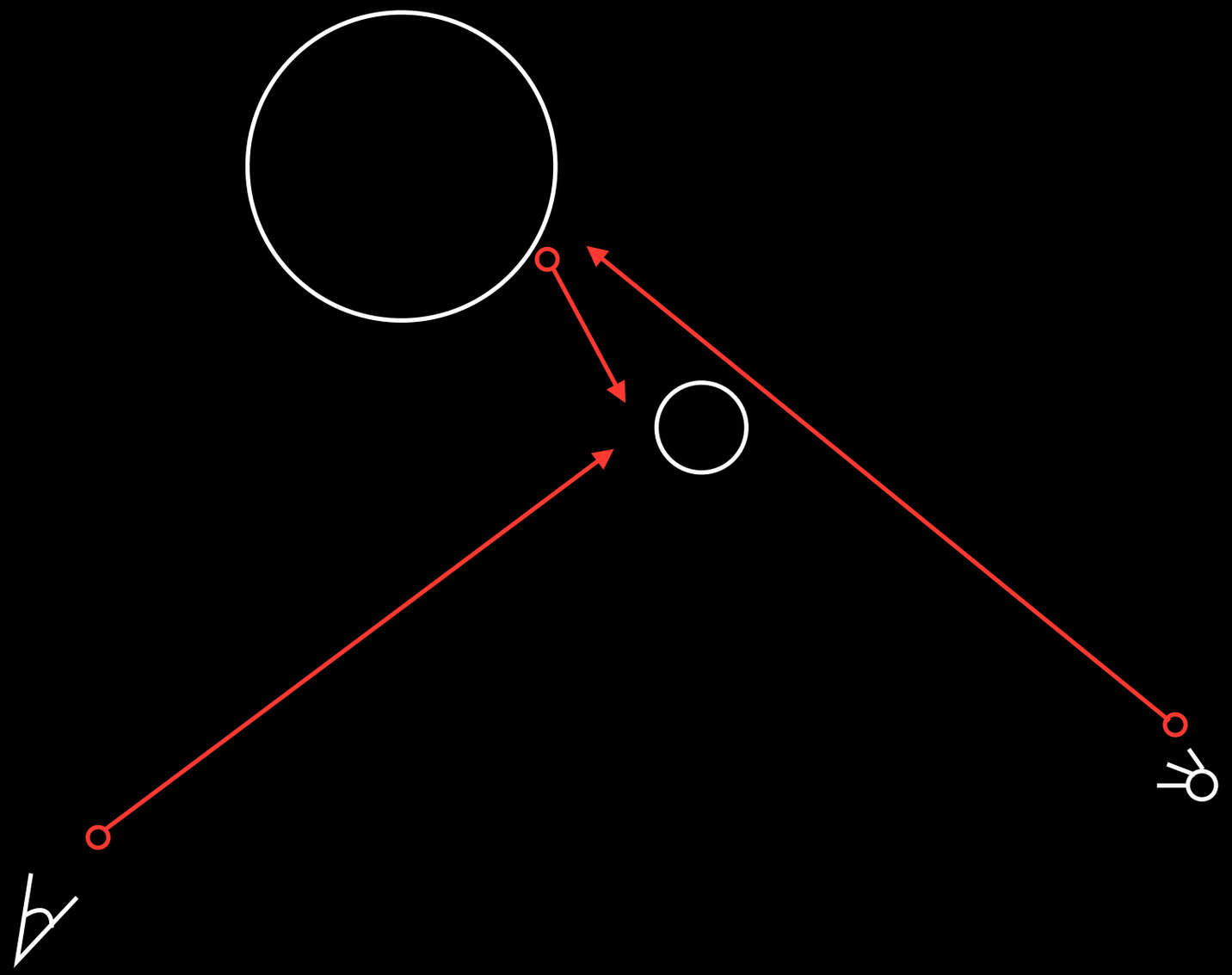
# An alternate approach



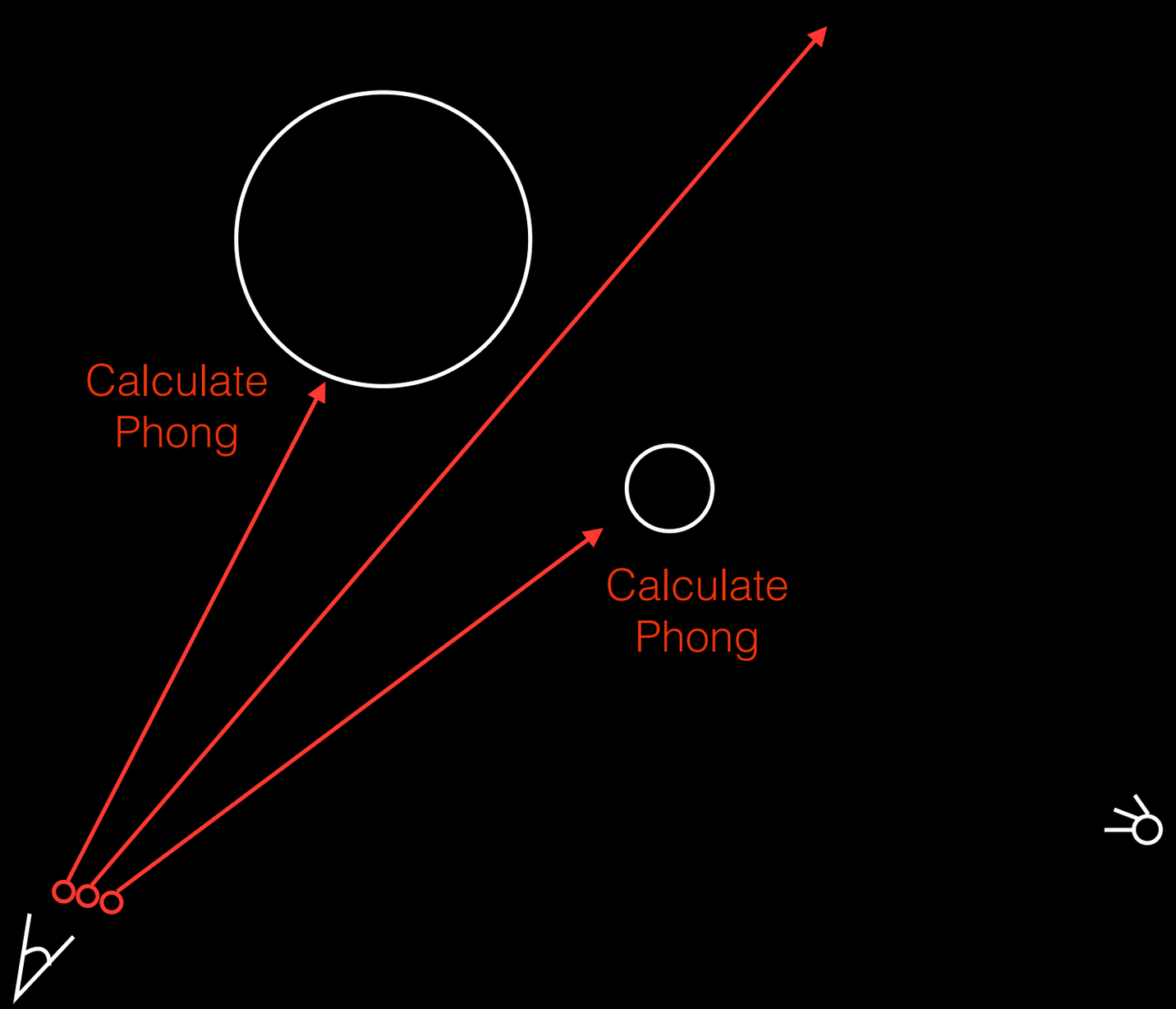
# Yet another alternate approach



# Hybrid approaches

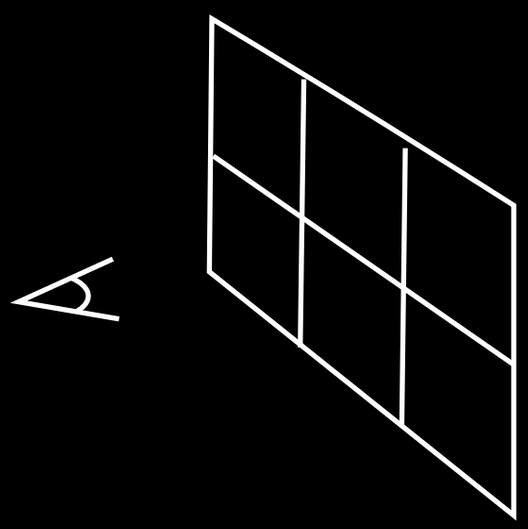


# But let's start with this

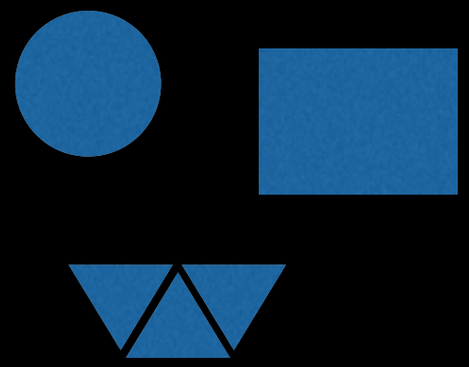


# Things you will need to implement

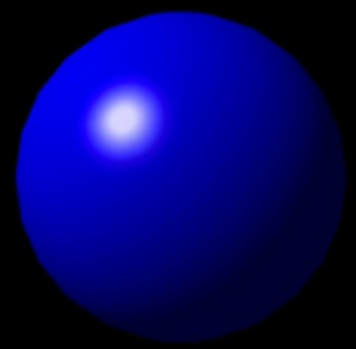
Shoot rays



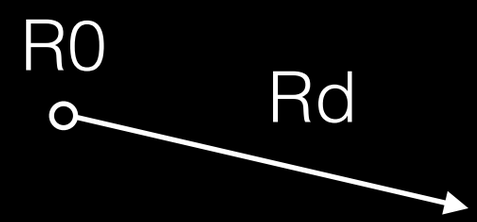

Intersect rays with objects

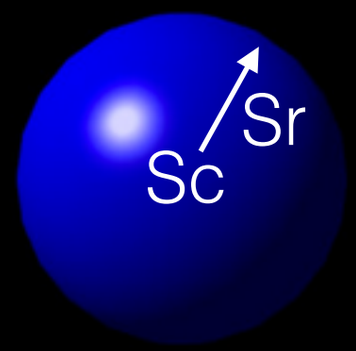


Calculate values according to illumination model



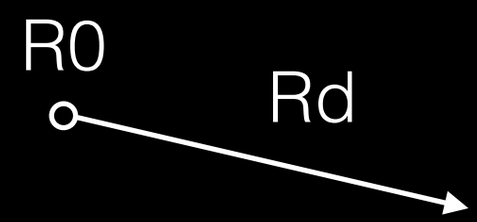
# Ray-sphere intersection



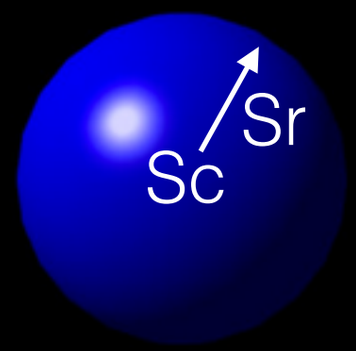




https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

# Ray-sphere intersection

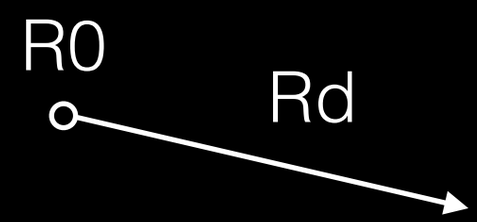



$$R(t) = R0 + t * Rd \ , \ t > 0$$



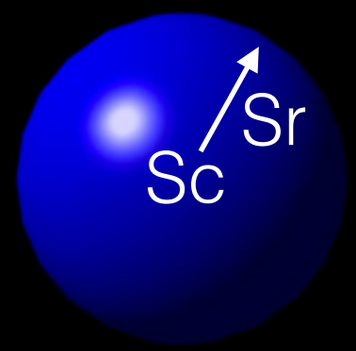

https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

# Ray-sphere intersection



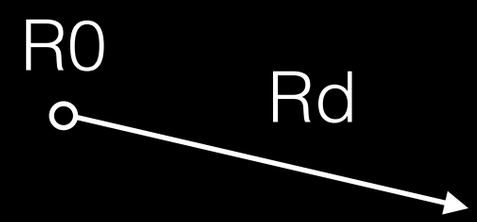

$R(t) = R0 + t * Rd , t > 0$

$x = X0 + Xd * t$
$y = Y0 + Yd * t$
$z = Z0 + Zd * t$



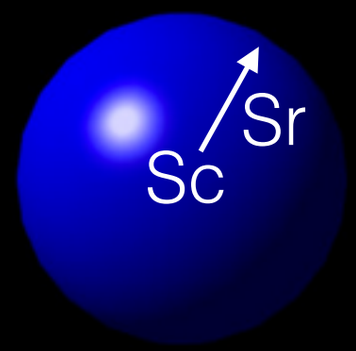

https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

# Ray-sphere intersection




$R(t) = R0 + t * Rd\ ,\ t > 0$

$x = X0 + Xd * t$
$y = Y0 + Yd * t$
$z = Z0 + Zd * t$




$$S = \{(x, y, z) \mid (x - Xc)^2 + (y - Yc)^2 + (z - Zc)^2 = Sr^2\}$$

https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

# Ray-sphere intersection

$x = X0 + Xd * t$
$y = Y0 + Yd * t$
$z = Z0 + Zd * t$

$S = \{(x, y, z) \mid (x - Xc)^2 + (y - Yc)^2 + (z - Zc)^2 = Sr^2\}$

$(X0 + Xd * t - Xc)^2 + (Y0 + Yd * t - Yc)^2 + (Z0 + Zd * t - Zc)^2 = Sr^2$

https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

# Ray-sphere intersection

$x = X0 + Xd * t$
$y = Y0 + Yd * t$
$z = Z0 + Zd * t$

$S = \{(x, y, z) \mid (x - Xc)^2 + (y - Yc)^2 + (z - Zc)^2 = Sr^2\}$

$(X0 + Xd * t - Xc)^2 + (Y0 + Yd * t - Yc)^2 + (Z0 + Zd * t - Zc)^2 = Sr^2$

A quadratic equation: $A*t^2 + B*t + C = 0$

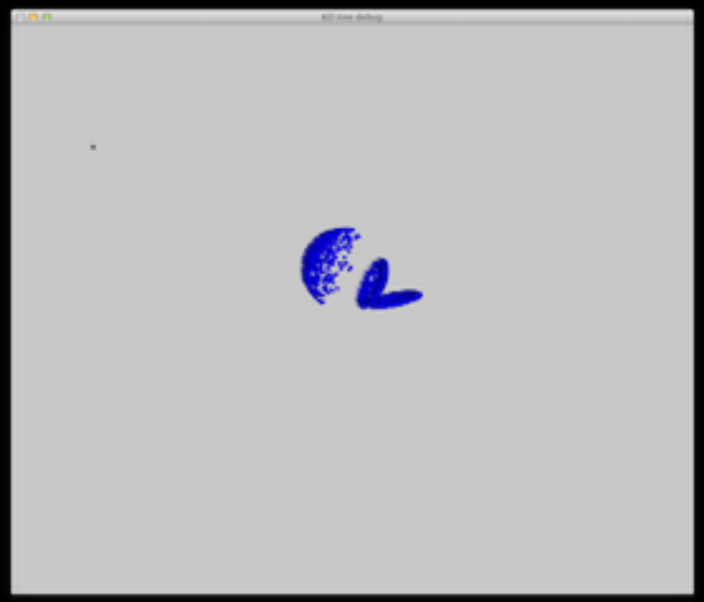
https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

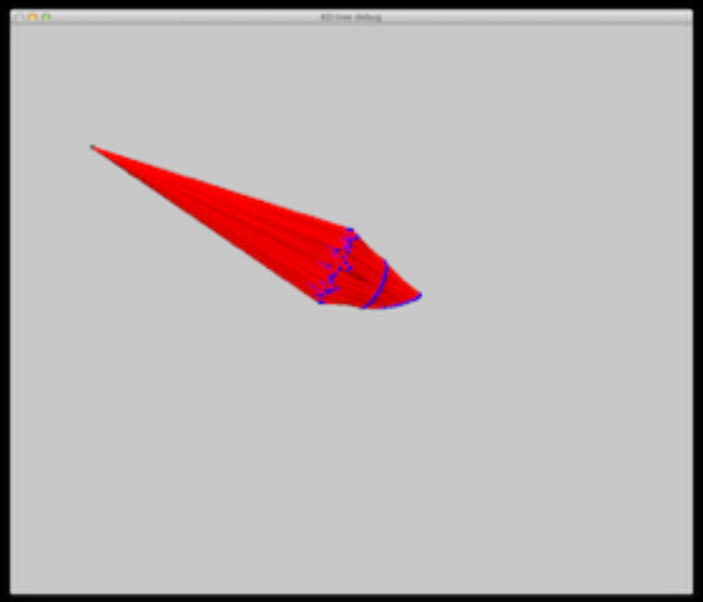
Debugging:
(2) Primary ray visualization: Provide code that will produce an image showing line segments indicating the paths of primary rays starting at the camera eye point and stopping at the first surface intersections. This option could be implemented by writing a .scn file with a representation 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.

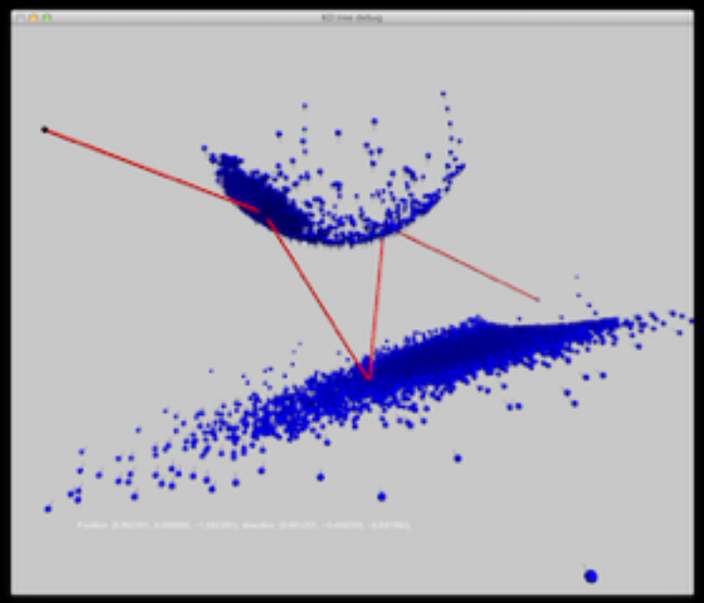
What’s wrong here?

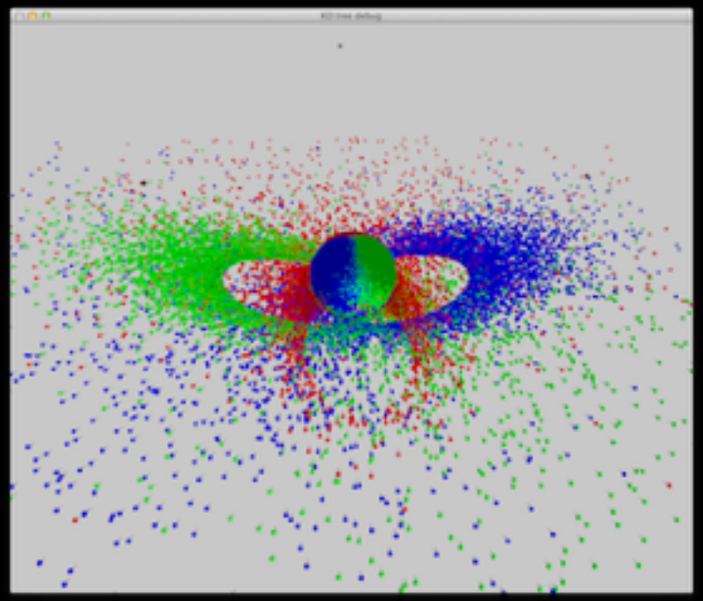
Debugging:
**(2) Primary ray visualization:** Provide code that will produce an image showing line segments indicating the paths of primary rays starting at the camera eye point and stopping at the first surface intersections. This option could be implemented by writing a .scn file with a representation 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.

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.

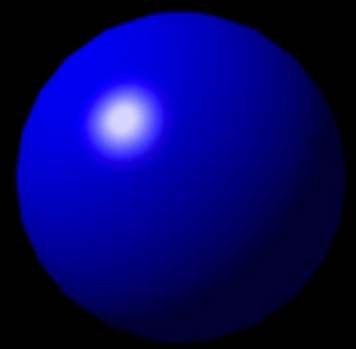




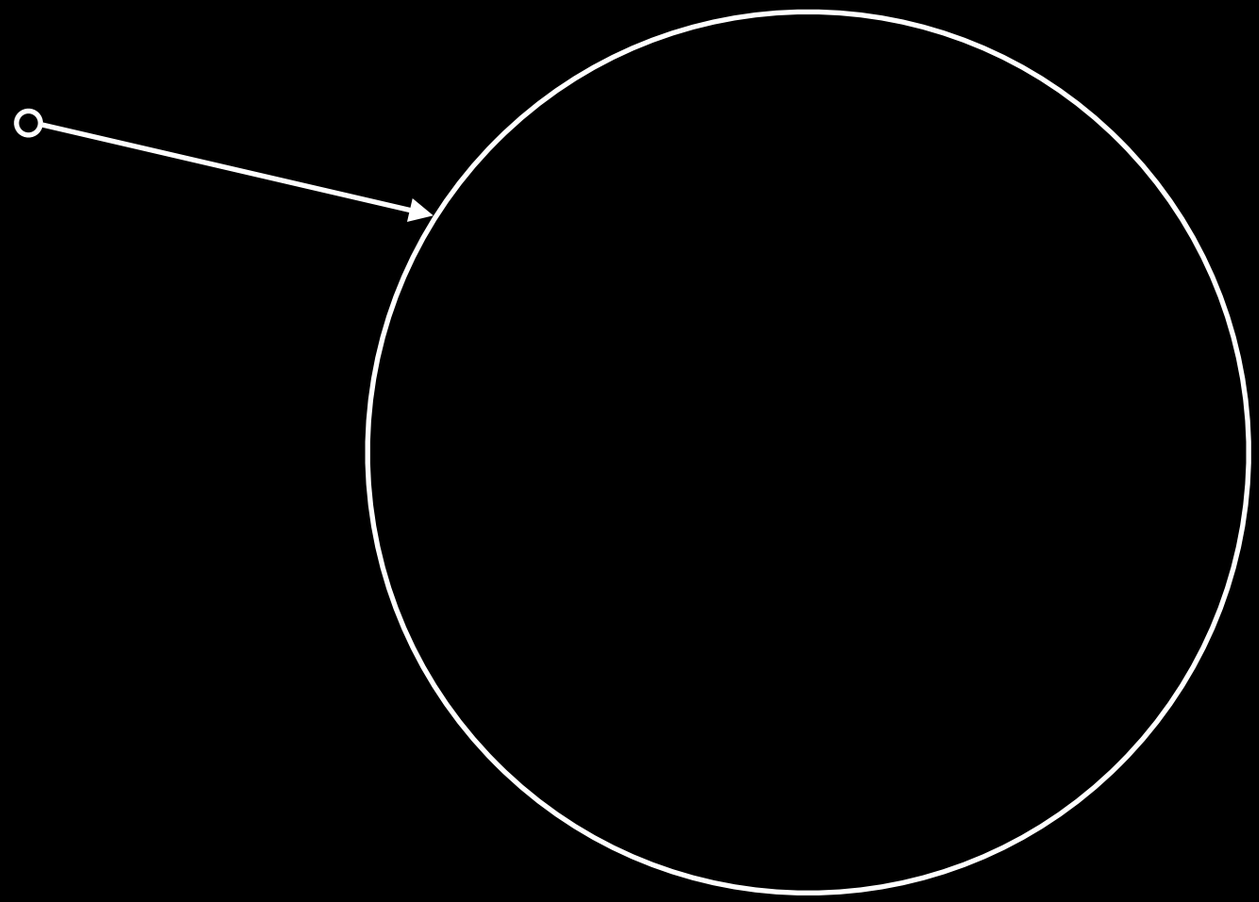




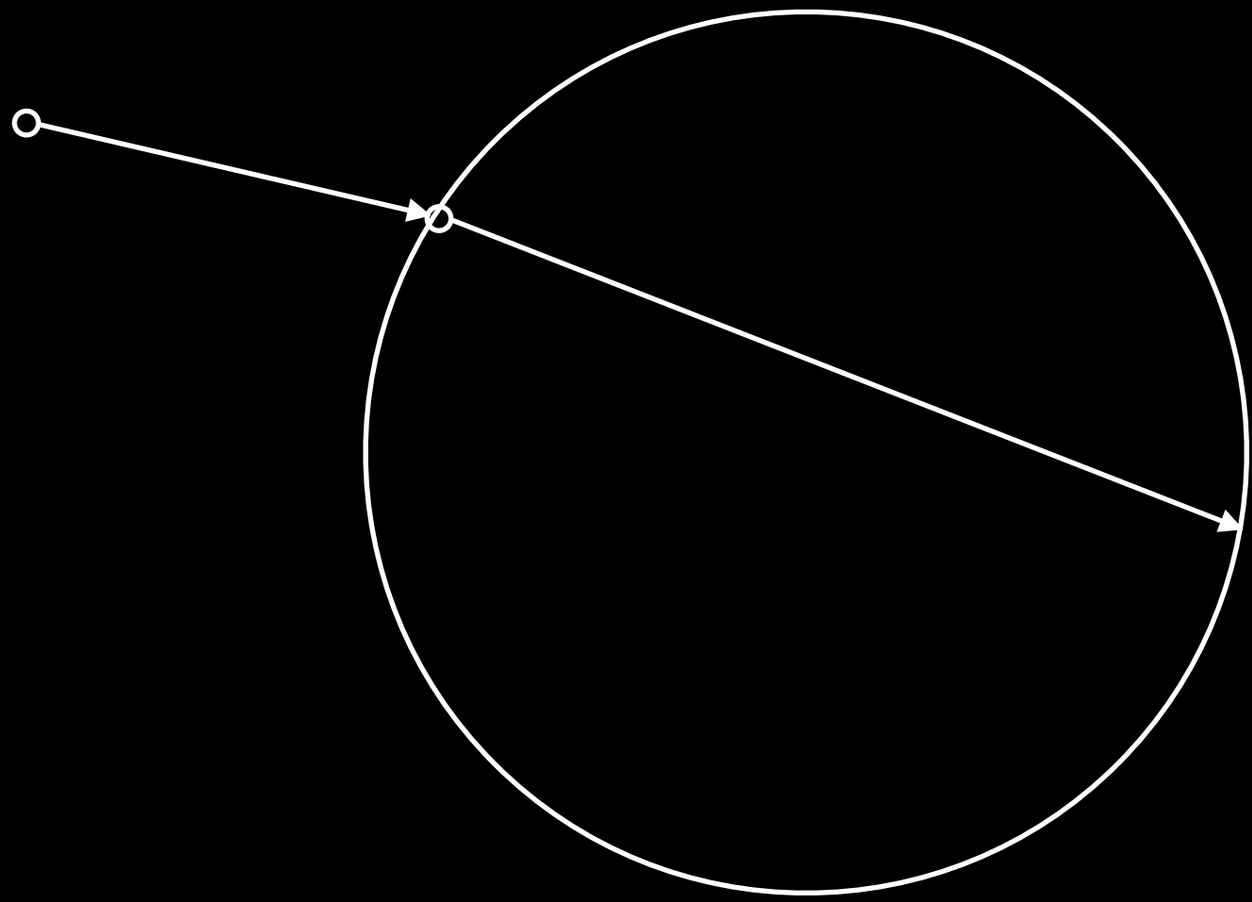
# Common pitfalls



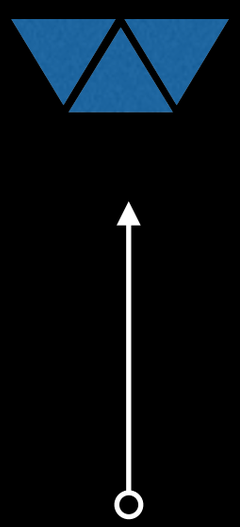
# Common pitfalls



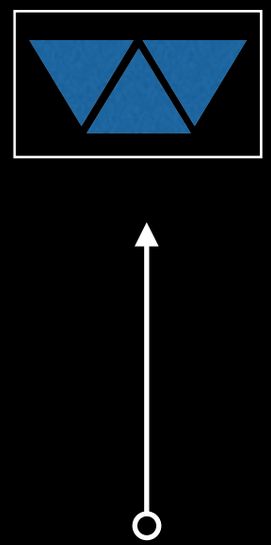
# Common pitfalls



# Acceleration

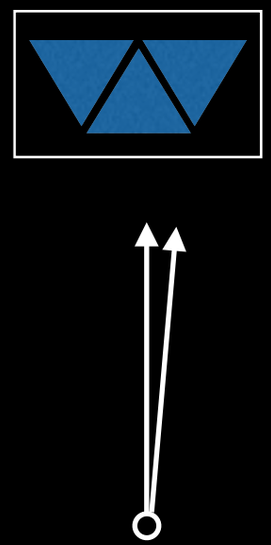


# Acceleration



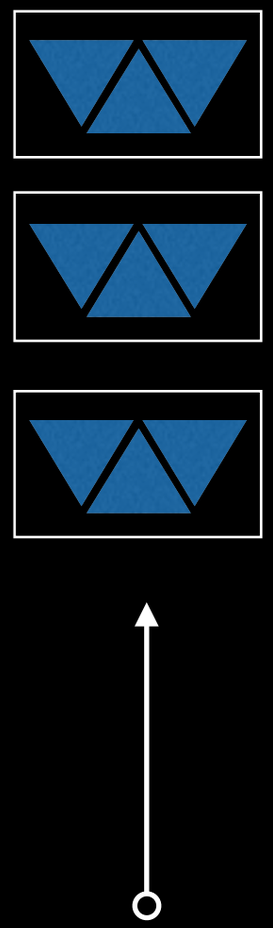
Test bounding box

# Acceleration



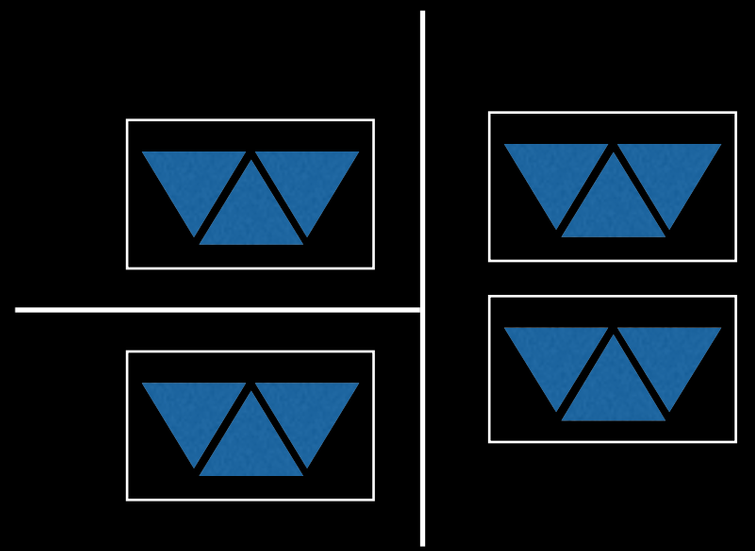
Cache last intersection

# Acceleration



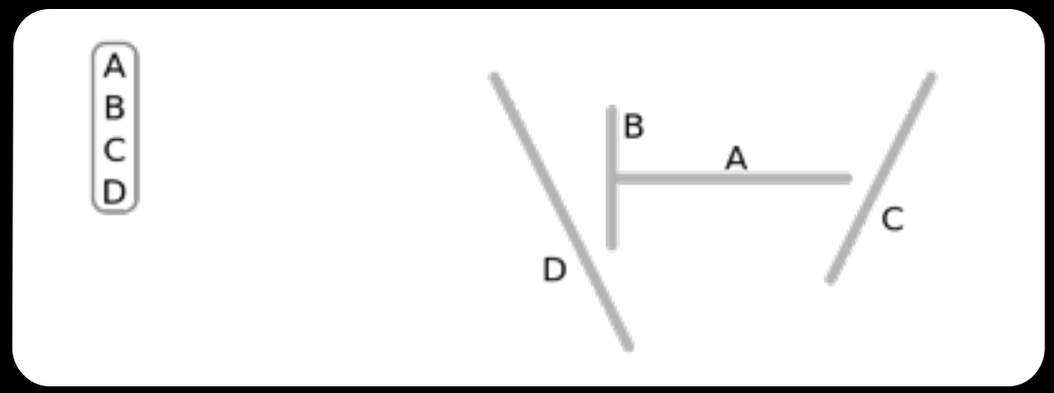
Front to back

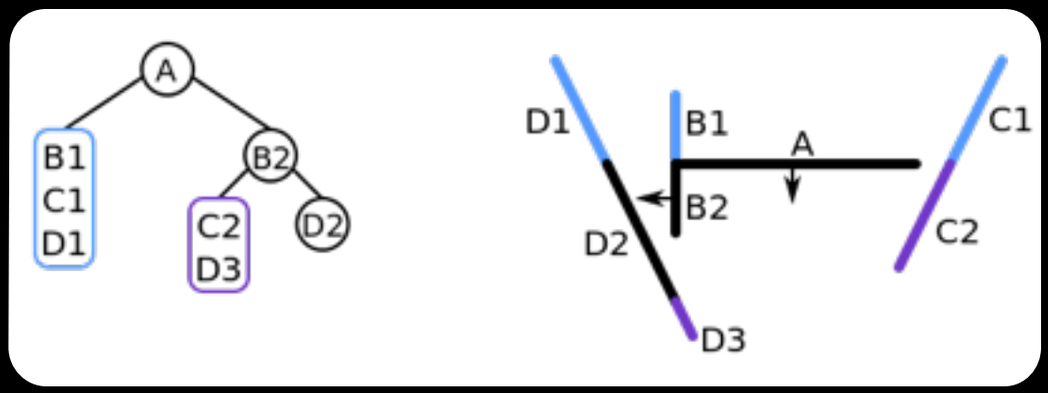
# Acceleration

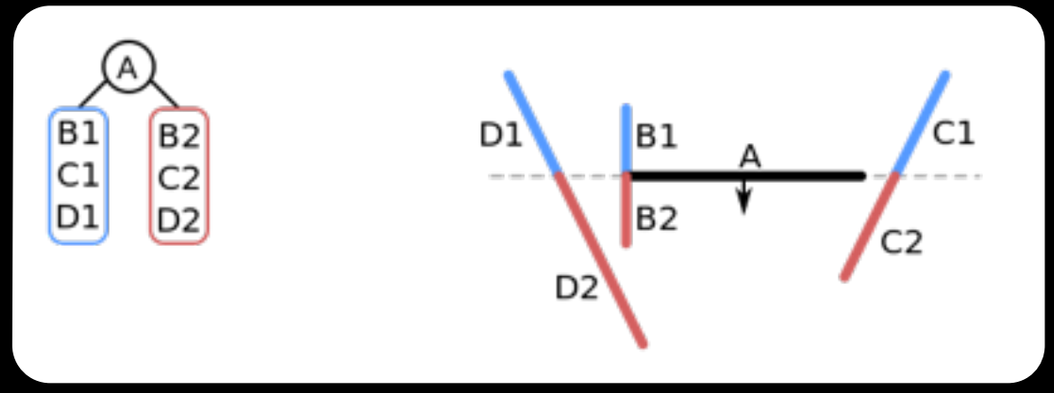


Use data structures

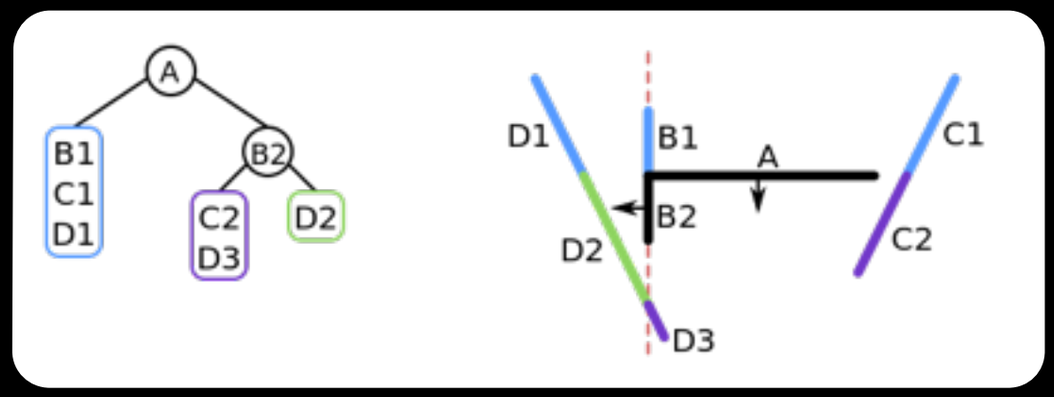
# BSP Tree

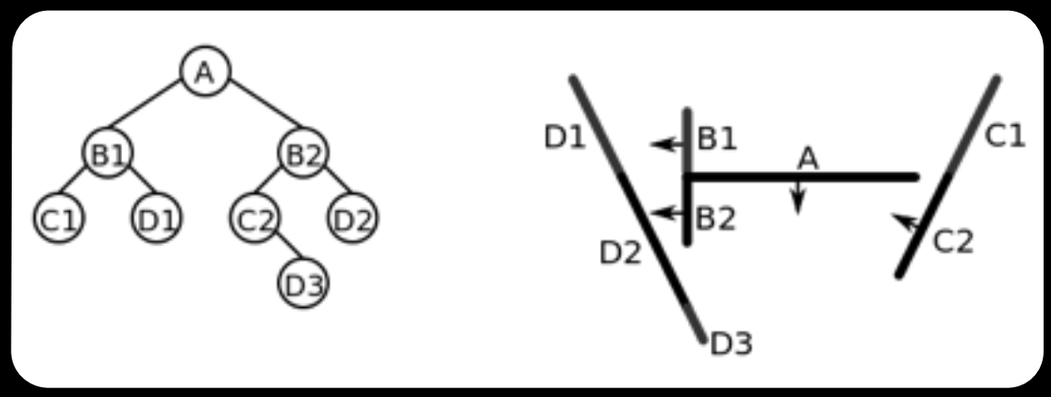


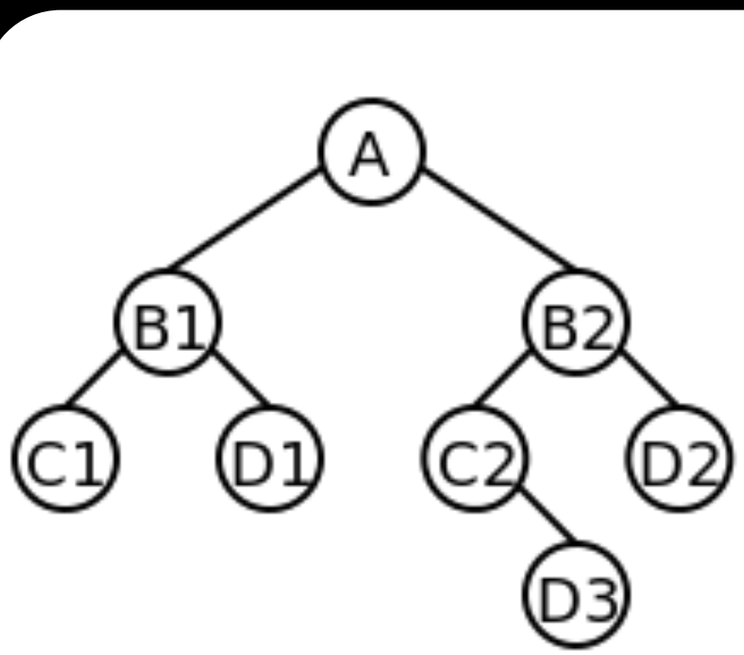

⋮

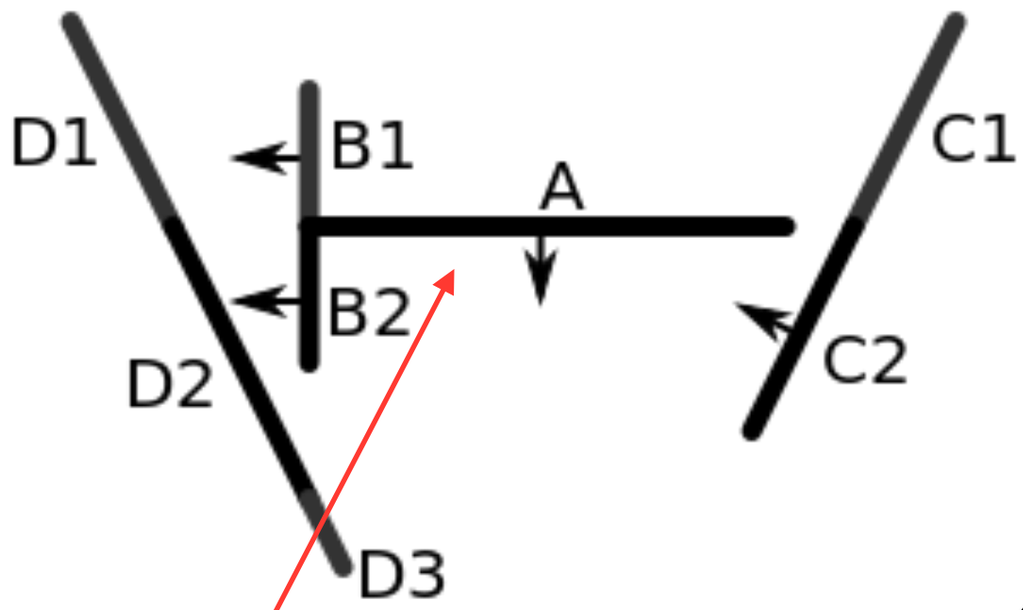



http://en.wikipedia.org/wiki/Binary_space_partitioning

# BSP Tree




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