IT UNIVERSITY OF COPENHAGEN

Second Year Project

Writing your own ray tracer

IT UNIVERSITY OF COPENHAGEN

Happy Easter

Resolution
1000x1000

Render time
6.8 seconds

Second Year Project 2016

Jesper Bengtson

Why a ray tracer?

- Pretty pictures
- Used in industry (Pixar, Disney, GPUs)
- A solid project from scratch
- The extension possibilities are close to endless
- It is fun :)

A ray tracer in F#

- Implicit surfaces (spheres, tori, etc)
- Triangle meshes (Stanford Bunny)
- Optimisation (Parallelisation and acceleration structures)
- Object Composition (hemispheres, objects with bevelled edges, etc)
- Textures and Reflection
- Object modification (rotation, skewing, scaling, displacement)

Implicit surface

Implicit surface

Triangle mesh
69 451 faces
35 947 vertices

Implicit surface

Triangle mesh
69 451 faces
35 947 vertices

Reflection (5x)

Implicit surface

Objects have been scaled, rotated and translated to fit the scene

Triangle mesh
69 451 faces
35 947 vertices

Reflection (5x)

IT UNIVERSITY OF COPENHAGEN

Implicit surface

Objects have been scaled, rotated and translated to fit the scene

1000 x 1000 pixels rendered in 6.8 seconds (kd-trees and axis-aligned bounding boxes)

Triangle mesh
69 451 faces
35 947 vertices

Reflection (5x)

The project

- Write a ray tracer in groups of six (in rare cases seven)
- You will be given a minimal API that you must wrap your tracer in
  - This helps us grade
  - This allows us to give you plenty of test files that you can render
  - You should still internally use more elaborate interfaces

The project

- Write a ray tracer in groups of six (in rare cases seven)
- [redacted]

This API is subject to change!!!

  - You should still internally use more elaborate interfaces

Examination

- Hand in a working ray tracer that runs on Windows 10 and a report
- 16 minute oral examination
  - 8 minute for the entire group
  - 8 minute individual
- Grades in standard seven point scale
  - You will be graded on both the project and on on your individual performance

Who are we?

Jesper Bengtson

- Associate professor at ITU since 2013
- Member of the Program Logics and Semantics group
- Researches formal verification of software
  - Java
  - x86 assembly

Who are we?

Patrick Bahr

- Postdoc at ITU since 2015
- Member of the Program Logics and Semantics group
- Researches type theory and formal verification

Contact information

Jesper

- email: jebe@itu.dk
- office: 4C10

Patrick

- email: paba@itu.dk
- office: 4C15

Teaching Assistants

Two of the teaching assistants from Functional Programming will stay on board

Jakob Merrild

- email: jmer@itu.dk

Mikkel Bybjerg Christophersen

- email: mbyb@itu.dk

They will have office hours every Tuesday and Thursday

Teaching Assistants

Two of the teaching assistants from Functional Programming will stay on

Teaching Assistants are not your debuggers

They will have office hours every Tuesday and Thursday

The hacking order

- For any course administrative issues come to me
- For any theoretical questions came to me or Patrick (which theory goes to whom will be made apparent)
- For any coding related problems talk to the TAs

The hacking order

- For any course administrative issues come to me
- For any theoretical questions came to me or Patrick (which theory goes to whom will be made apparent)
- For any coding related problems talk to the TAs

You may, of course always talk to any of us

Rough Schedule

- This course is heavily front loaded with theoretical prerequisites
  - This week I am teaching
  - Next two weeks Patrick is teaching (I am abroad, but available on Skype and e-mail)
- Lectures will decrease significantly once the theory is done but we will meet each group regularly

This week

- Ray tracing basics
  - vectors, points, rays, scenes, shading
- Colour handling
- Implicit surfaces
- Constructive Solid Geometry
- Affine transformations

Next week

- PLY-files
- Textures
- Acceleration Structures

IT UNIVERSITY OF COPENHAGEN

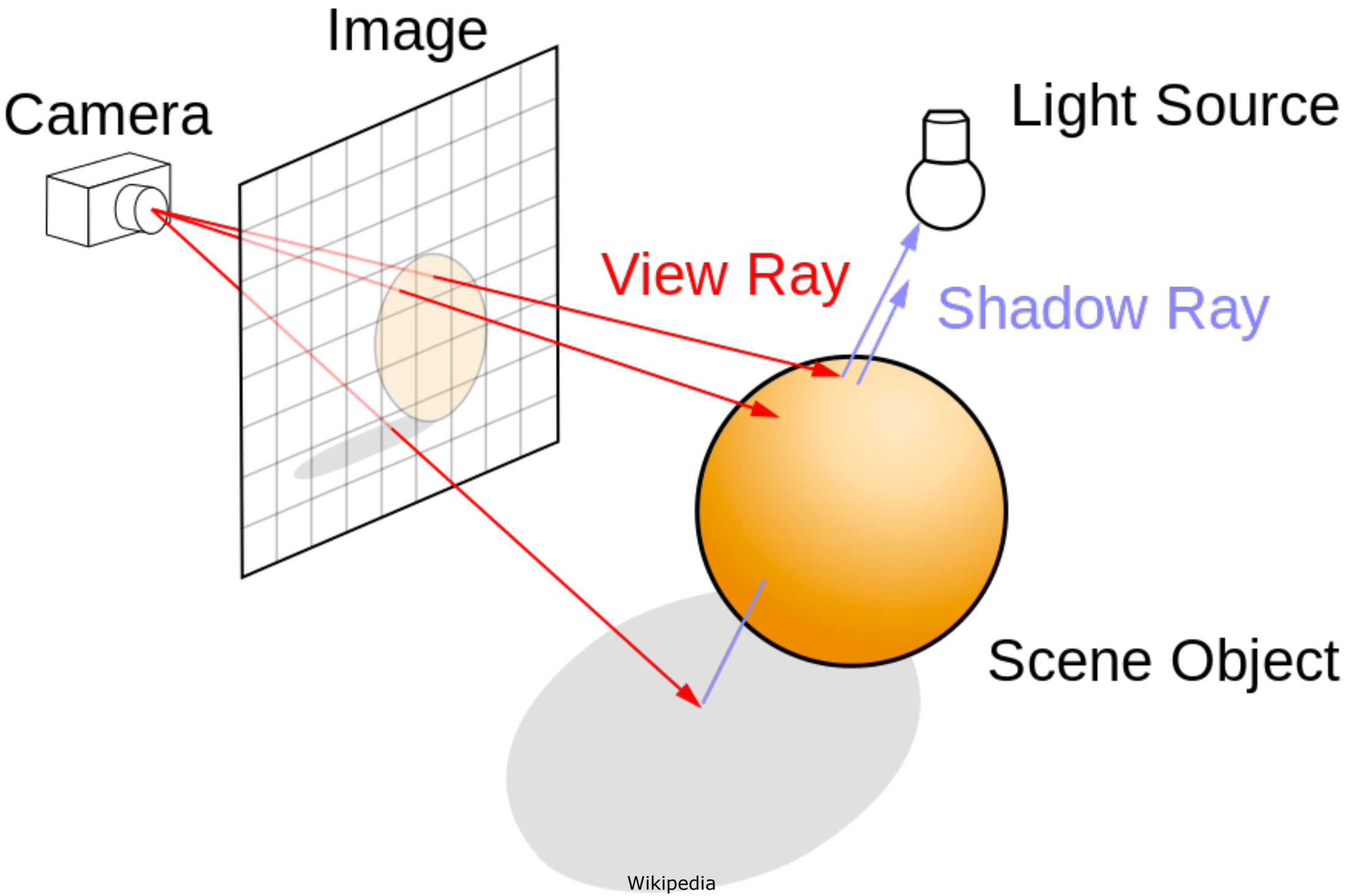
Questions?

IT UNIVERSITY OF COPENHAGEN

Ray Tracing

The basics

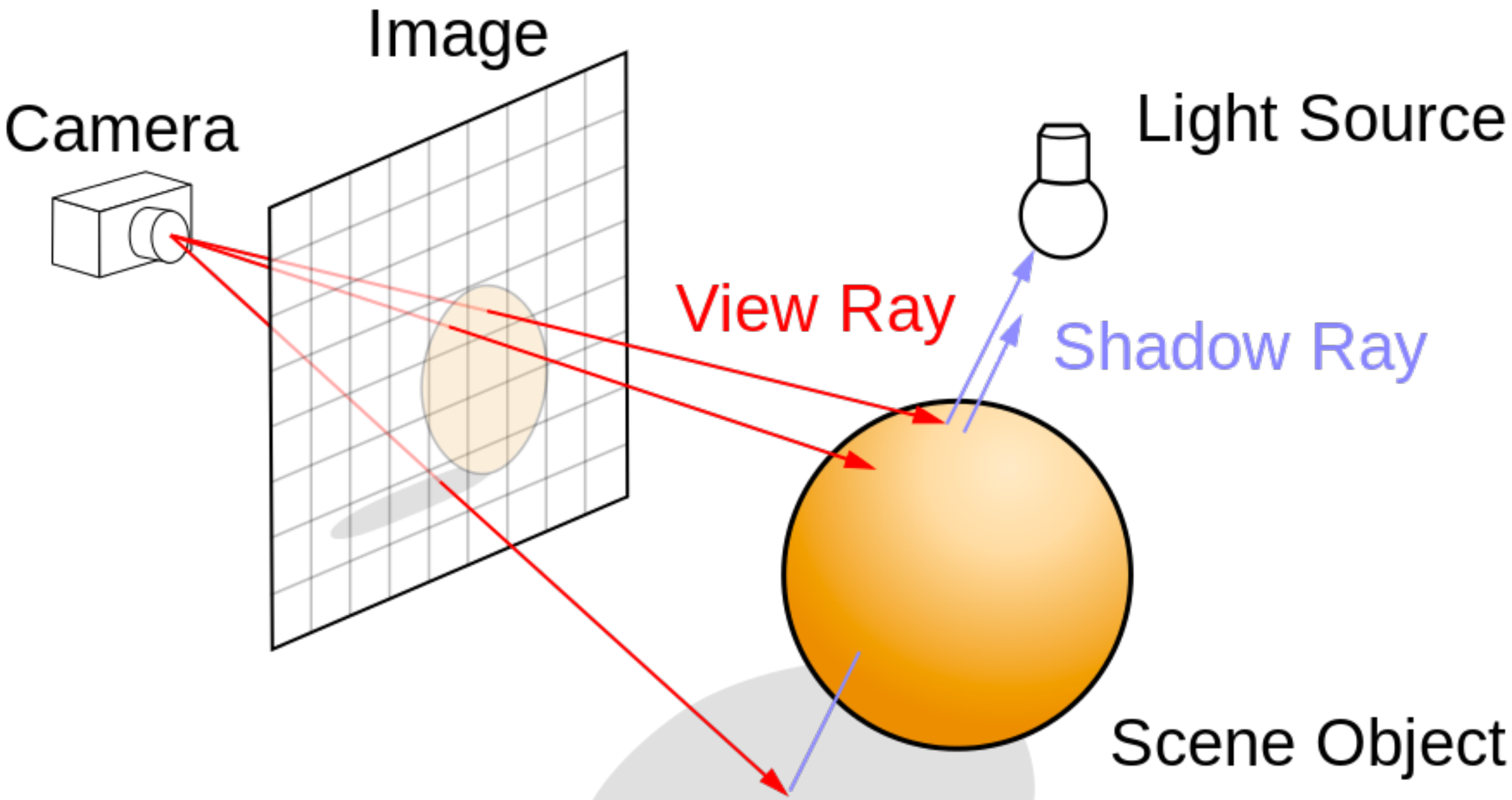
Ray tracing basics



Wikipedia

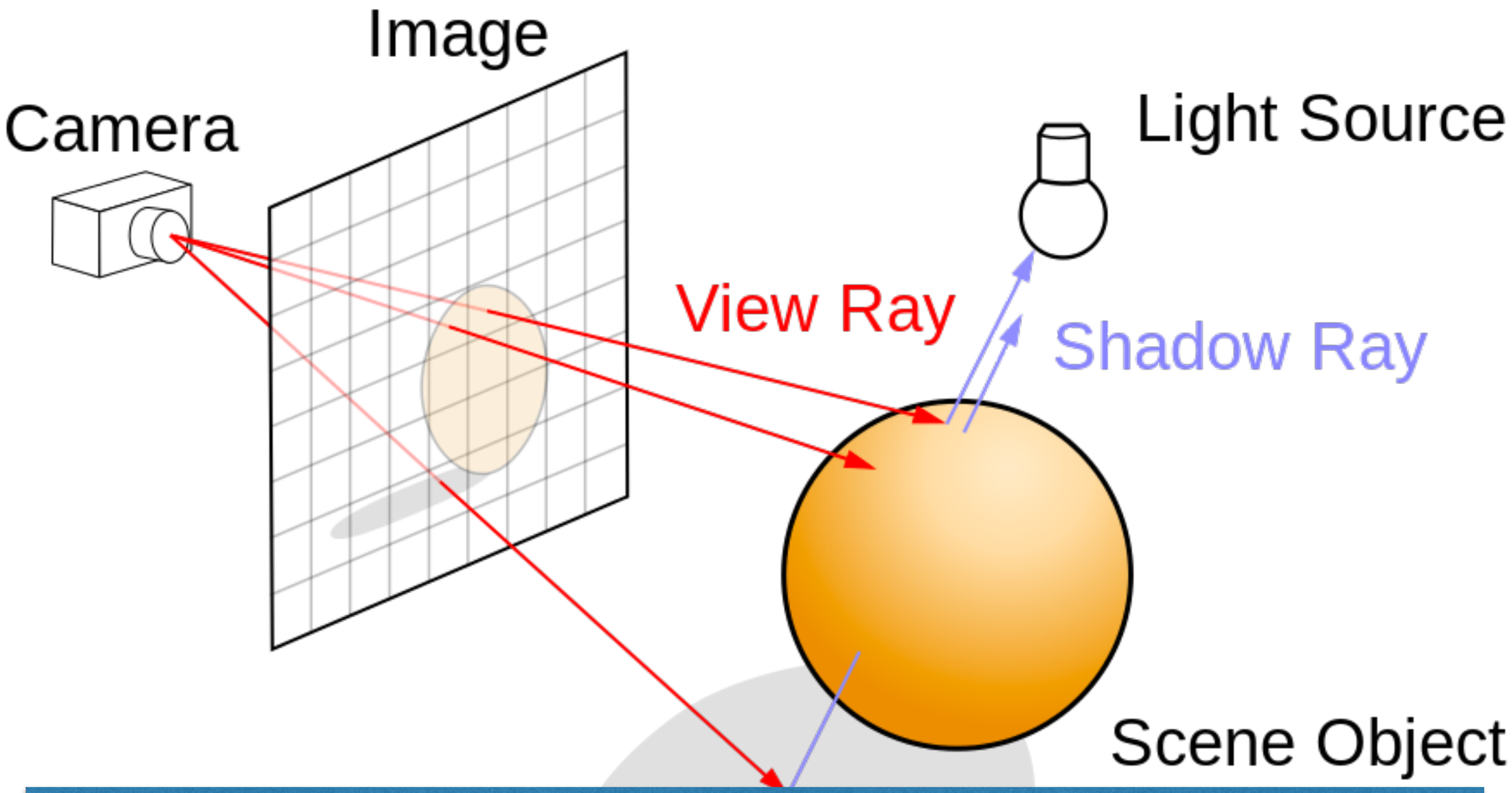
IT UNIVERSITY OF COPENHAGEN

Ray tracing basics



We render a three dimensional scene on a two dimensional plane

Ray tracing basics

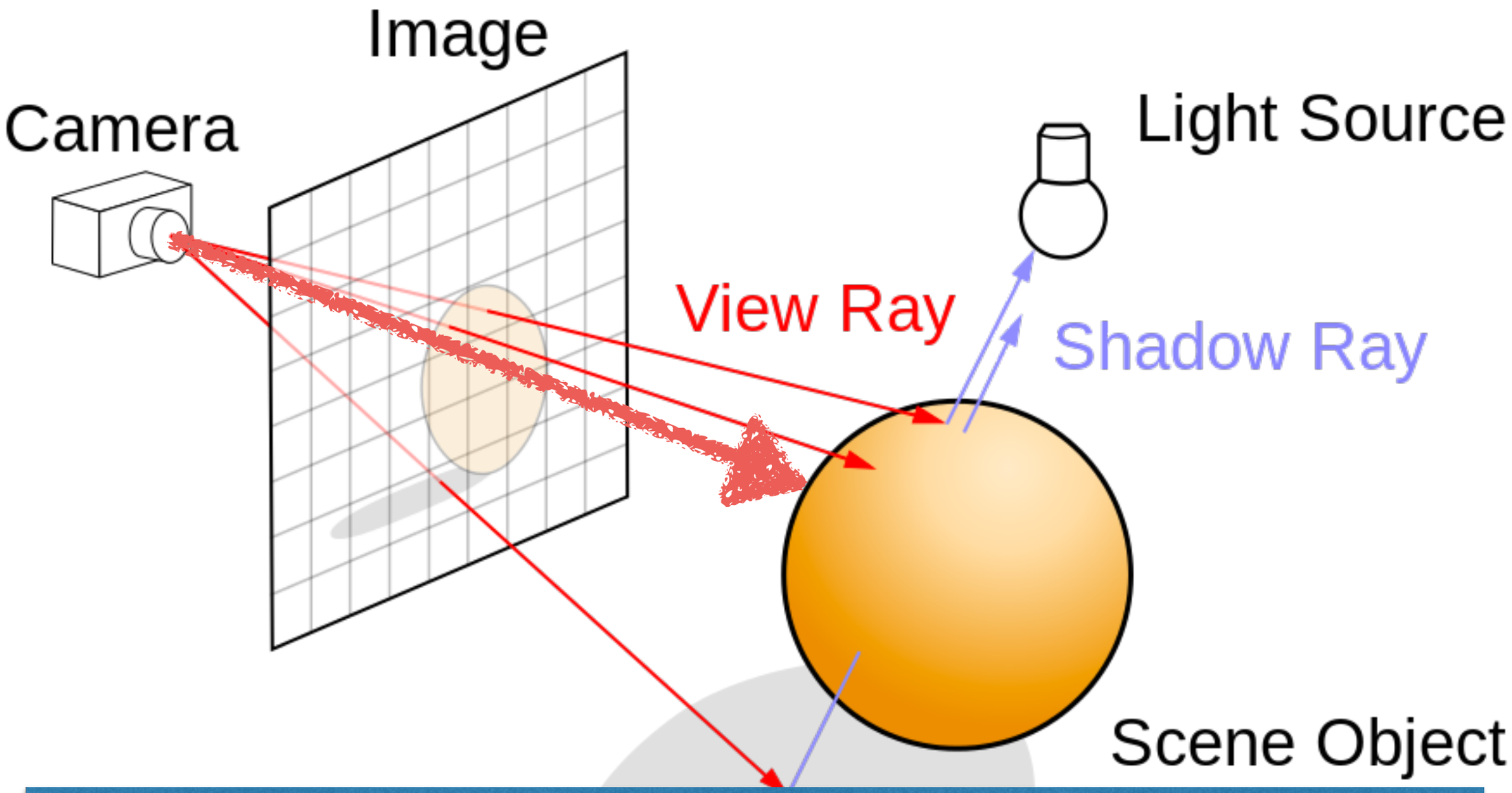


For every pixel we trace a ray into the scene to check the closest object we hit

Wikipedia

IT UNIVERSITY OF COPENHAGEN

Ray tracing basics

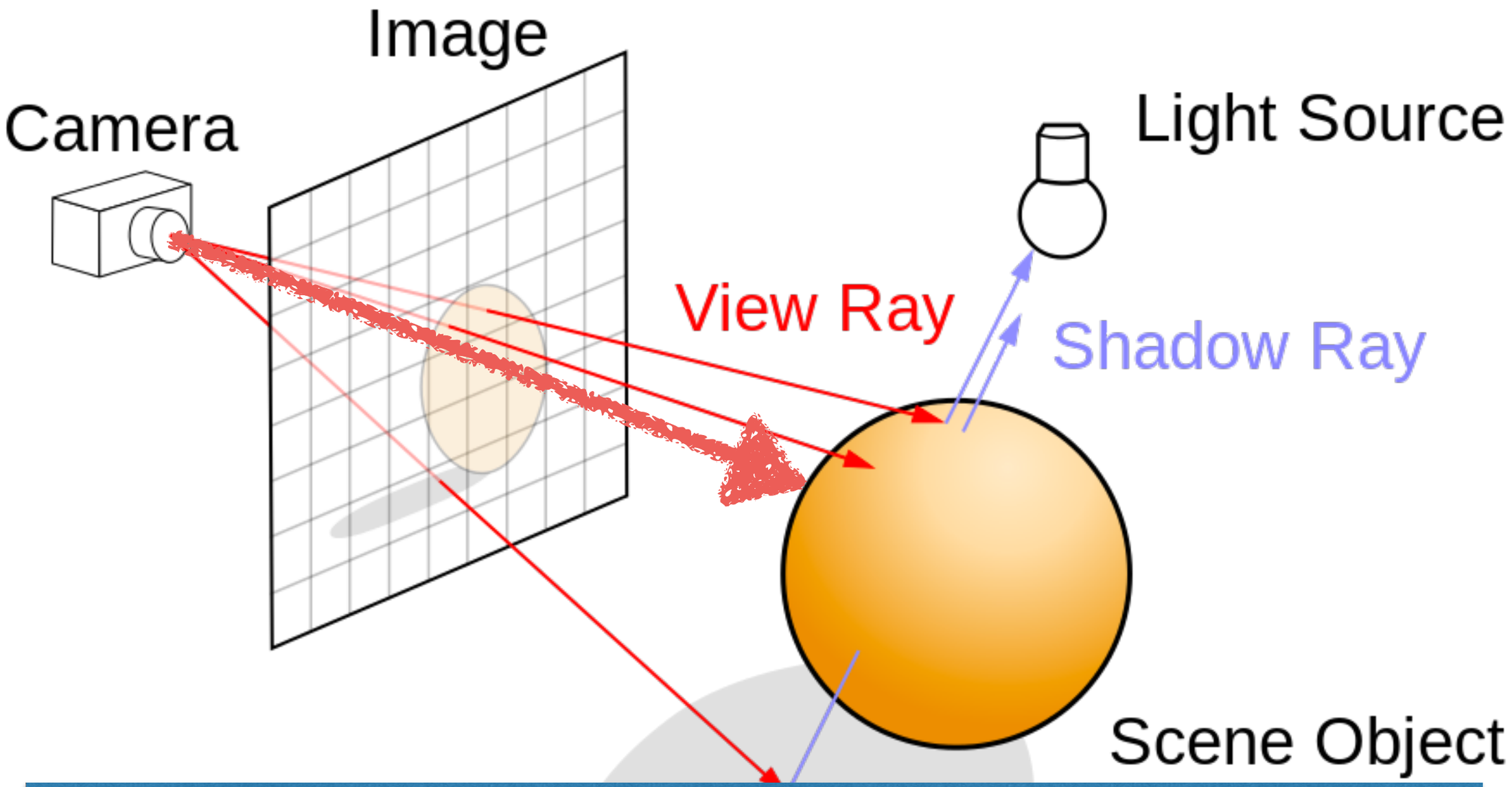


For every pixel we trace a ray into the scene to check the closest object we hit

Wikipedia

IT UNIVERSITY OF COPENHAGEN

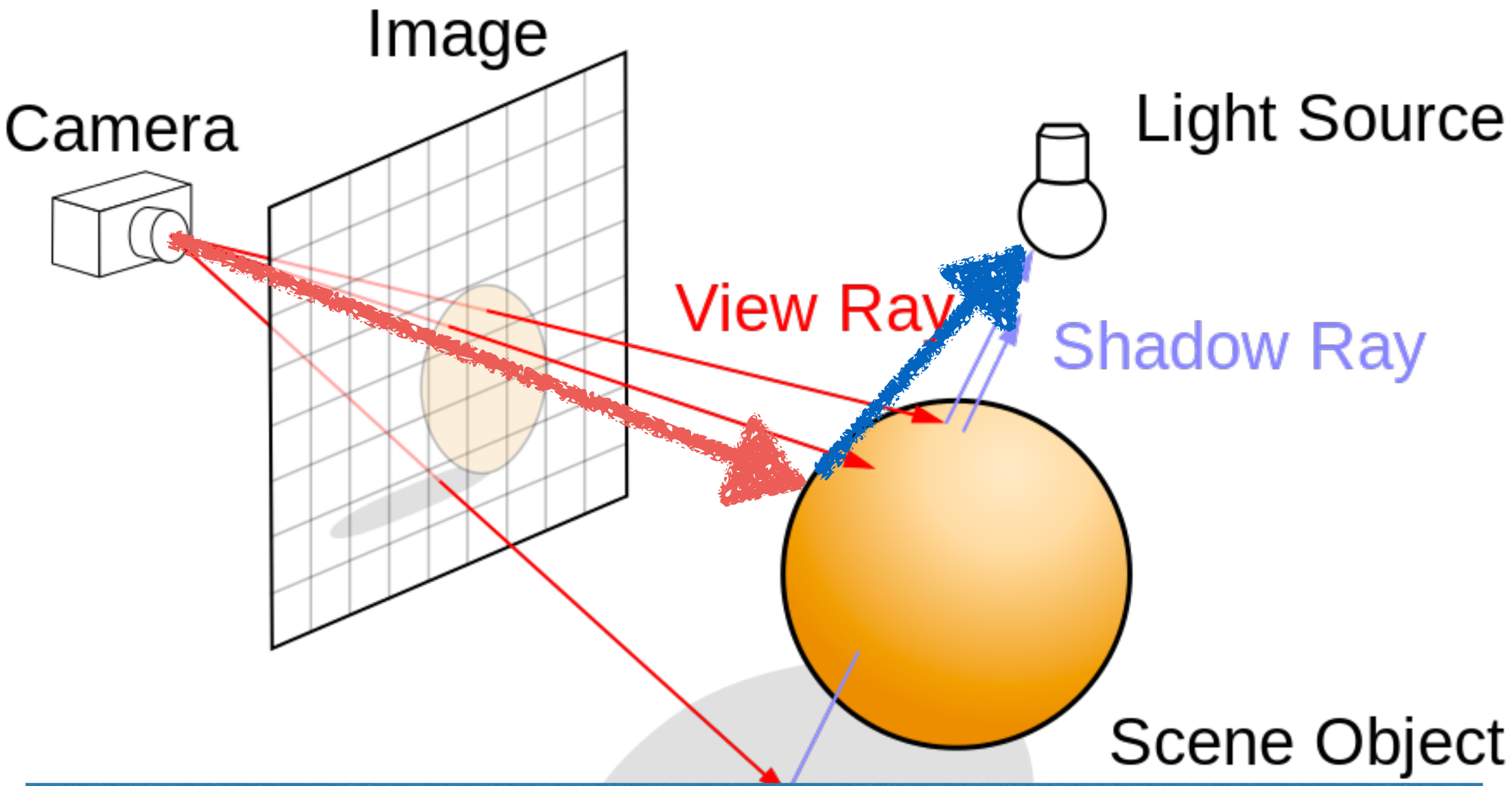
Ray tracing basics



For every hit we trace a shadow ray to every light source in the scene

IT UNIVERSITY OF COPENHAGEN

Ray tracing basics



For every hit we trace a shadow ray to every light source in the scene

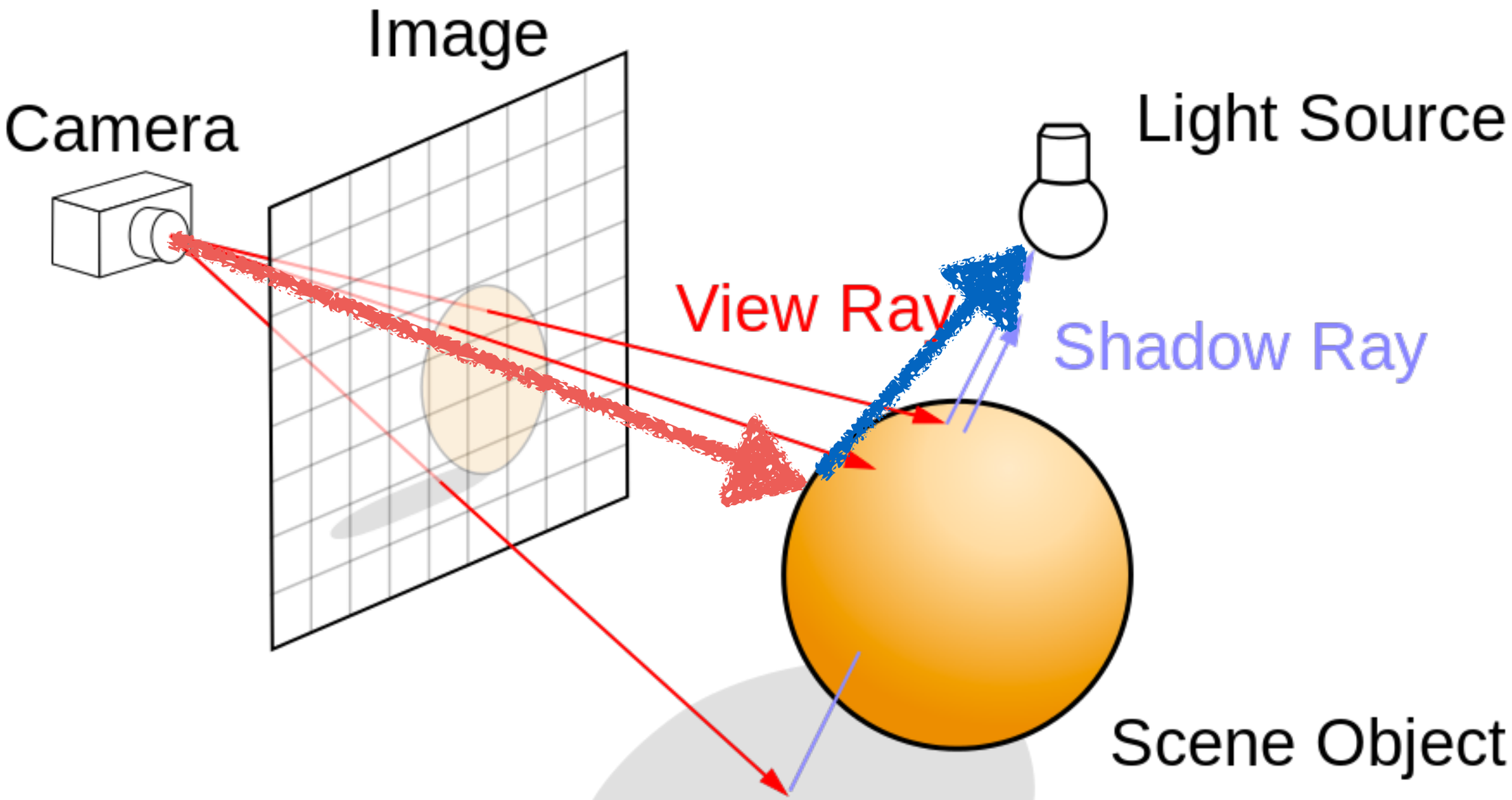
Ray tracing basics

The colour of the pixel depends on

1. The colour of the object
2. The colour of the lights
3. The intensity of the lights
4. Any ambient light
5. The angle that light hits the object
6. Any other objects mirrored in the object
7. Possibly many other things

IT U UNIVERSITY OF COPENHAGEN

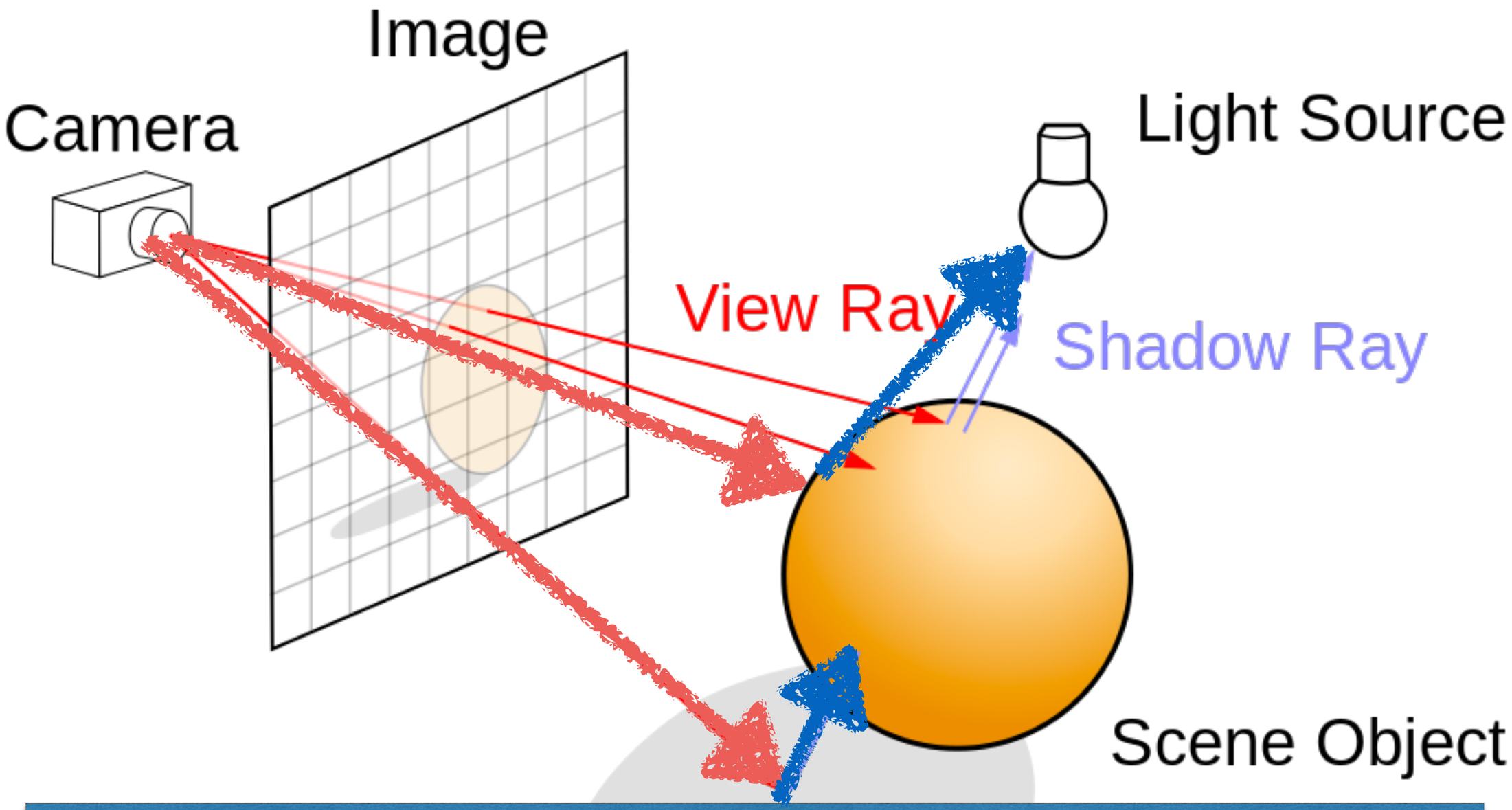
Ray tracing basics



If a shadow ray hits no light that pixel is not coloured (in simple cases)

IT U UNIVERSITY OF COPENHAGEN

Ray tracing basics



If a shadow ray hits no light that pixel is not coloured (in simple cases)

We need the following

- Three dimensional vectors
- Points in three dimensional space
- A camera (position, direction, resolution)
- Colours (Standard RGB)
- Lights (position, colour, and intensity)
- A shape interface (have I been hit?)
- A scene (A collection of shapes, lights, and a camera)

We need the following

- Three dimensional vectors
- Points in three dimensional space
- A camera (position, direction, resolution)

- Colours (Standard RGB)
- Lig tensity)
- A shape interface (have I been hit?)
- A scene (A collection of shapes, lights, and a camera)

We start here

Vectors

Vectors consist of a x-, a y-, and a z-coordinate in 3D space

Vectors represent a direction (not a position) 3D space

In ray tracing we use vectors to calculate how light travels over a scene

Vector Addition

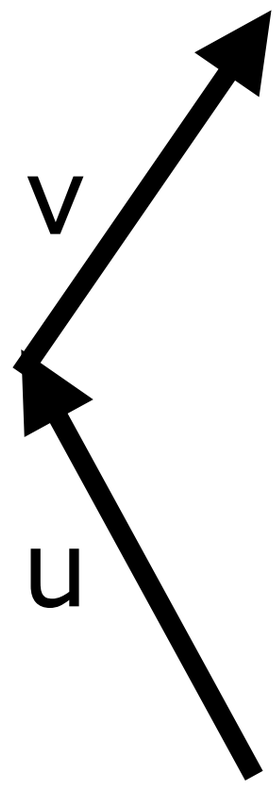
We add vectors by pointwise adding their components

{a, b, c}+{d, e, f} = {a+d, b+e, c+f}

Vector Addition

We add vectors by pointwise adding their components

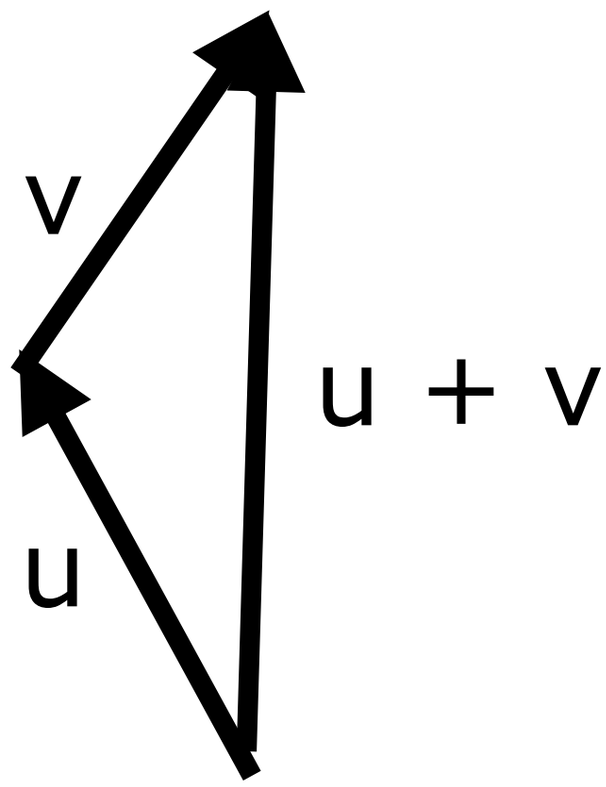
$$\{a, b, c\}+\{d, e, f\} = \{a+d, b+e, c+f\}$$



Vector Addition

We add vectors by pointwise adding their components

{a, b, c}+{d, e, f} = {a+d, b+e, c+f}



Vector Subtraction

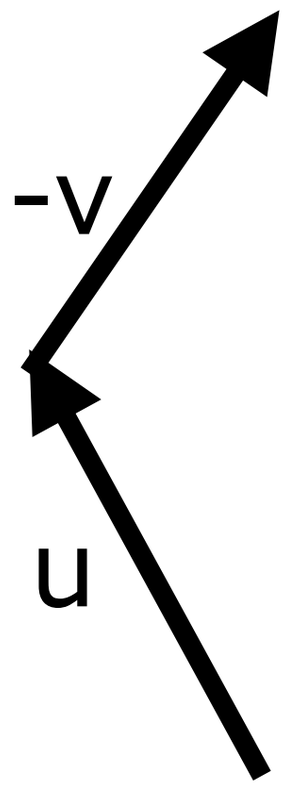
We subtract vectors by pointwise subtracting their components

{a, b, c}-{d, e, f} = {a-d, b-e, c-f}

Vector Subtraction

We subtract vectors by pointwise subtracting their components

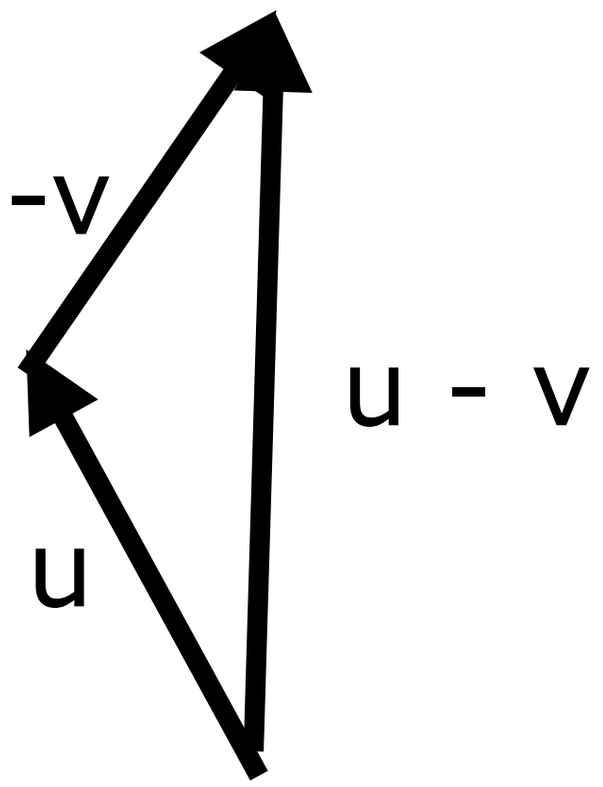
{a, b, c}-{d, e, f} = {a-d, b-e, c-f}



Vector Subtraction

We subtract vectors by pointwise subtracting their components

{a, b, c}-{d, e, f} = {a-d, b-e, c-f}



Scalar multiplication

We can lengthen or shorten a vector without changing its direction by multiplying with a scalar

{a, b, c} * x = {a * x, b * x, c * x}

Vector magnitude

The magnitude is the same as the length of the vector

$$|\{a, b, c\}| = \sqrt{a^2 + b^2 + c^2}$$

Normalised vectors

A normalised vector is a vector with the length of one

$$\{a, b, c\}\hat{} = \{a/X, b/X, c/X\}$$
$$\text{where } X = |\{a, b, c\}|$$

Normalised vectors

A normalised vector is a vector with the length of one

$$\{a, b, c\}\hat{} = \{a/X, b/X, c/X\}$$
$$\text{where } X = |\{a, b, c\}|$$

We frequently use normalised vectors when we are interested in only direction and not distance

Cross product

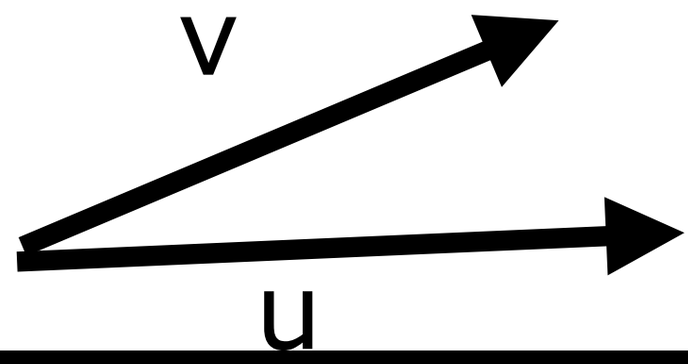
The cross product of two vectors is a vector that is perpendicular to the plane formed by those vectors

{a, b, c} x {d, e, f} =
{be - cf, cd - af, ae - bd}

Cross product

The cross product of two vectors is a vector that is perpendicular to the plane formed by those vectors

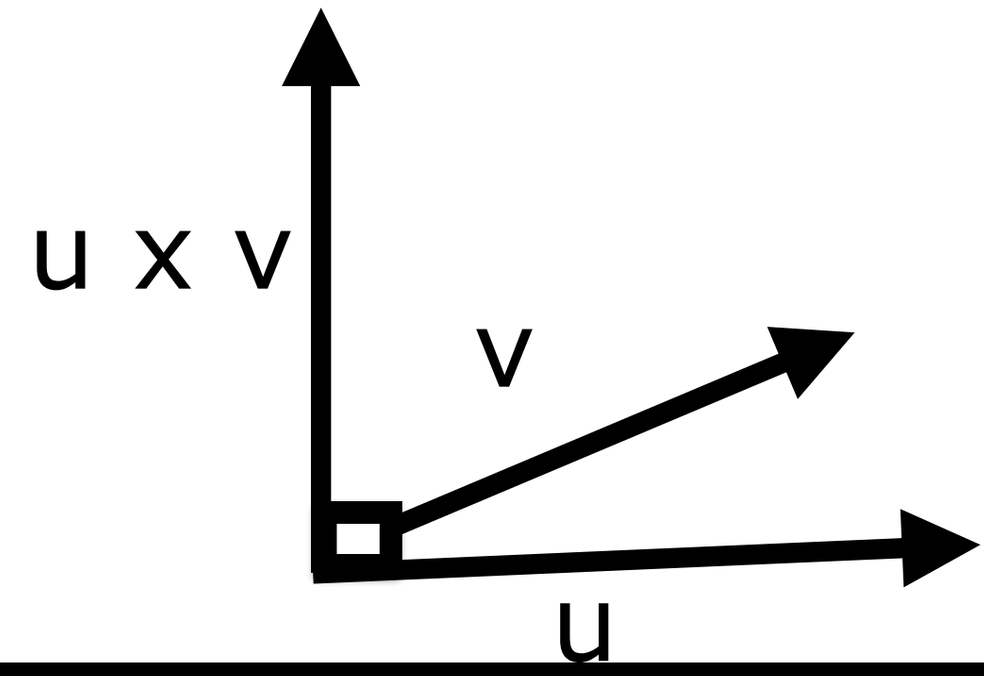
$$\{a, b, c\} \times \{d, e, f\} = \{be - cf, cd - af, ae - bd\}$$



Cross product

The cross product of two vectors is a vector that is perpendicular to the plane formed by those vectors

$$\{a, b, c\} \times \{d, e, f\} = \{be - cf, cd - af, ae - bd\}$$



The dot product

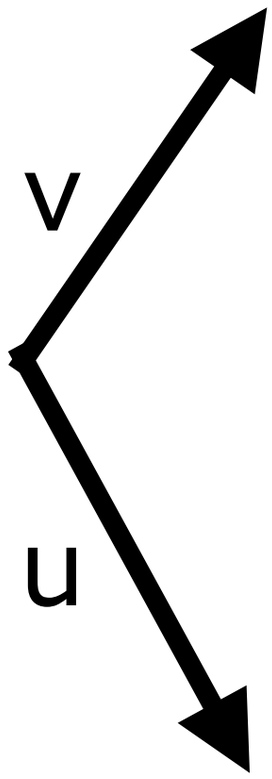
The dot product of two vectors is a the cosine of the angle between those vectors

$$\{a, b, c\} * \{d, e, f\} = ad + be + cf$$

The dot product

The dot product of two vectors is a the cosine of the angle between those vectors

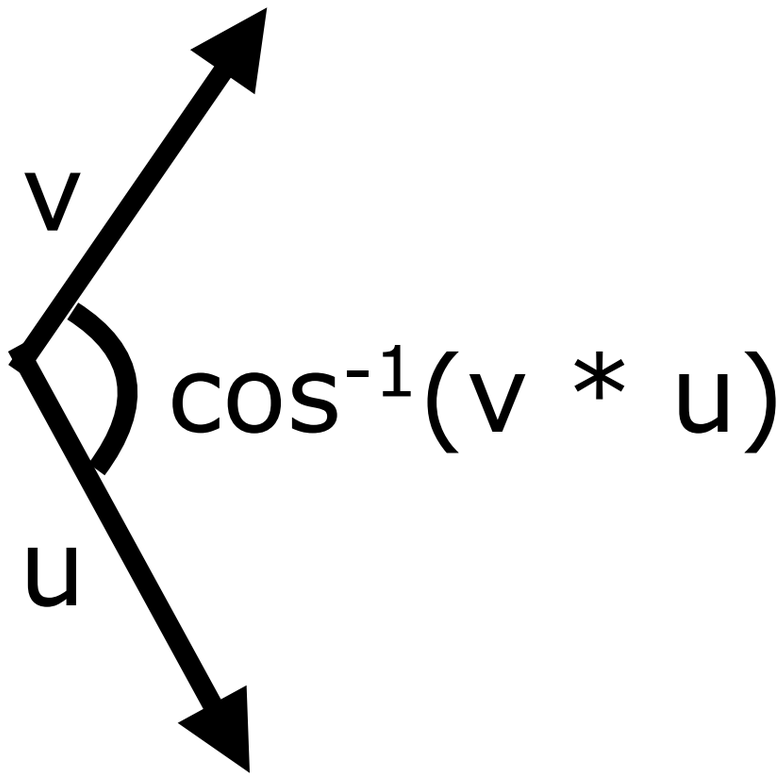
$$\{a, b, c\} * \{d, e, f\} = ad + be + cf$$



The dot product

The dot product of two vectors is a the cosine of the angle between those vectors

{a, b, c} * {d, e, f} =
ad + be + cf



Points

Points consist of a x-, a y-, and a z-coordinate in 3D space

Even though they have the same elements as vectors it is often considered good coding practice not to mix their code

Point distance

We obtain the distance vector between two points by pointwise subtracting their components

D [a, b, c] [d, e, f]

=

{d-a, e-b, f-c}

Moving points

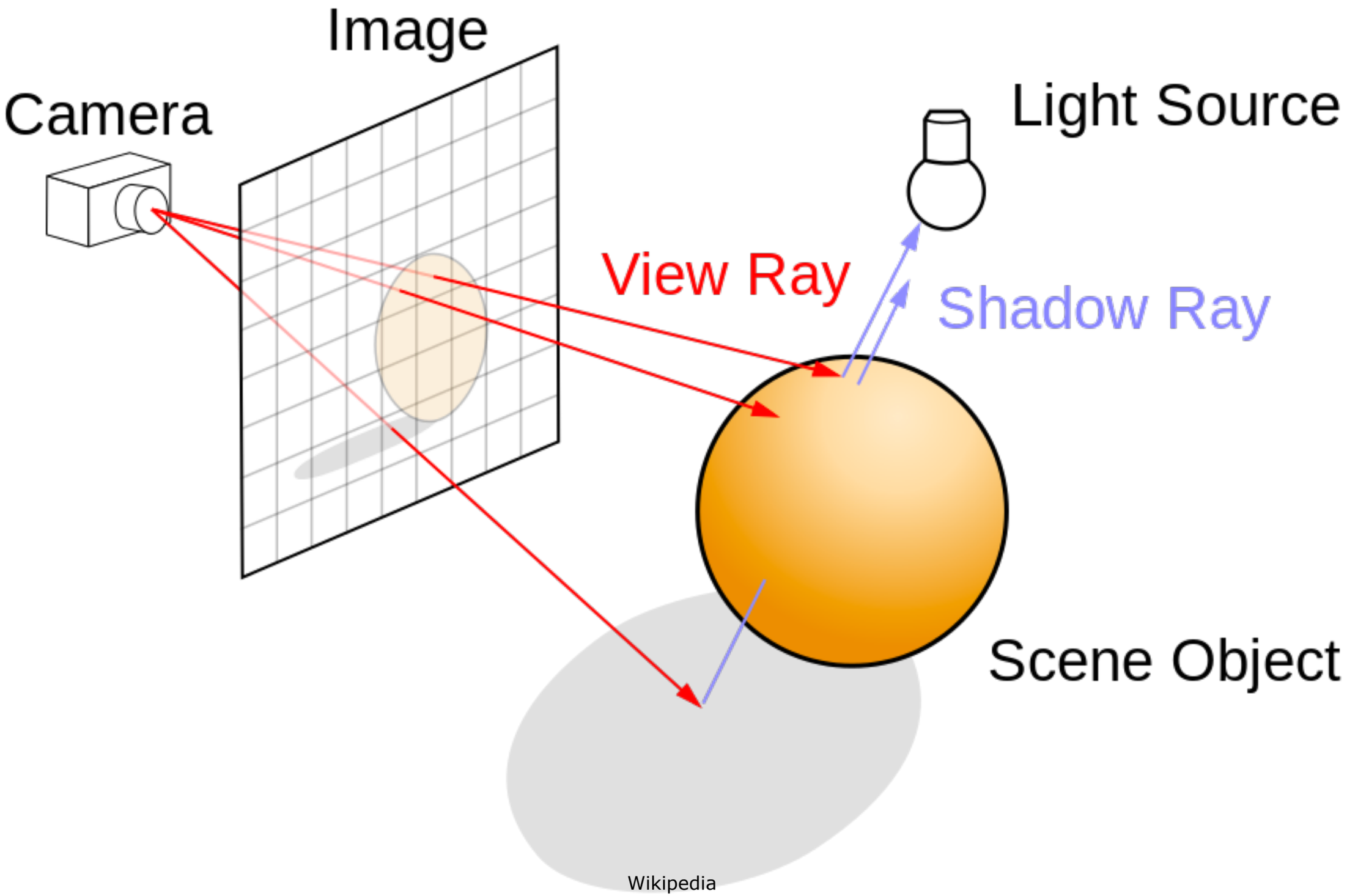
We can move points along a vector by pointwise multiplying their components

M [a, b, c] {d, e, f}

=

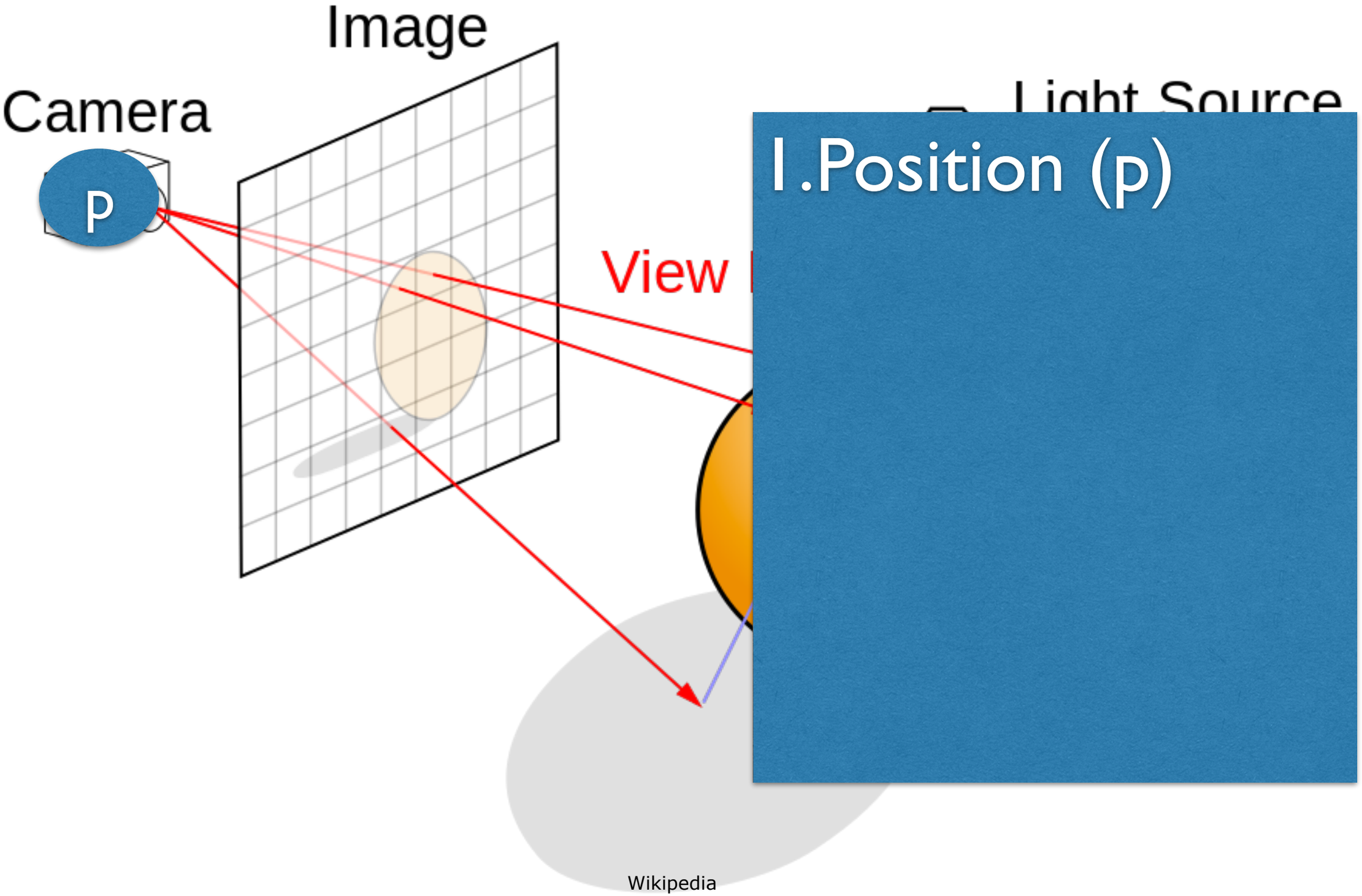
[a*d, b*e, c*f]

The camera



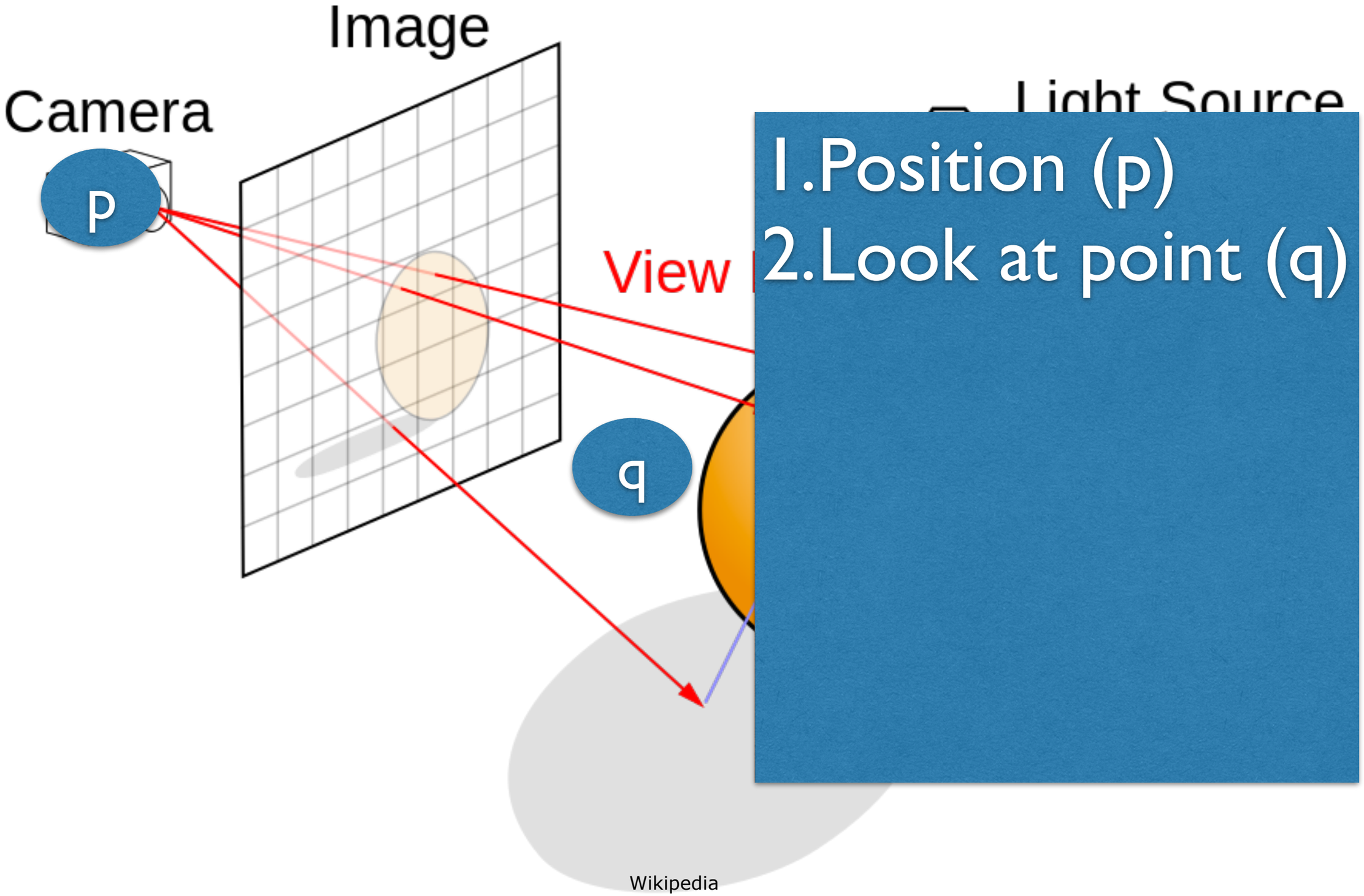
Wikipedia

The camera



Wikipedia

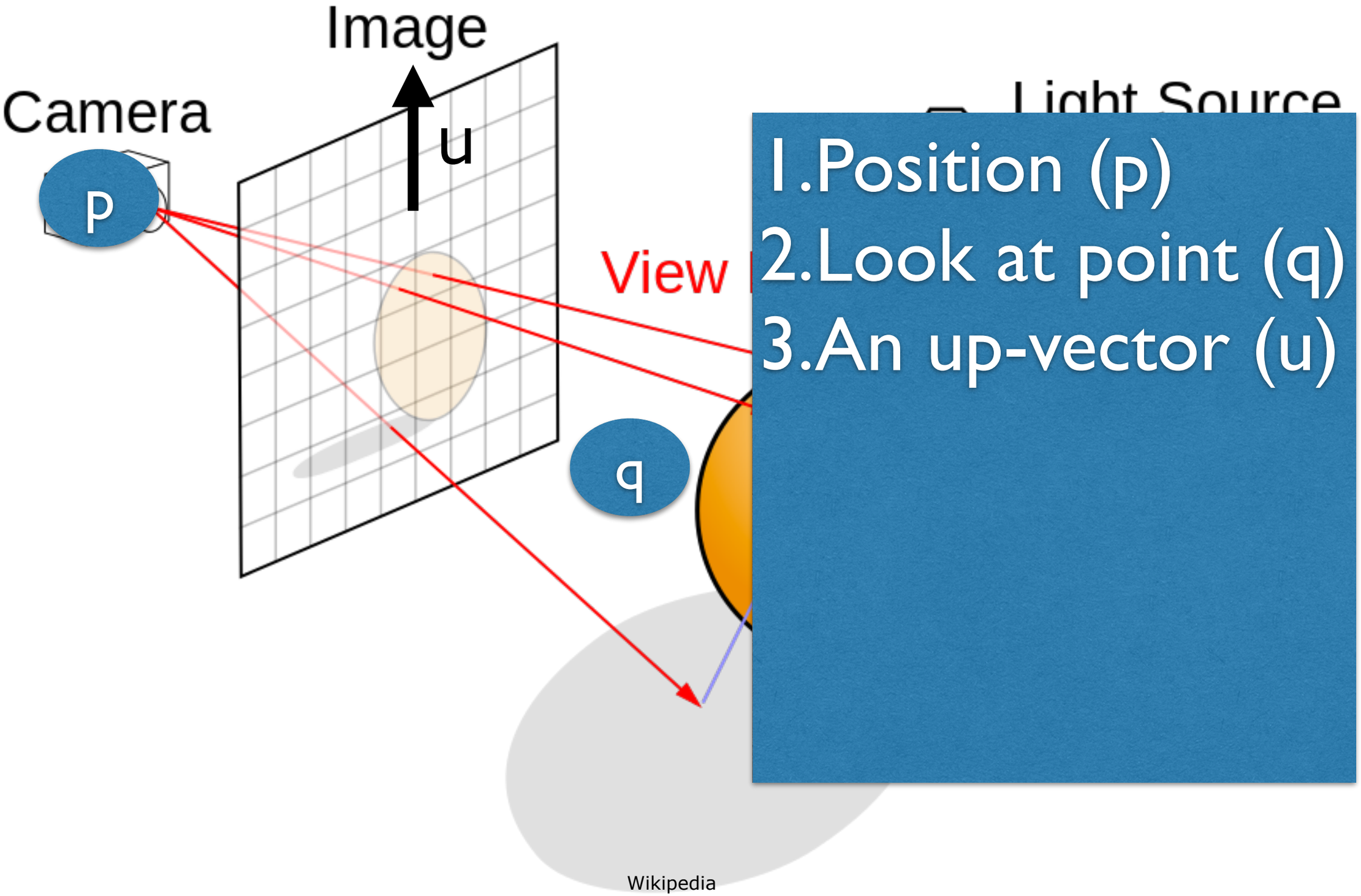
The camera



1. Position (p)
2. Look at point (q)

Wikipedia

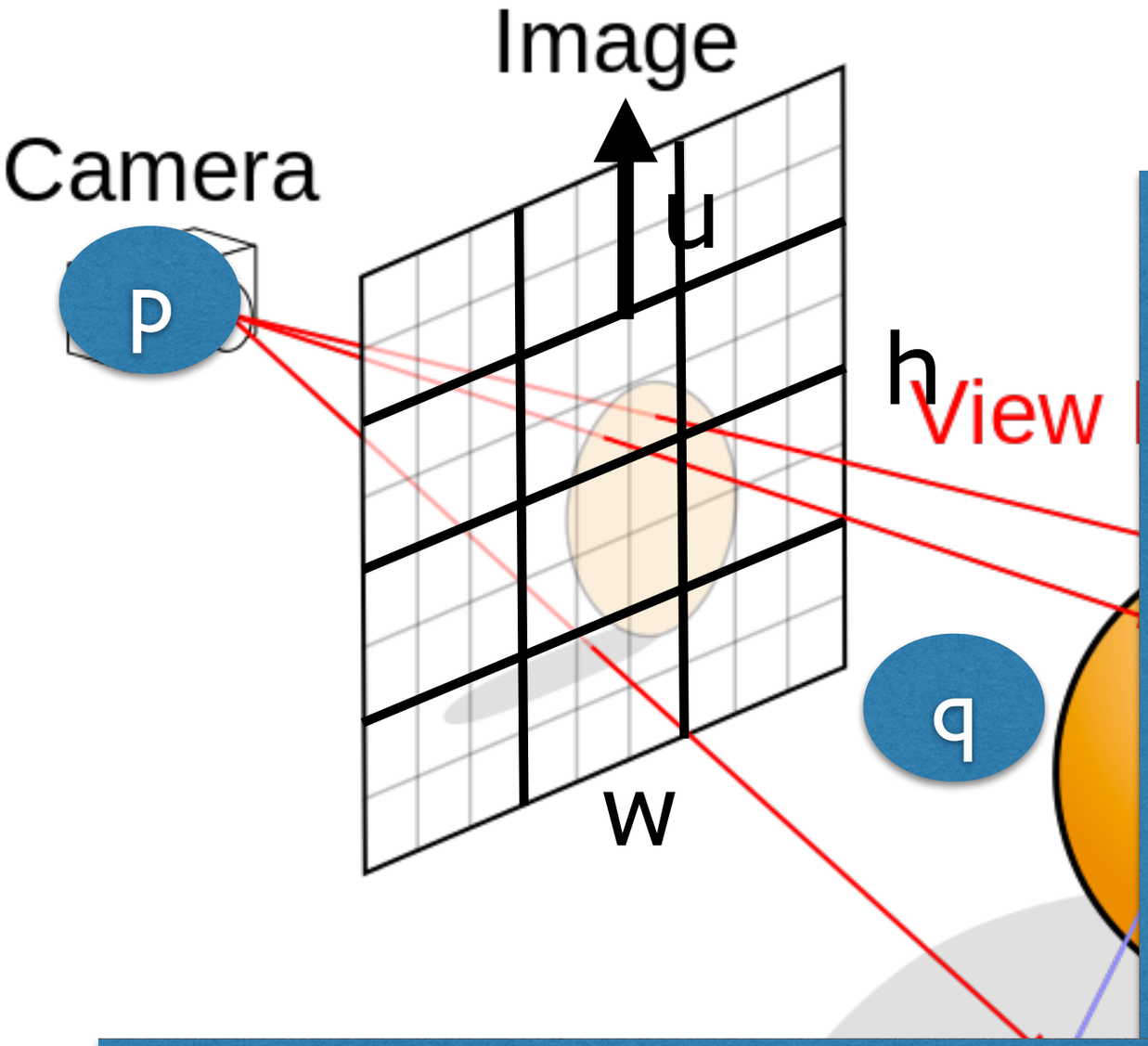
The camera



1. Position (p)
2. Look at point (q)
3. An up-vector (u)

Wikipedia

The camera

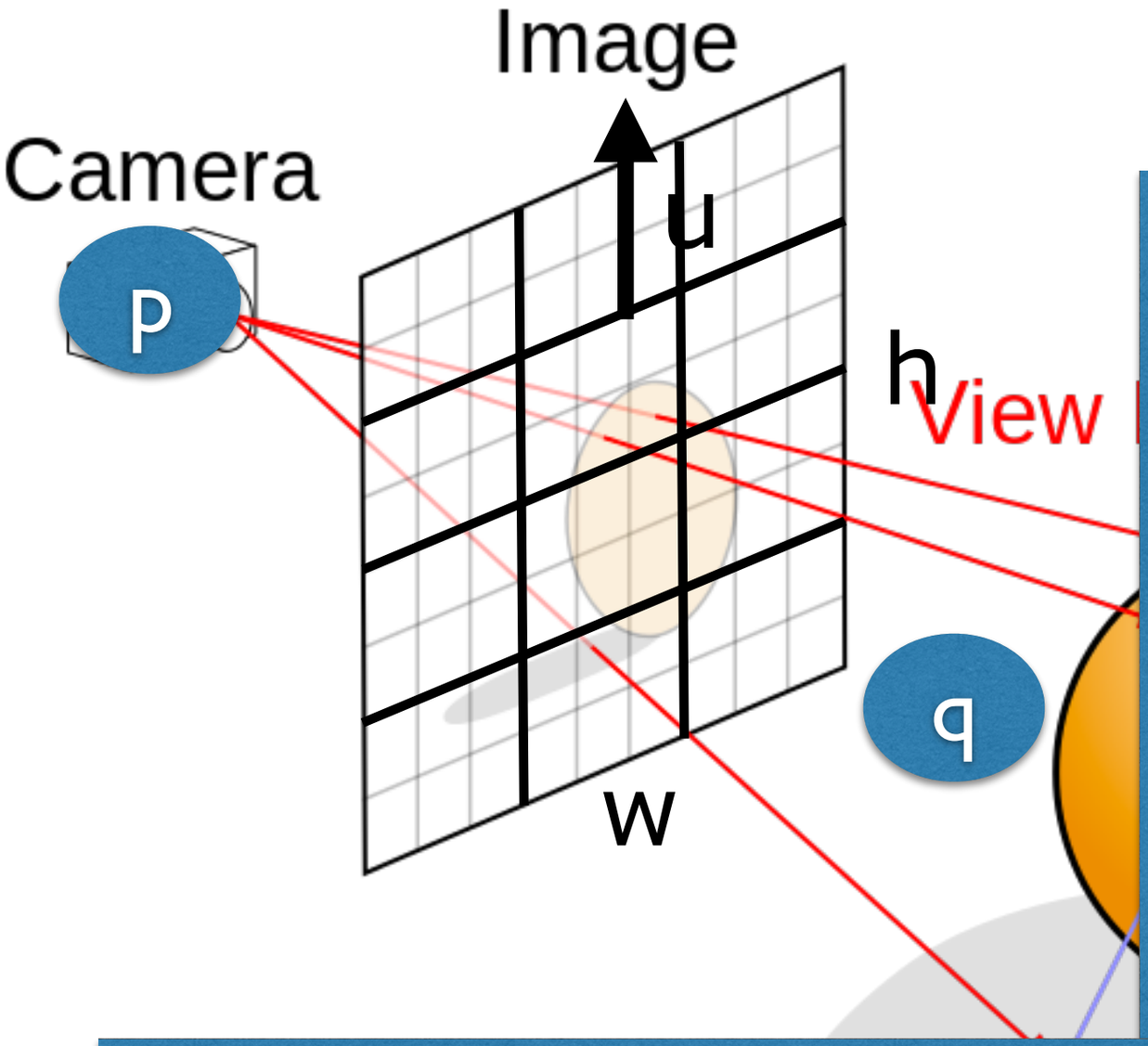


1. Position (p)
2. Look at point (q)
3. An up-vector (u)
4. Unit resolution

Picture resolution in units is 3 * 4 (h * w)

Wikipedia

The camera

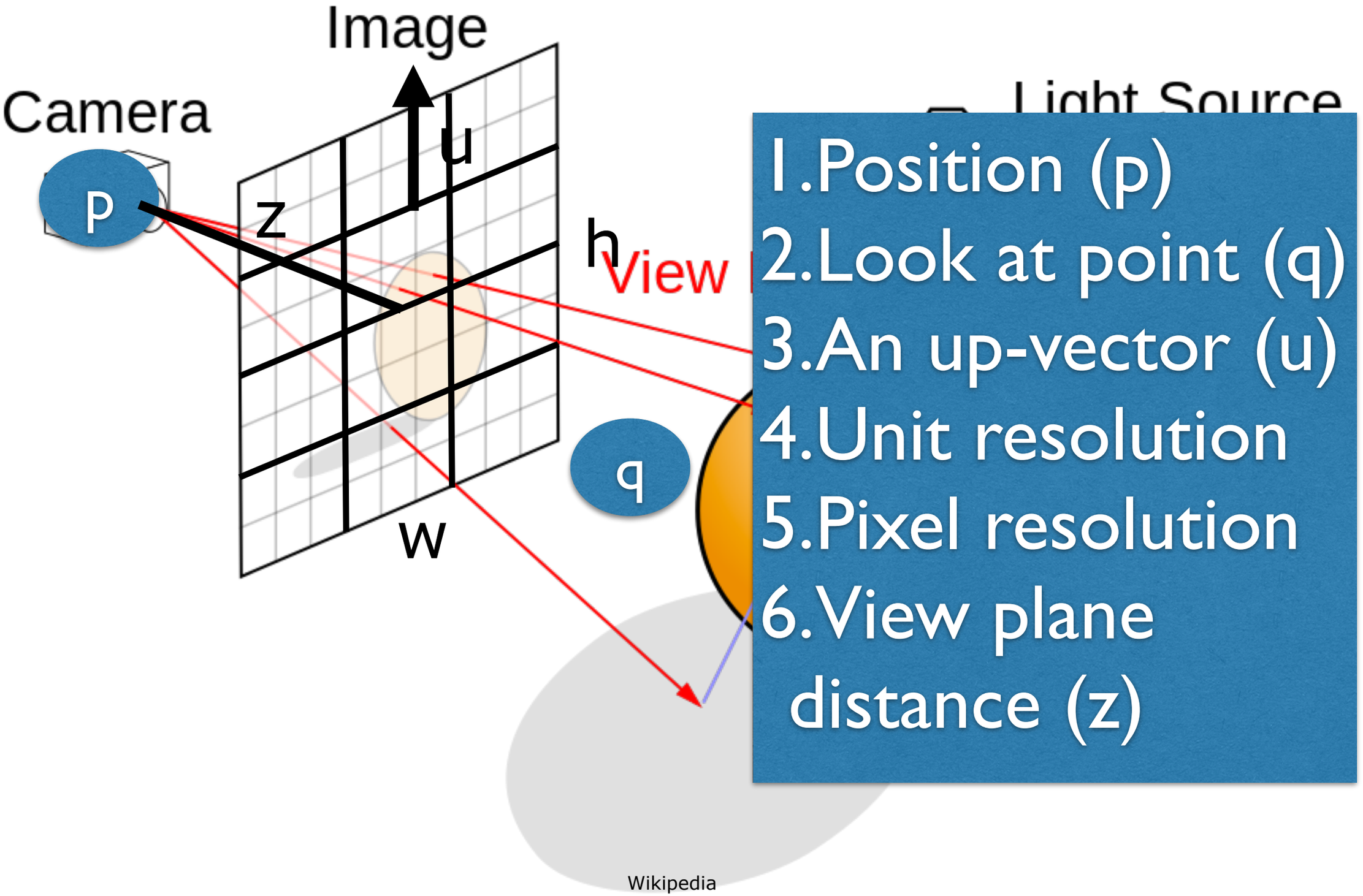


1. Position (p)
2. Look at point (q)
3. An up-vector (u)
4. Unit resolution
5. Pixel resolution

Picture resolution in pixels is 9 * 8 (x * y)

Wikipedia

The camera



1. Position (p)
2. Look at point (q)
3. An up-vector (u)
4. Unit resolution
5. Pixel resolution
6. View plane distance (z)

Wikipedia

Rays

A ray consists of a point of origin and a normalised direction vector

- We fire rays out of the camera
- We bounce rays off of objects
- We trace rays towards light sources

Many more possibilities

Rays

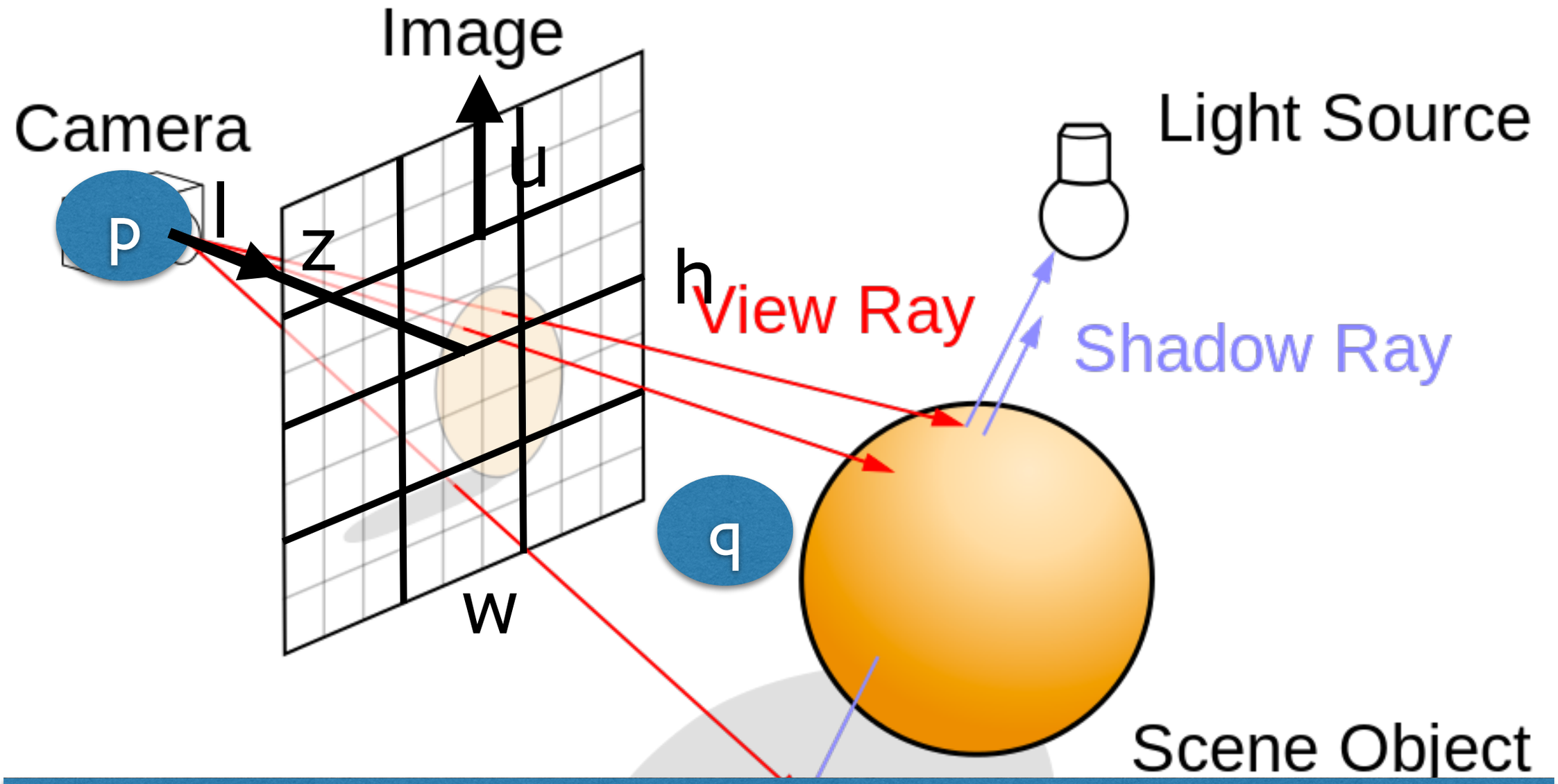
A ray consists of a point of origin and a normalised direction vector

Key idea!

Fire a ray from the camera through the centre of every pixel into the scene

Many more possibilities

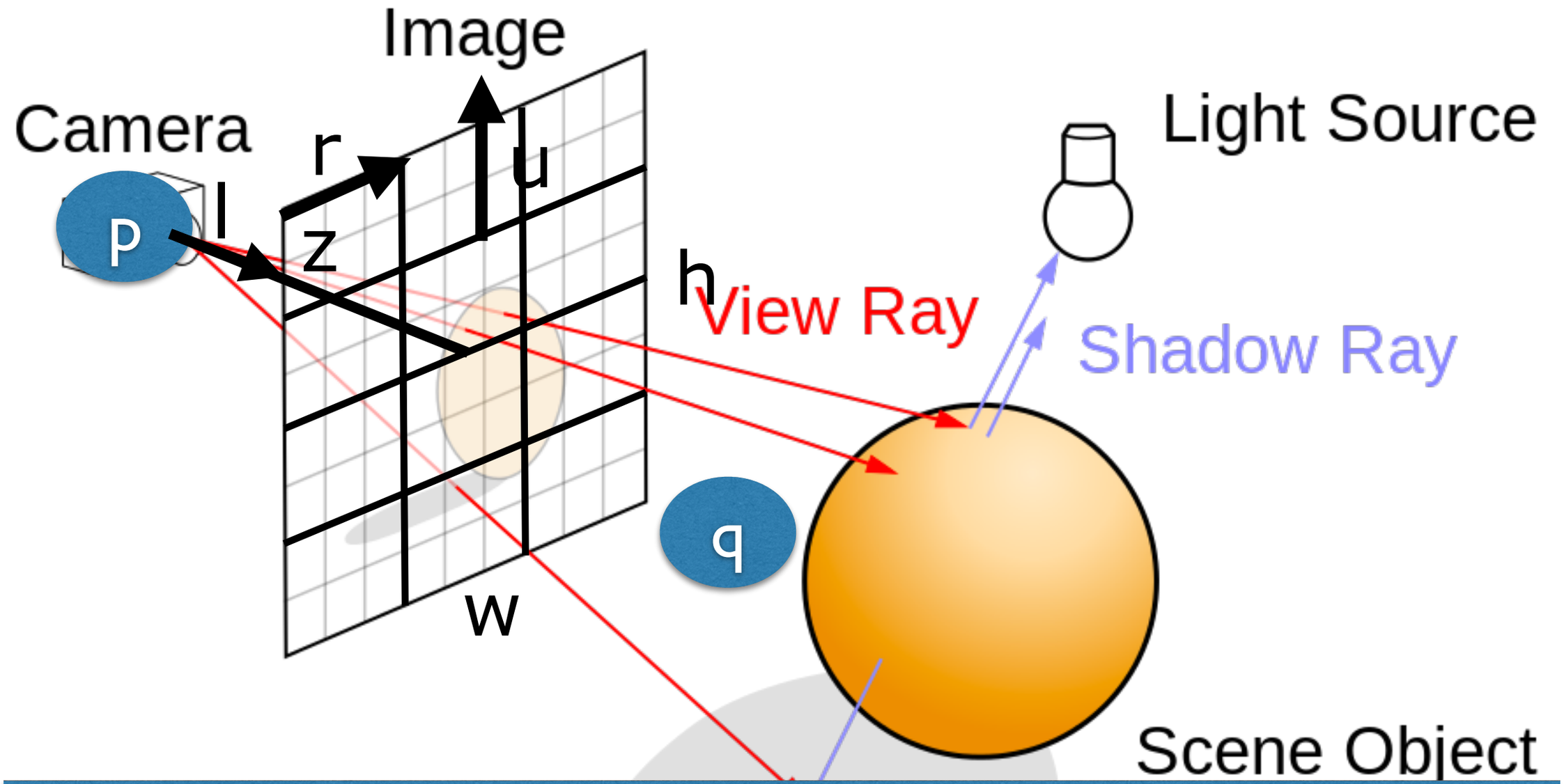
The camera



Calculate the normalised direction vector l between p and q (l = (D p q)^)

IT UNIVERSITY OF COPENHAGEN

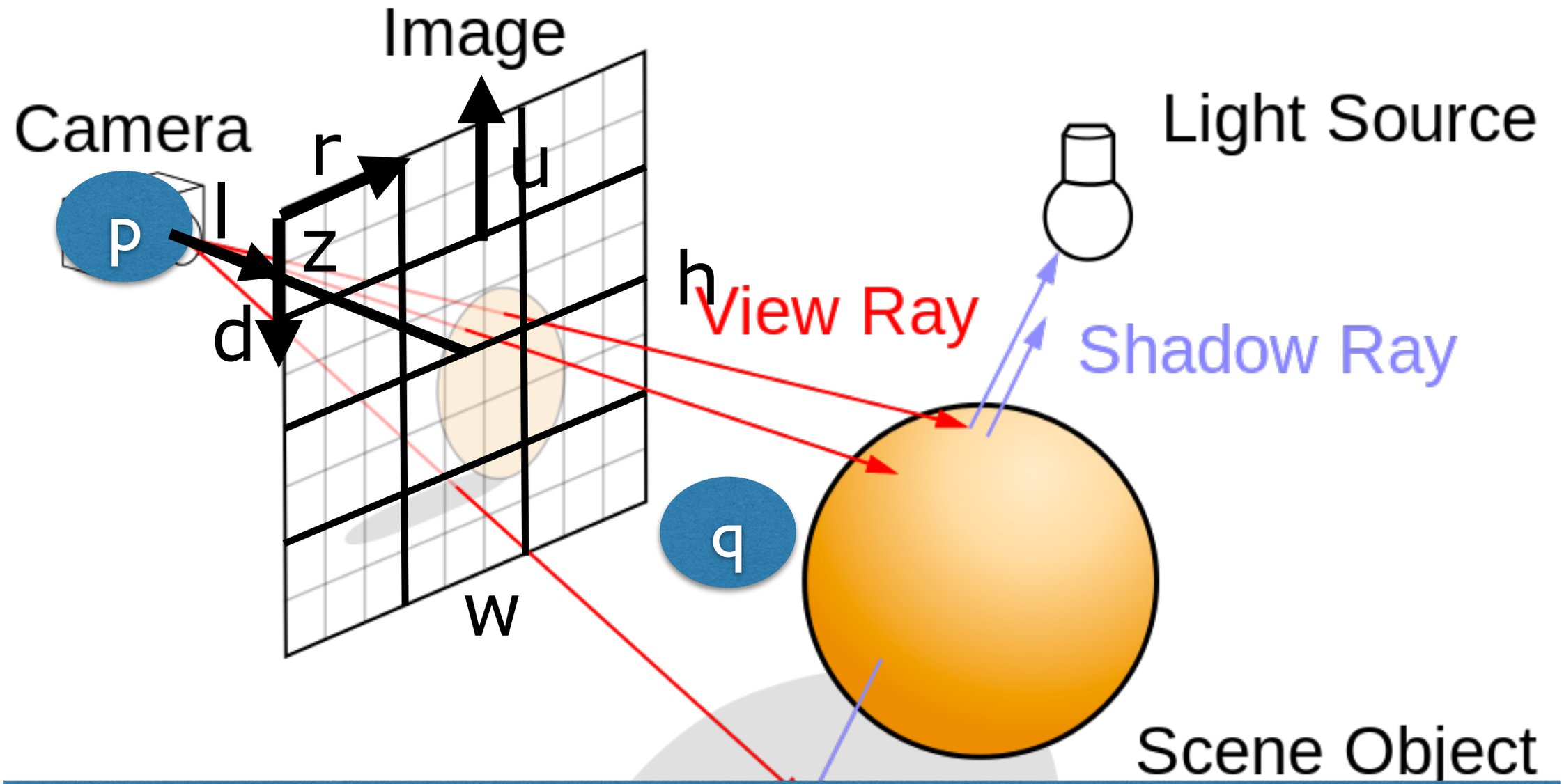
The camera



Calculate the right direction vector r of the image plane (r = (u x l)^)

IT UNIVERSITY OF COPENHAGEN

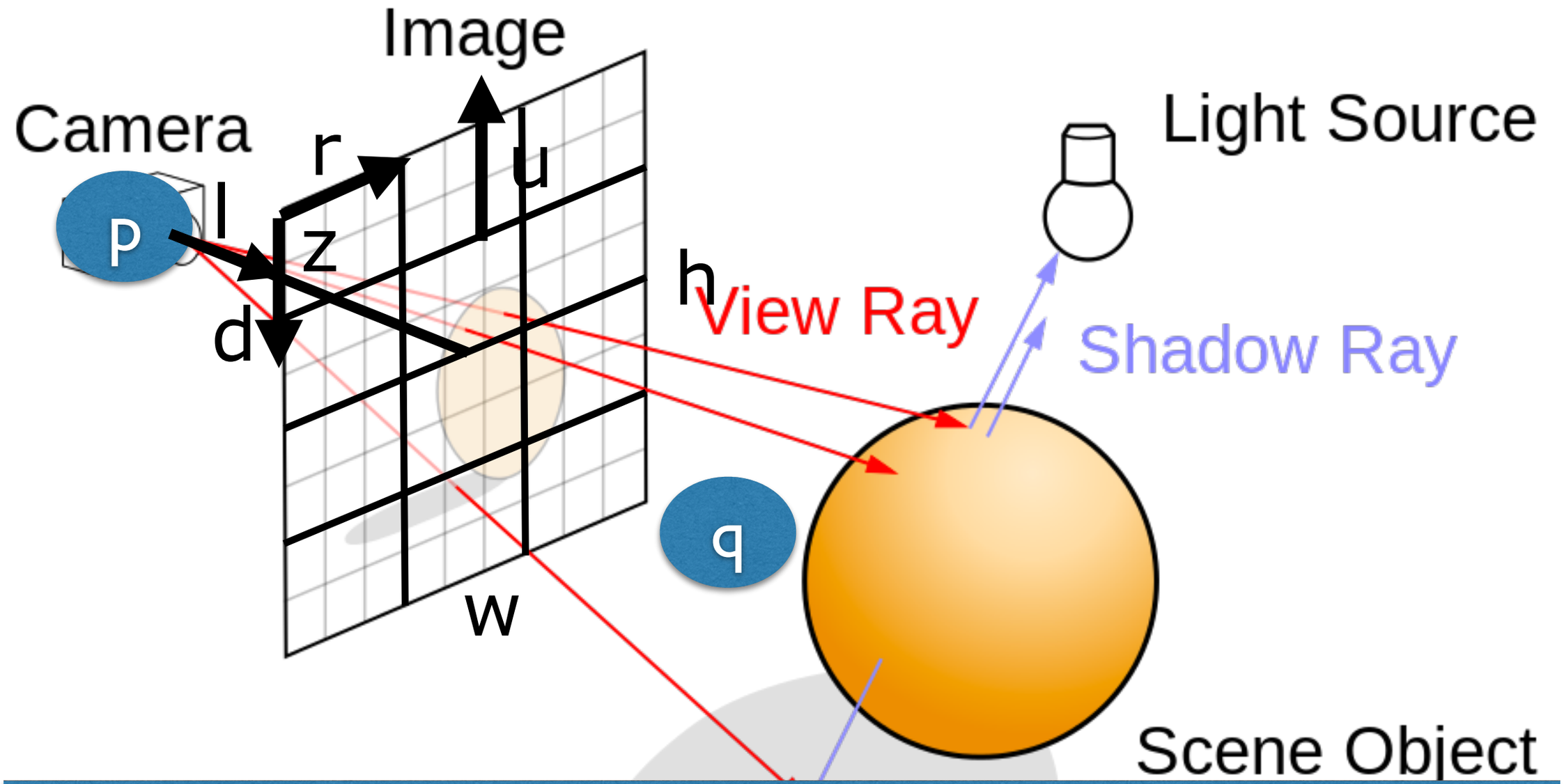
The camera



Calculate the down direction vector d of the image plane (d = (r x l)^) or u * -1

IT UNIVERSITY OF COPENHAGEN

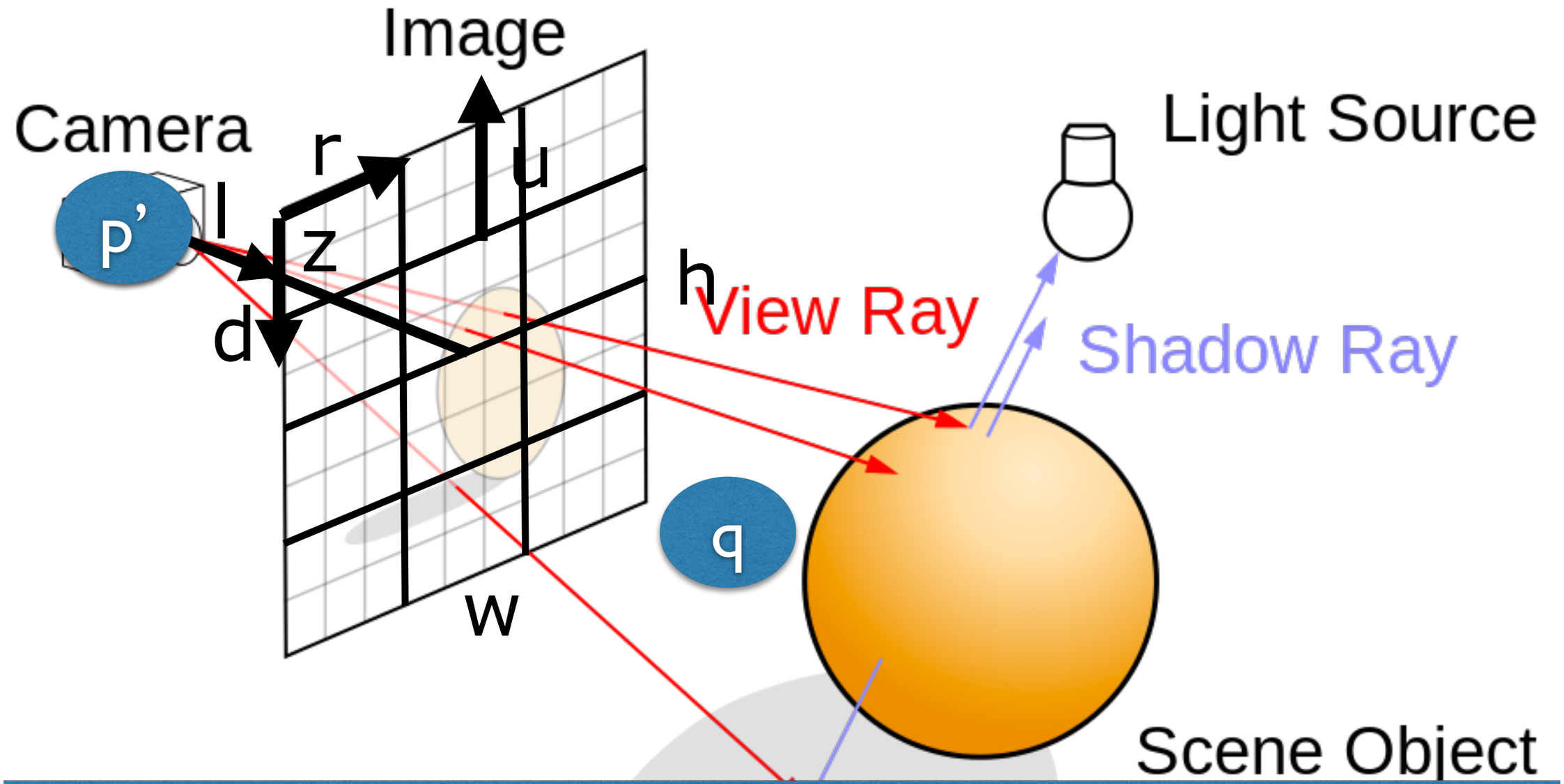
The camera



Calculate the down direction vector d of the image plane (d = (r x l)^) or u * -1

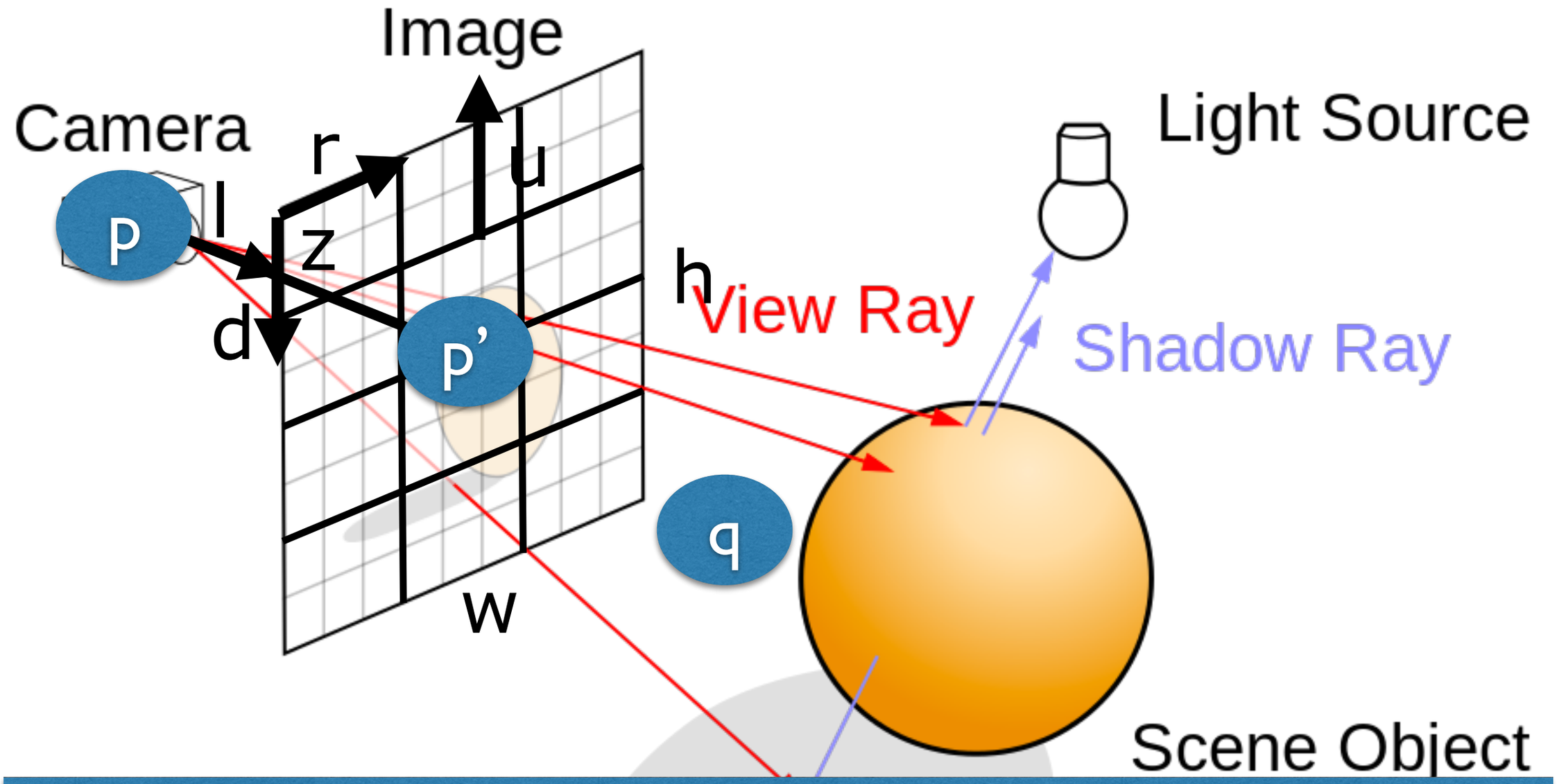
IT UNIVERSITY OF COPENHAGEN

The camera



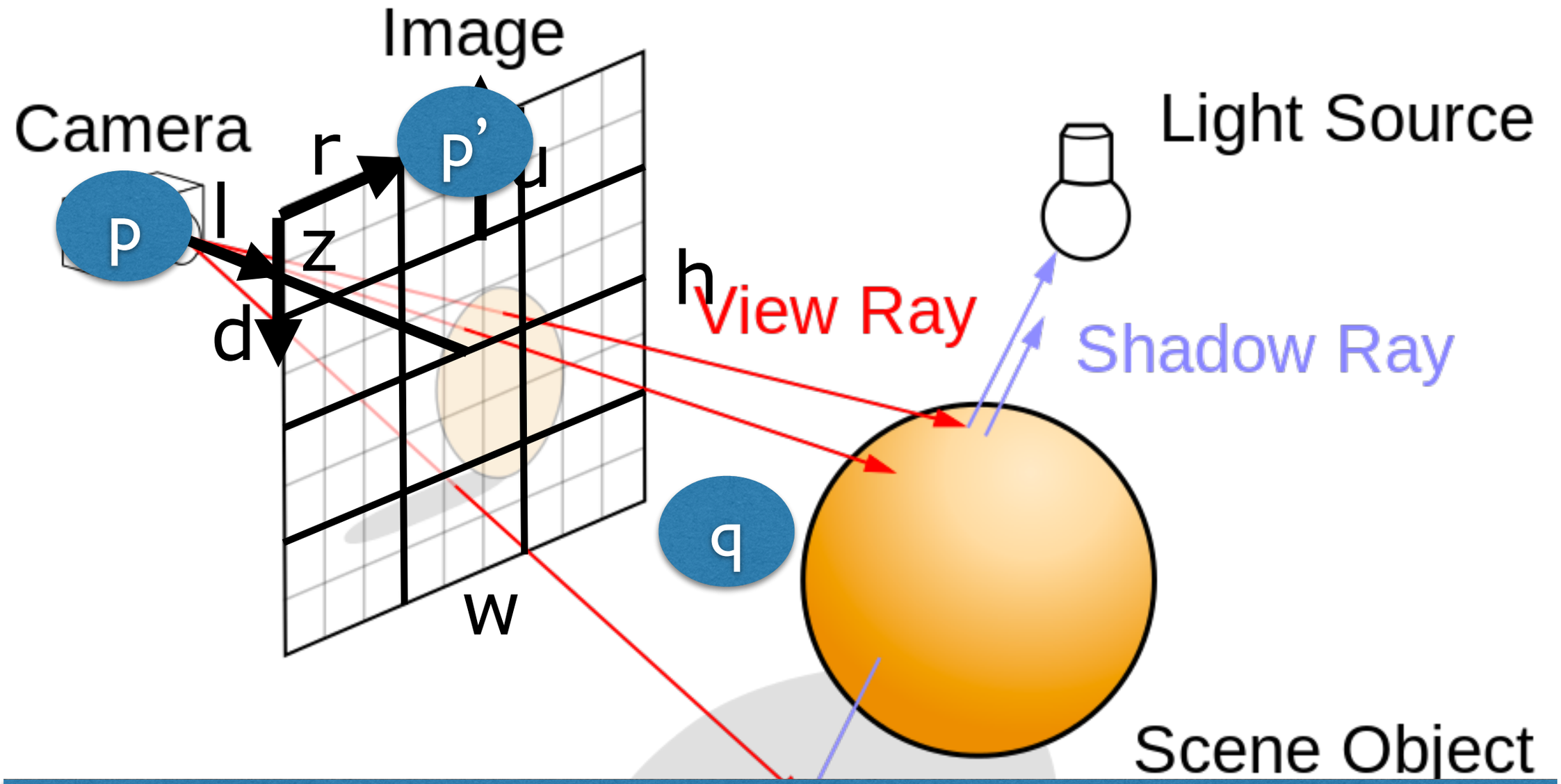
Calculate the down direction vector d of the image plane (d = (r x l)^) or u * -l

The camera



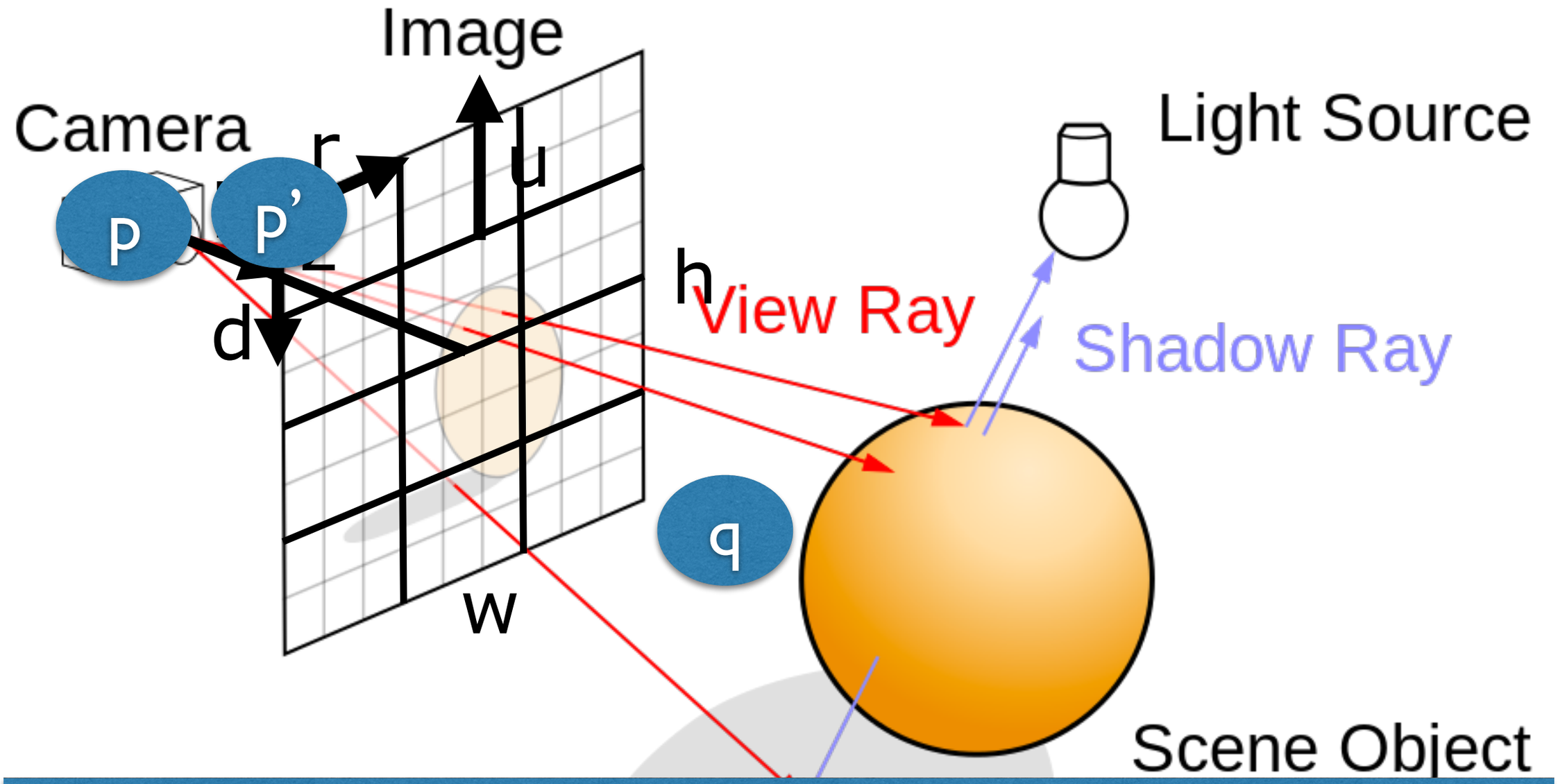
Move p to the centre (p' = M p (l * z))

The camera



Move p' to the top (p' = M p' (u * (h/2)))

The camera



Move p' to the left (p' = M p' (r * (-w/2)))

The camera

Image

Came ource

P ay

bject

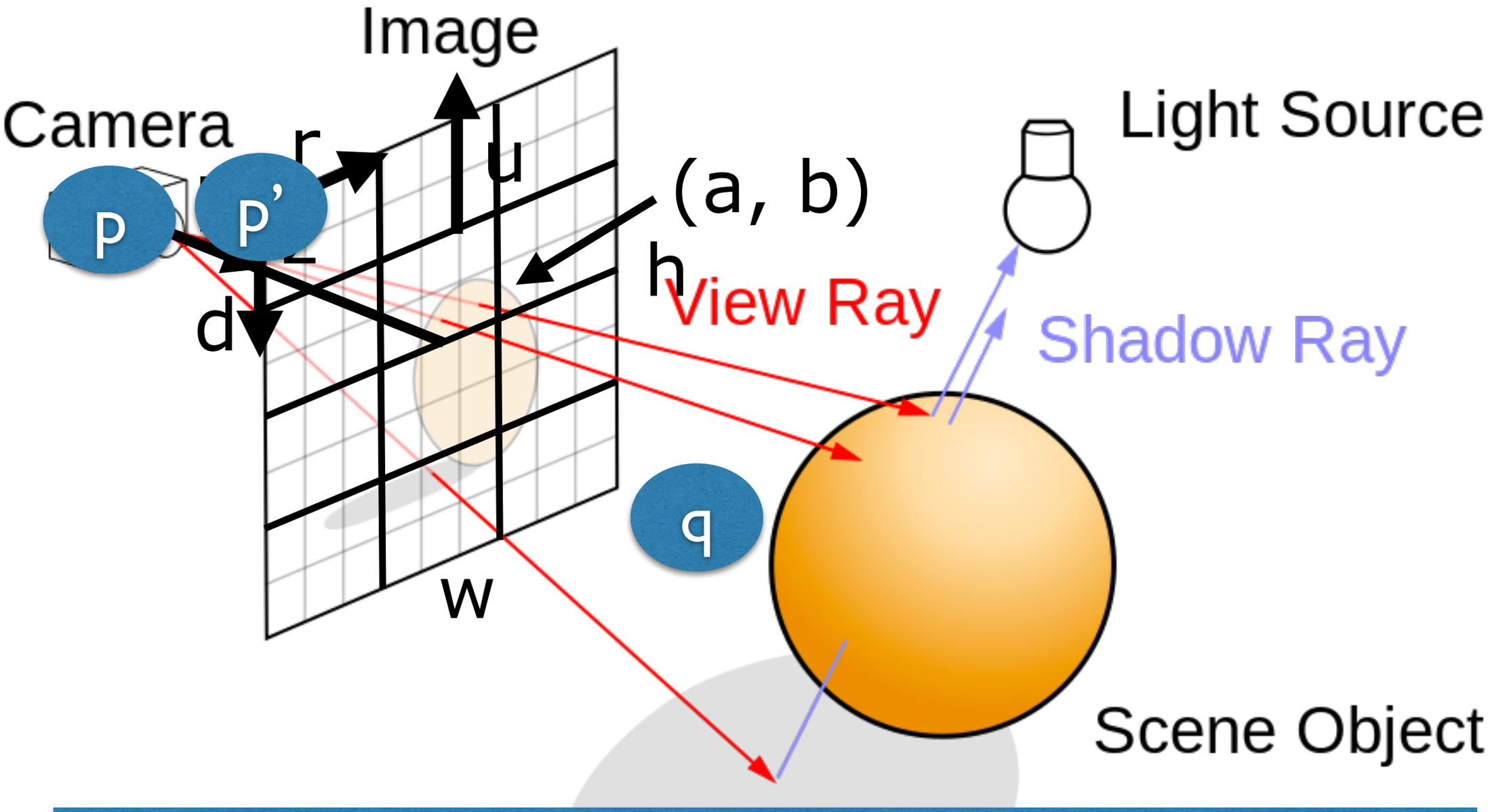
The size of each pixel in units is
W = w / x times H = h / y

In our case that is
3 / 9 times 4 / 8

W = 0.33
H = 0.5

Wikipedia

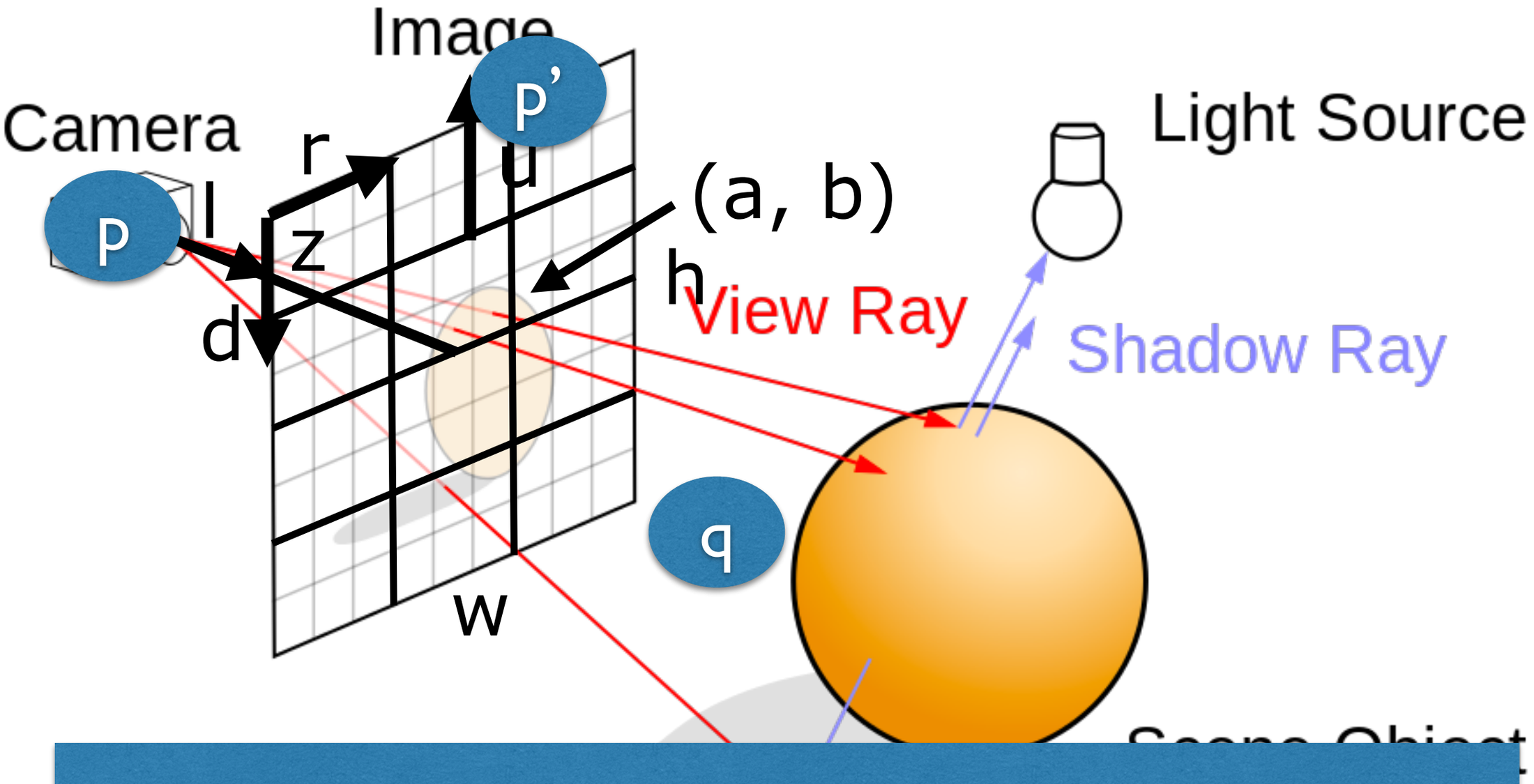
Target pixel (a, b)



The origin of each ray is p

Wikipedia

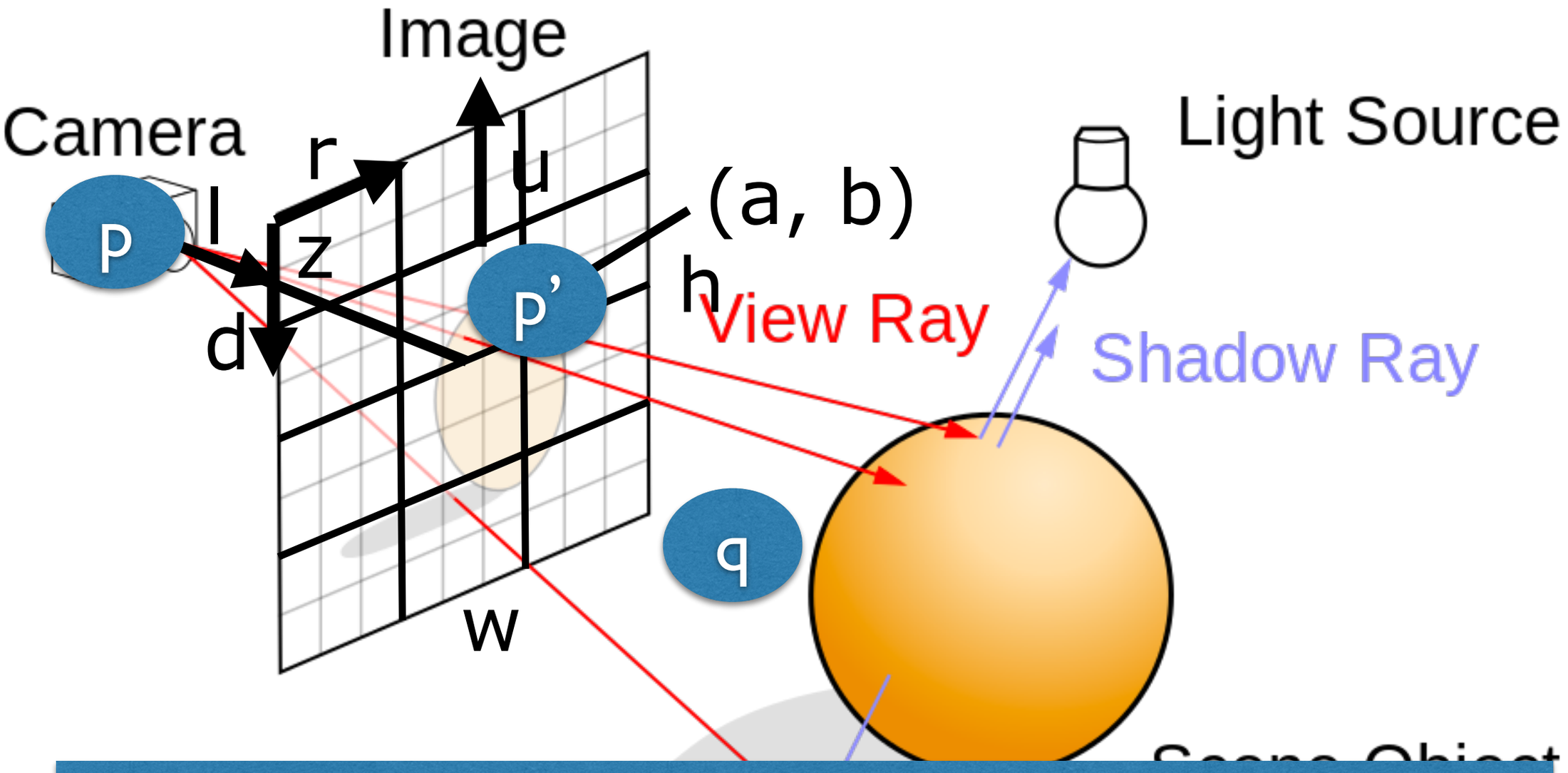
Target pixel (a, b)



Move p' to the pixel column

$p' = M\, p' (r * ((a + 0.5) * W))$

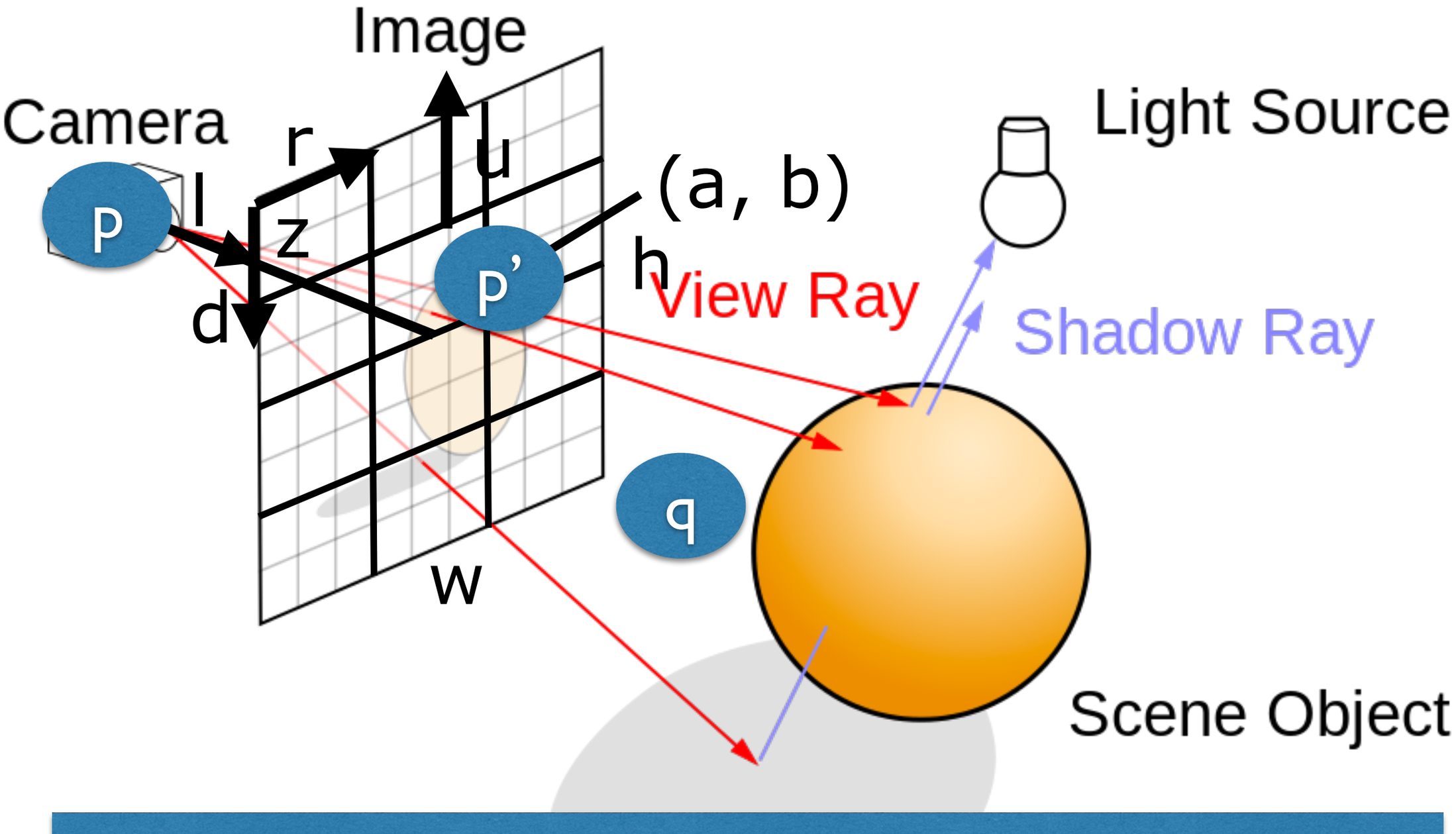
Target pixel (a, b)



Move p' to the pixel row

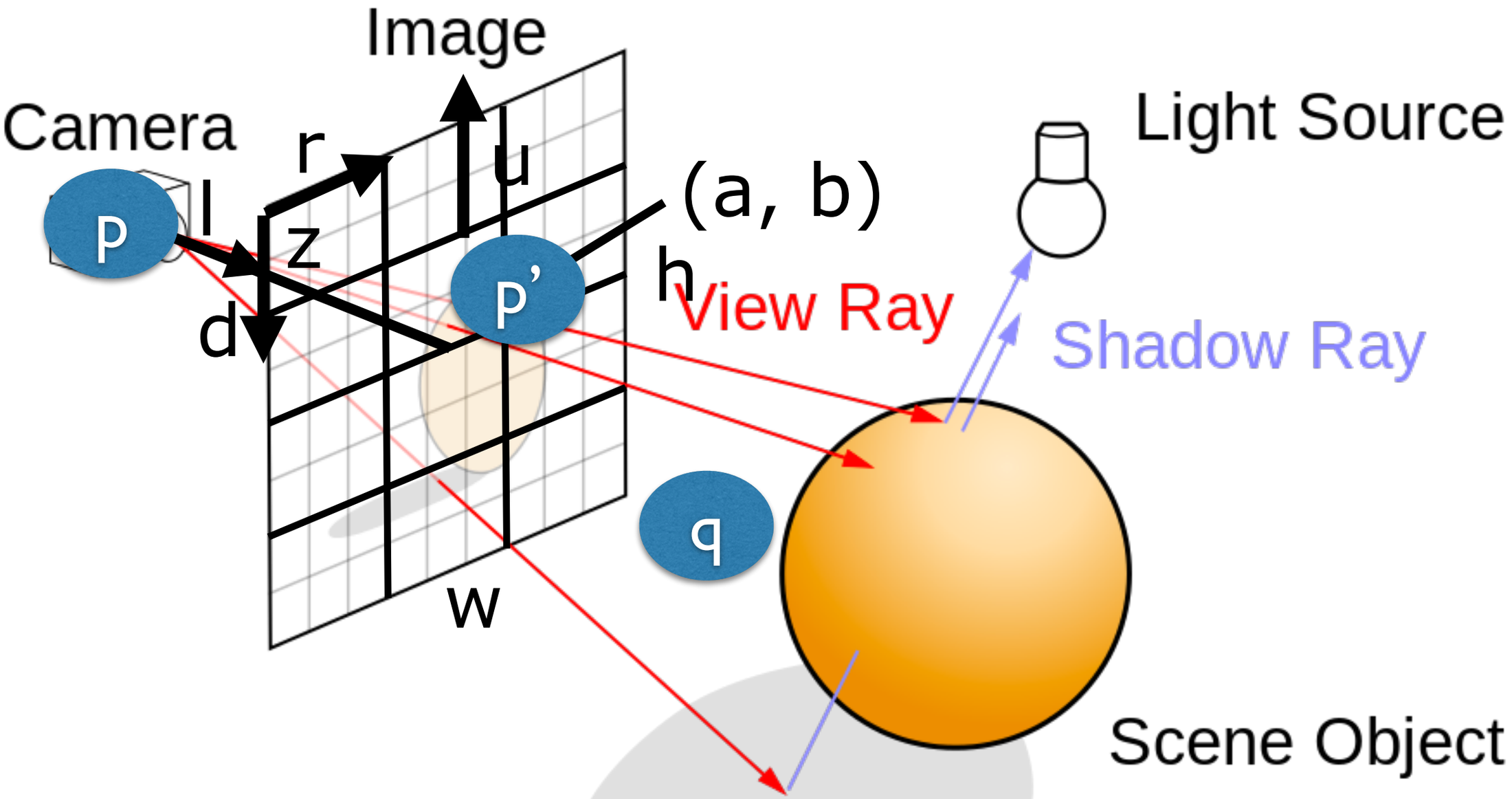
$$p' = M\, p' (d * ((b + 0.5) * H))$$

Target pixel (a, b)



The ray direction is (D p p')^

Target pixel (a, b)

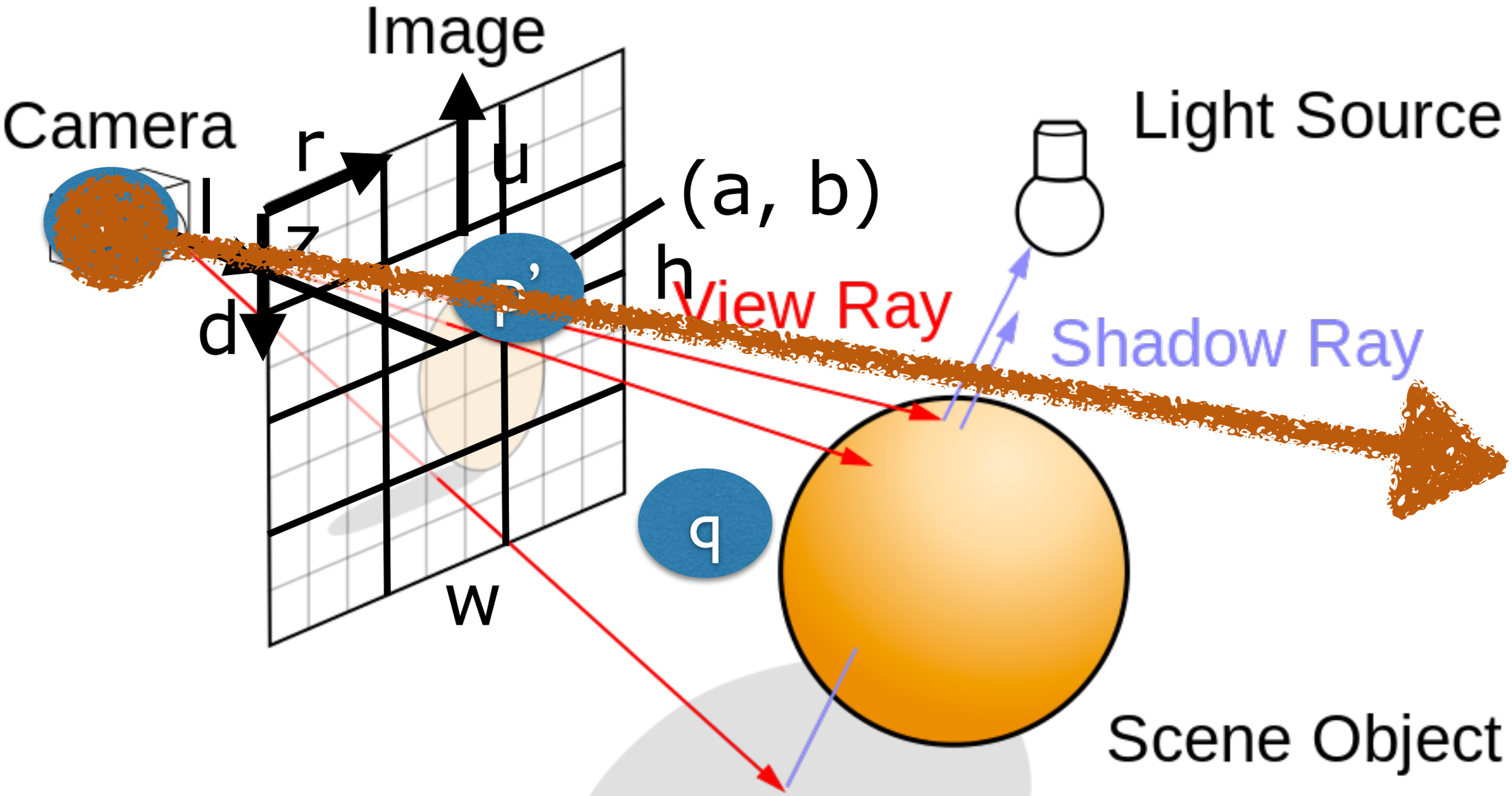


FIRE THE CANNON!!!!

Wikipedia

IT UNIVERSITY OF COPENHAGEN

Target pixel (a, b)



FIRE THE CANNON!!!!

Wikipedia

Did we hit anything?

The hit function

Every shape needs a hit function. The hit function takes a ray as an argument and returns (if it hits)

- The distance to the hit point
- The normal of the hit point (A vector perpendicular to the surface at the hit point)
- The material at the hit point (for our purposes colour and reflective property)

The hit function

Every shape needs a hit function. The hit function takes a ray as an argument and

-
-
-

or

hit

ur

property)

Key observation!!!

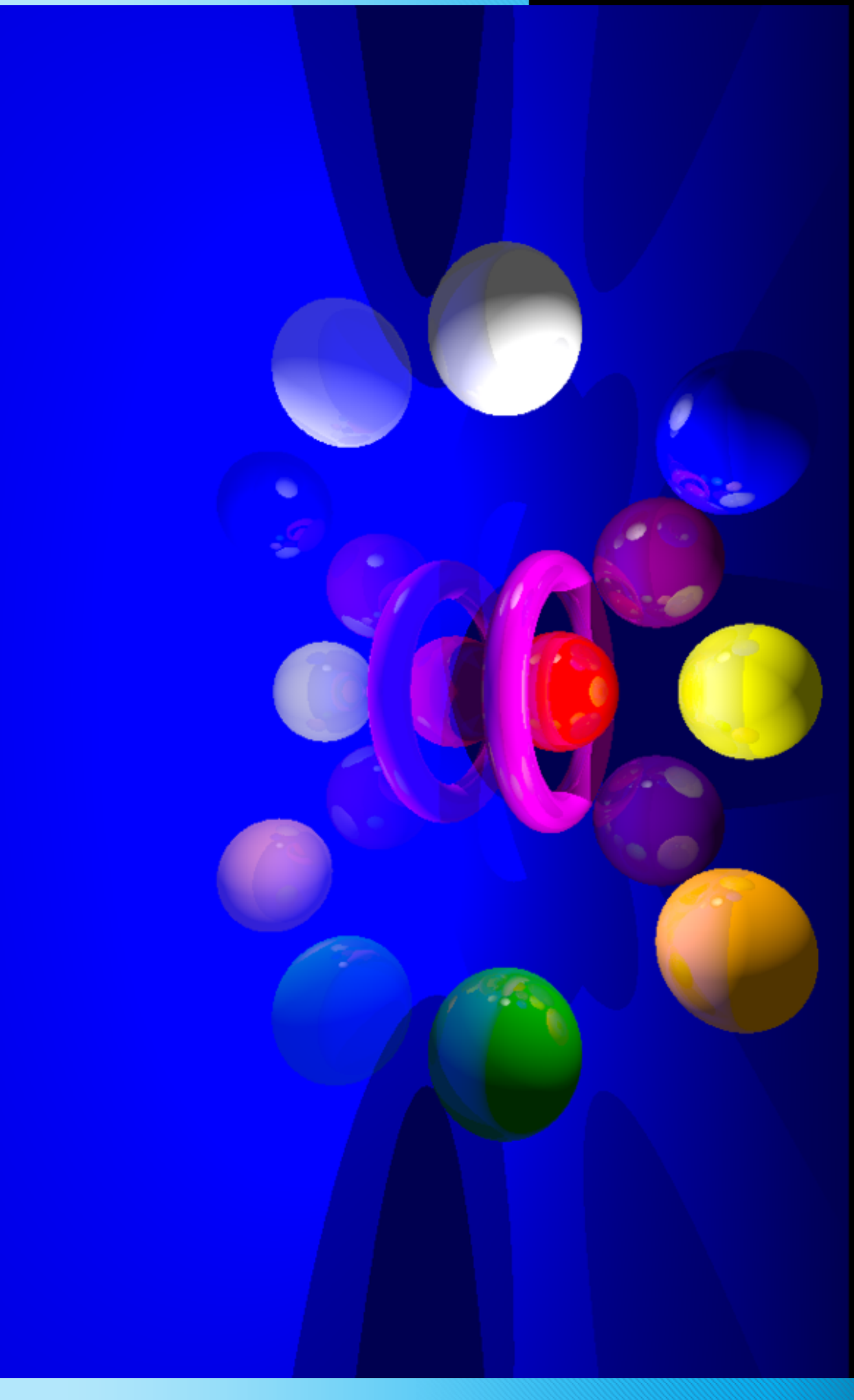
The rendering function does not know what shapes there are. All it assumes is that it fires rays into a scene populated by well-behaved shapes

IT UNIVERSITY OF COPENHAGEN

Questions?

Ray Tracing

Painting with colours

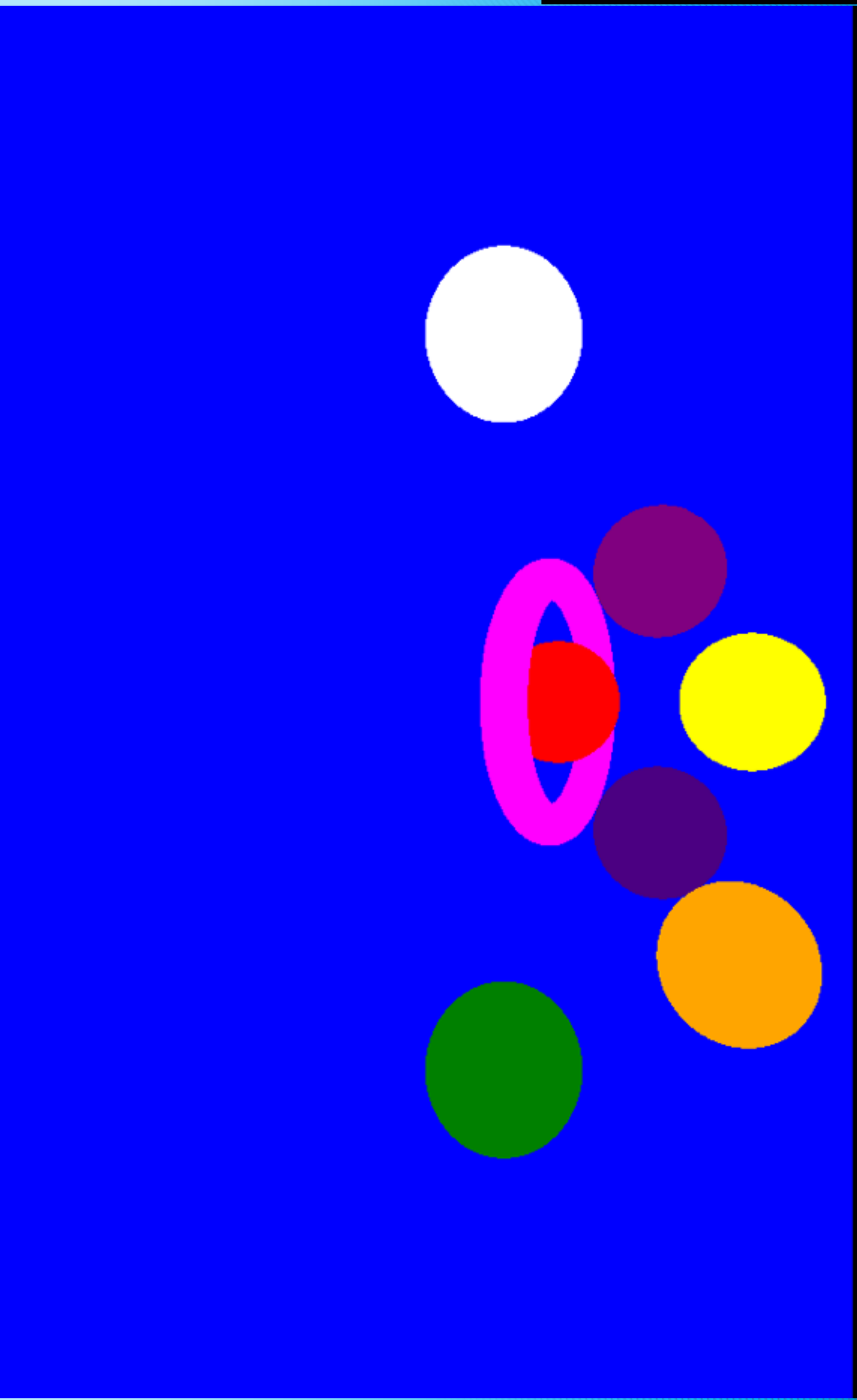


Colouring objects

The hit function tells us what type of material we hit

- What colour does it have
- How reflective is it?

If we just give our object their source colour we get something very flat



Colours in a computer

A colour is represented by three floating point numbers between 0 and 1 for red, green, and blue respectively

All colour work you do should be on these floating point numbers, the last thing you do before saving your result is to translate these numbers to integers between 0 and 255

Colour scaling

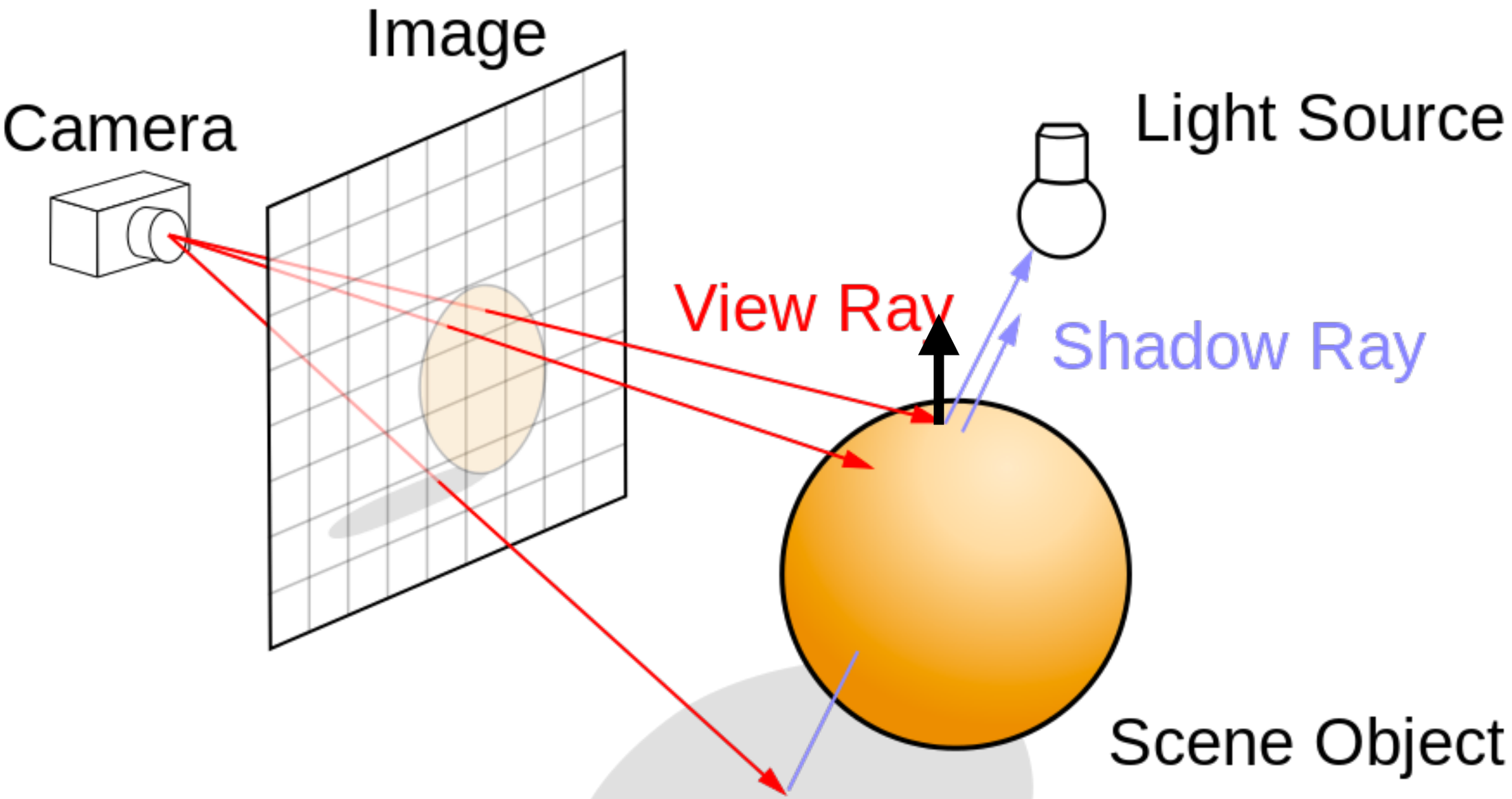
As with vectors, we need to scale our colours

$$(r, g, b) * s = (r * s, g * s, b * s)$$

IT UNIVERSITY OF COPENHAGEN

Basic colouring

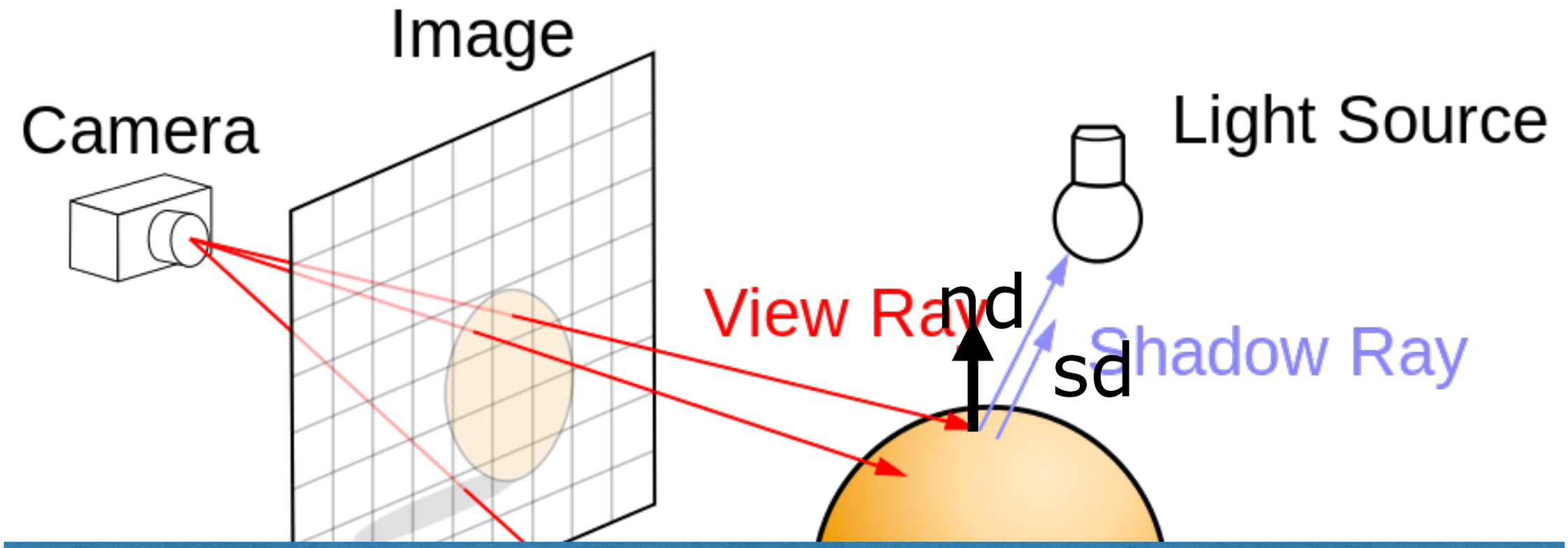
The colour of a hit point depends largely on the angle to the light that hits it



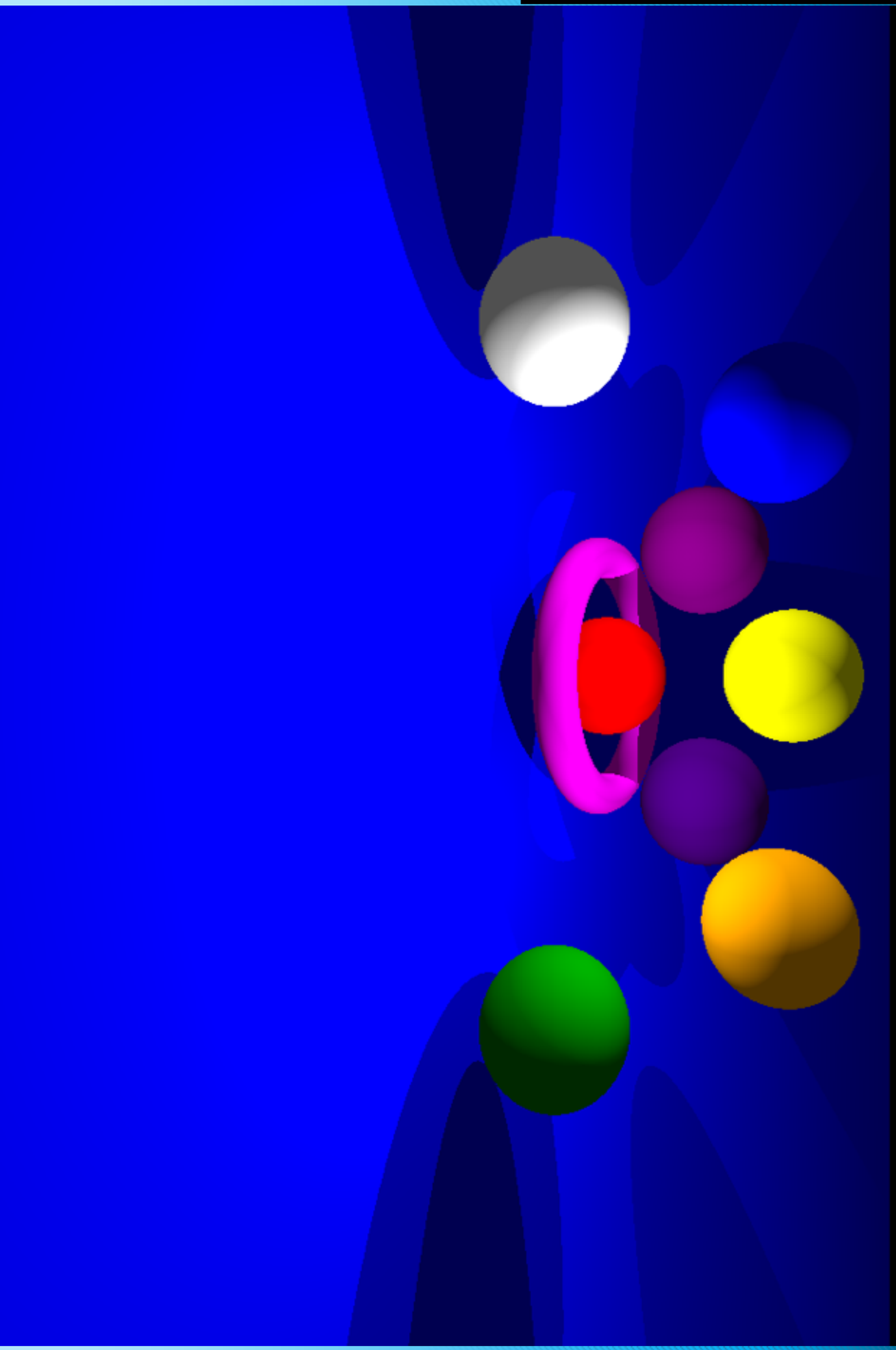
IT UNIVERSITY OF COPENHAGEN

Basic colouring

The colour of a hit point depends largely on the angle to the light that hits it

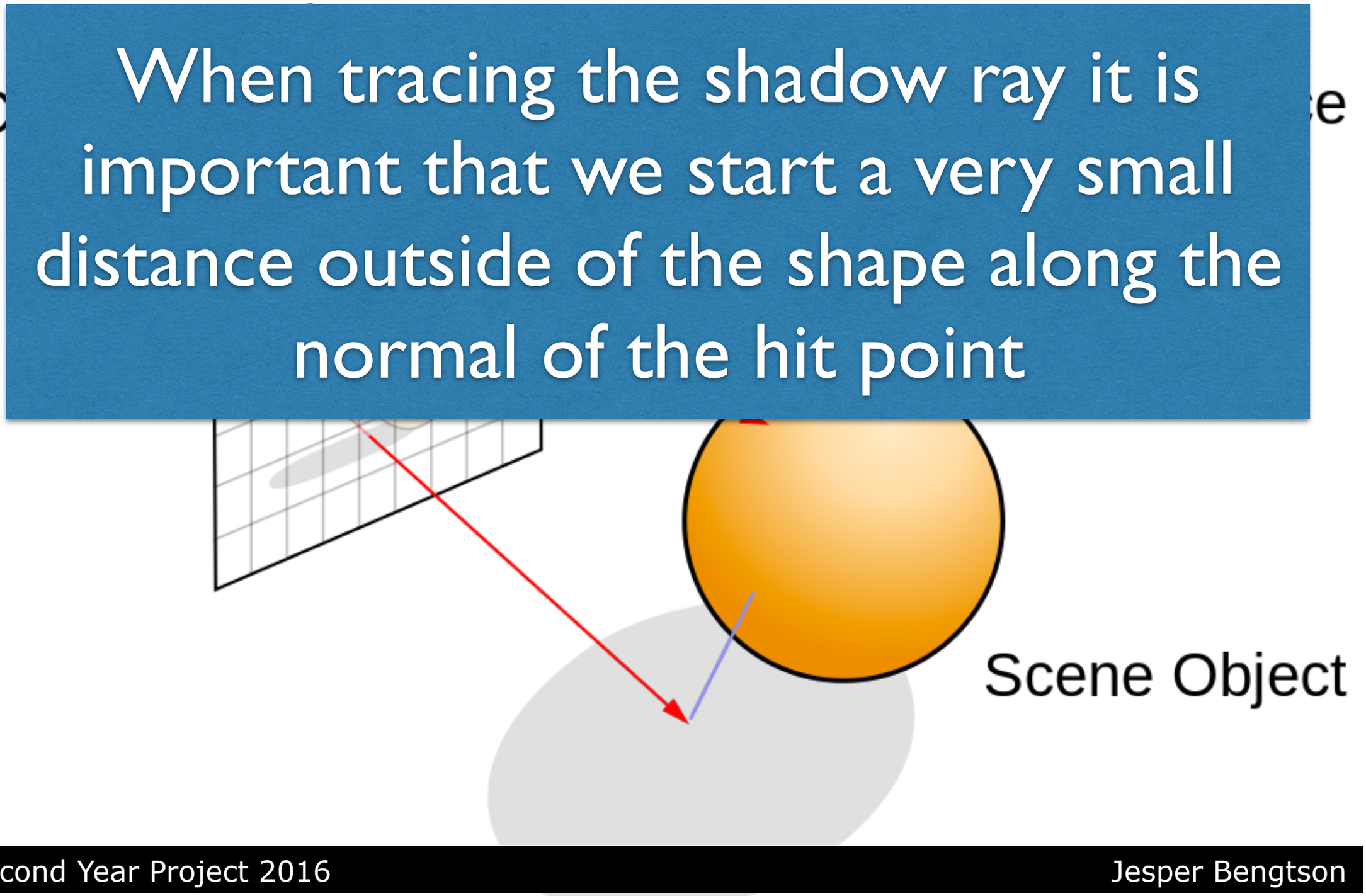


We scale the colour with the dot product of the ray normal direction and shadow ray direction
(c * (nd * sd))



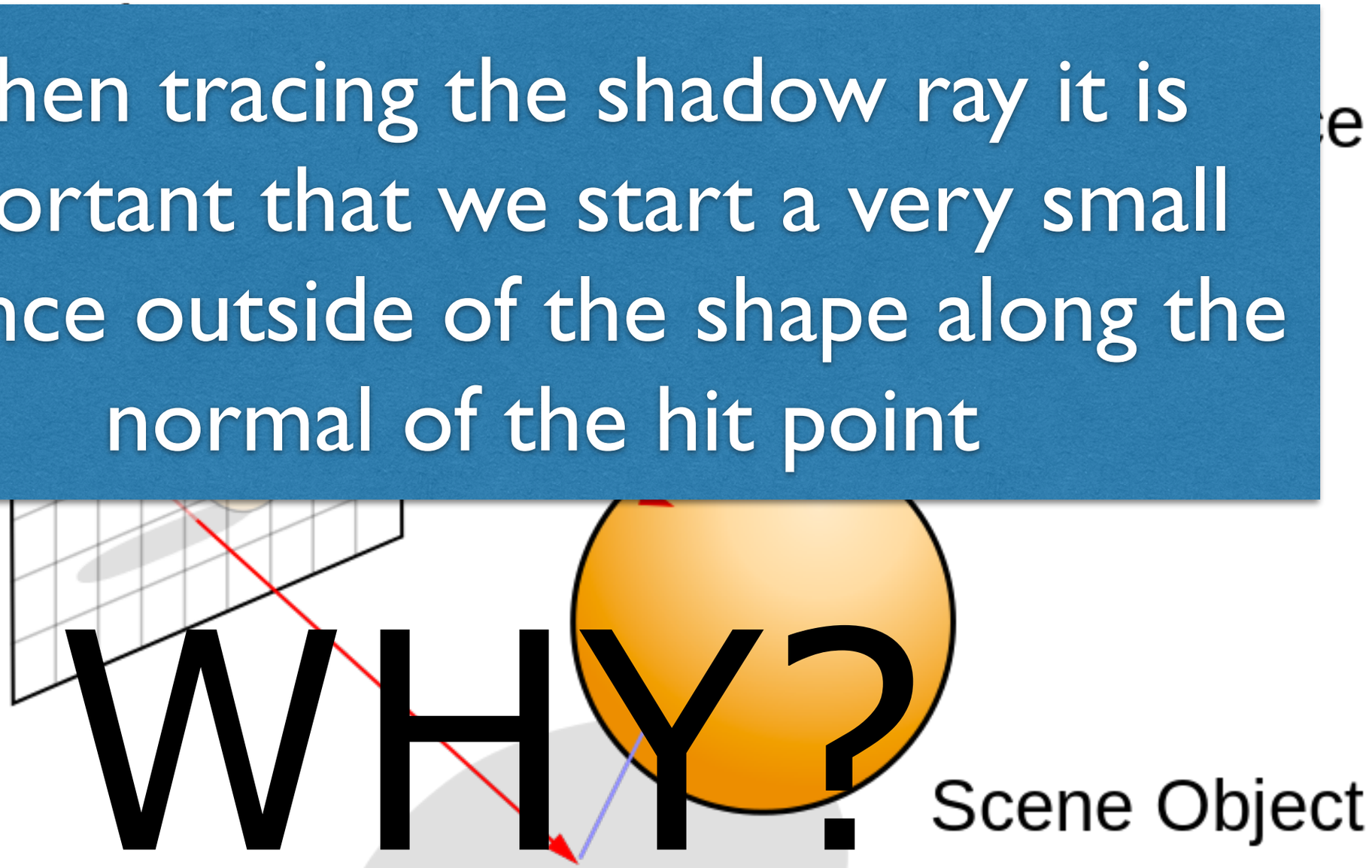
Basic colouring

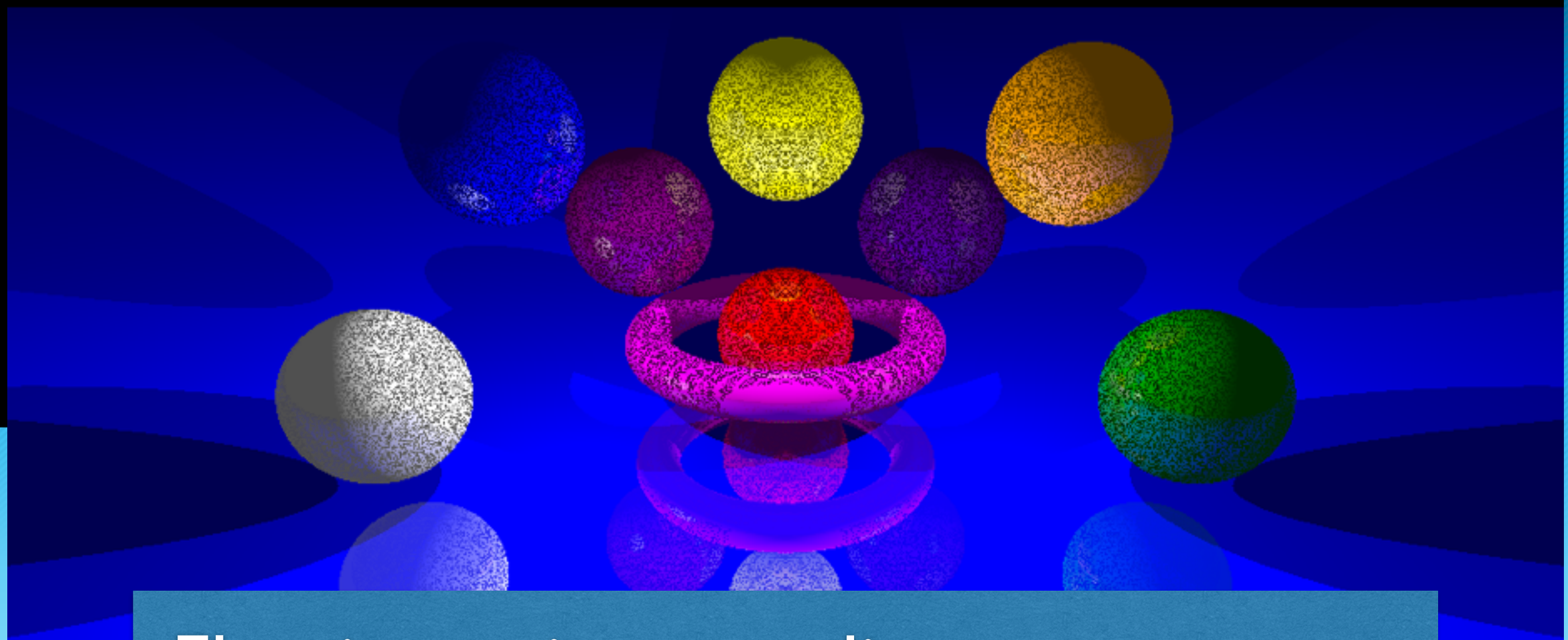
When tracing the shadow ray it is important that we start a very small distance outside of the shape along the normal of the hit point



Basic colouring

When tracing the shadow ray it is important that we start a very small distance outside of the shape along the normal of the hit point





Floating point rounding errors cause us to start our shadow rays inside the shapes

Drawing shadow ray

Always start your shadow rays slightly outside the shapes along the normal (about 0.0001 units)

Do not just subtract 0.0001 from the hit distance

IT UNIVERSITY OF COPENHAGEN

See how the light leaks into and out of the cylinder

Reflection

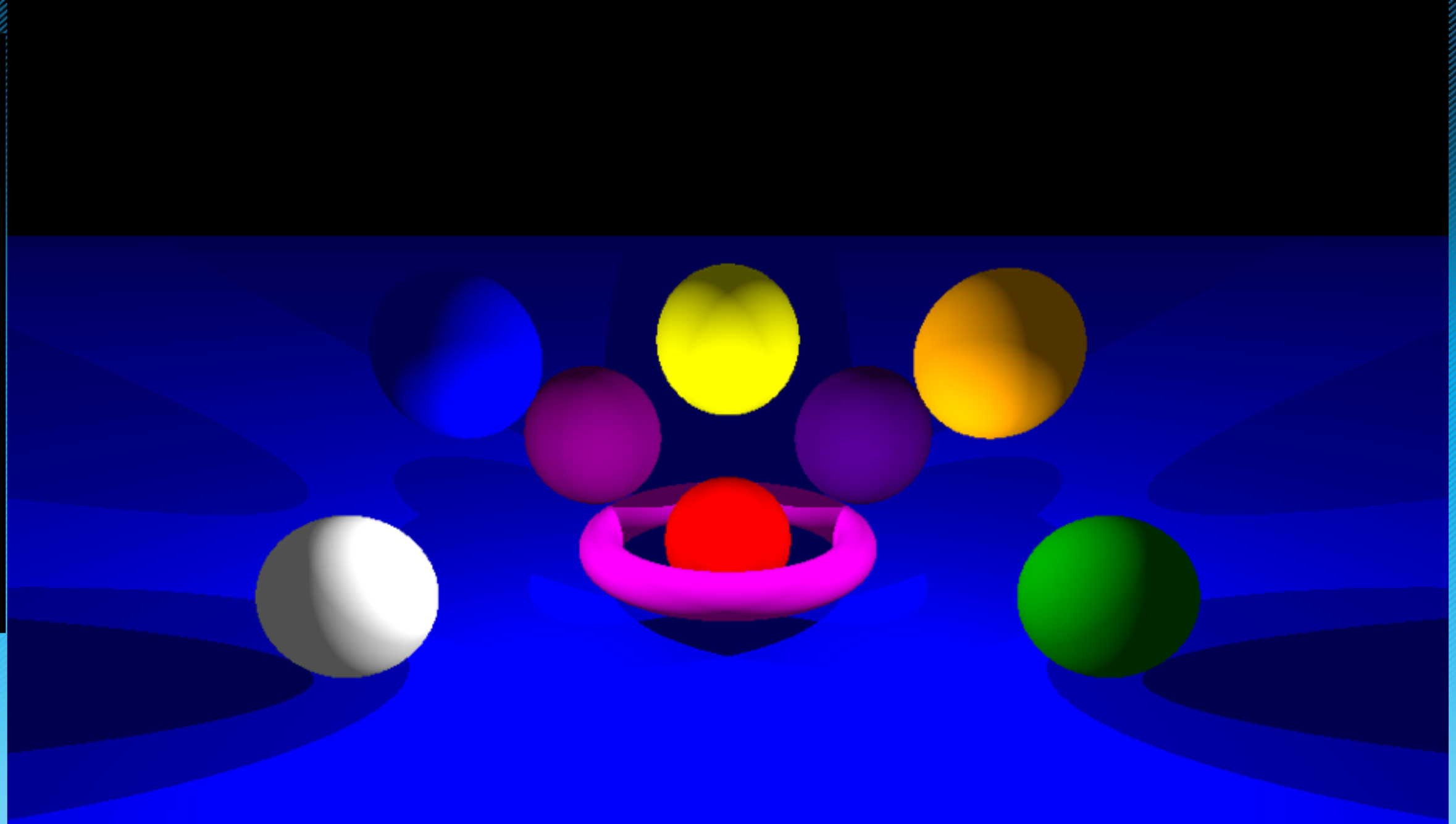
Reflection is remarkably easy

- If a material is reflective, send off another ray in the same angle with respect to the normal as you came in, but in the other direction
- Check what colour you receive
- Merge the colours with respect to how reflective the material is (value between 0 and 1)

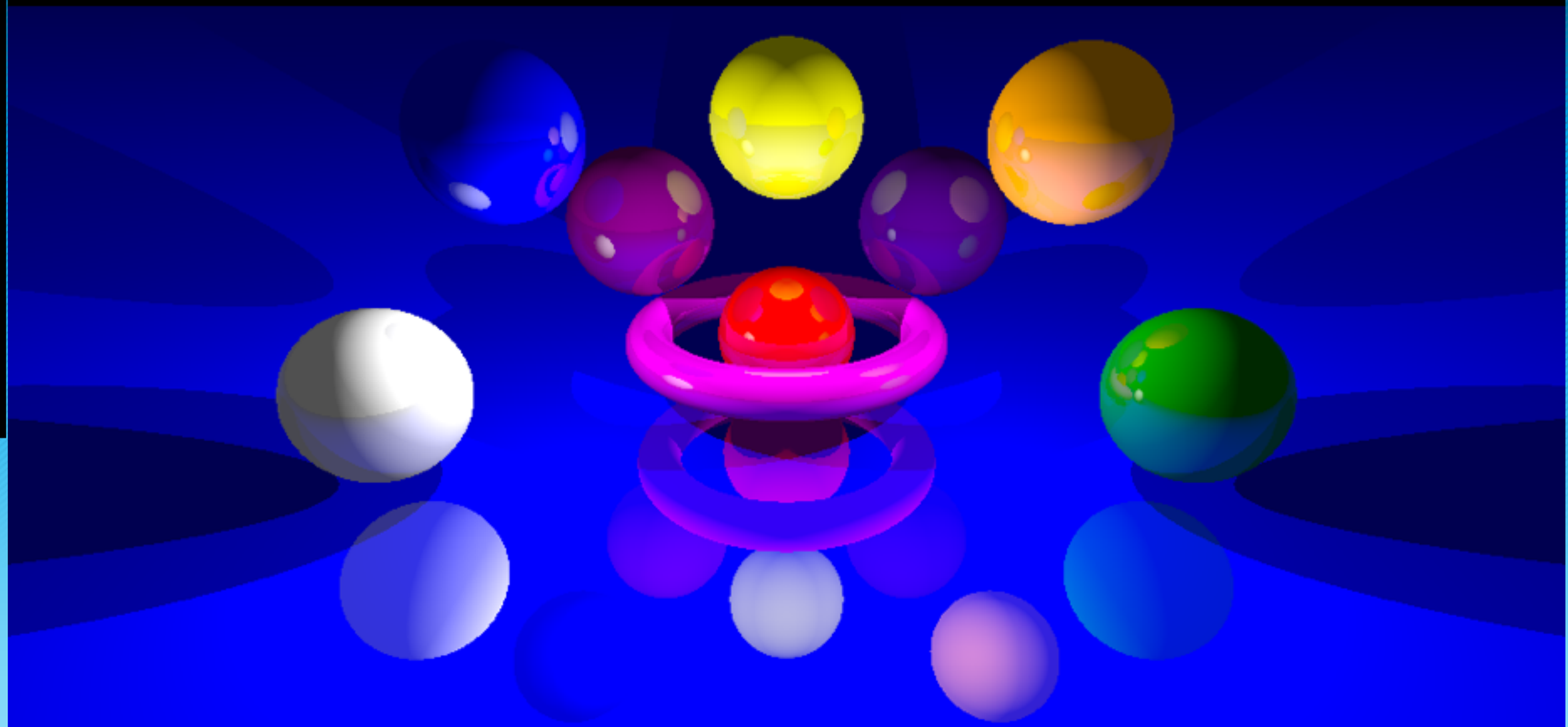
Reflection

Reflection is remarkably easy

- Beware!!!
- You must set a maximum number of reflections or you may loop forever. Besides, reflections have diminishing returns
- reflective the material is (value between 0 and 1)

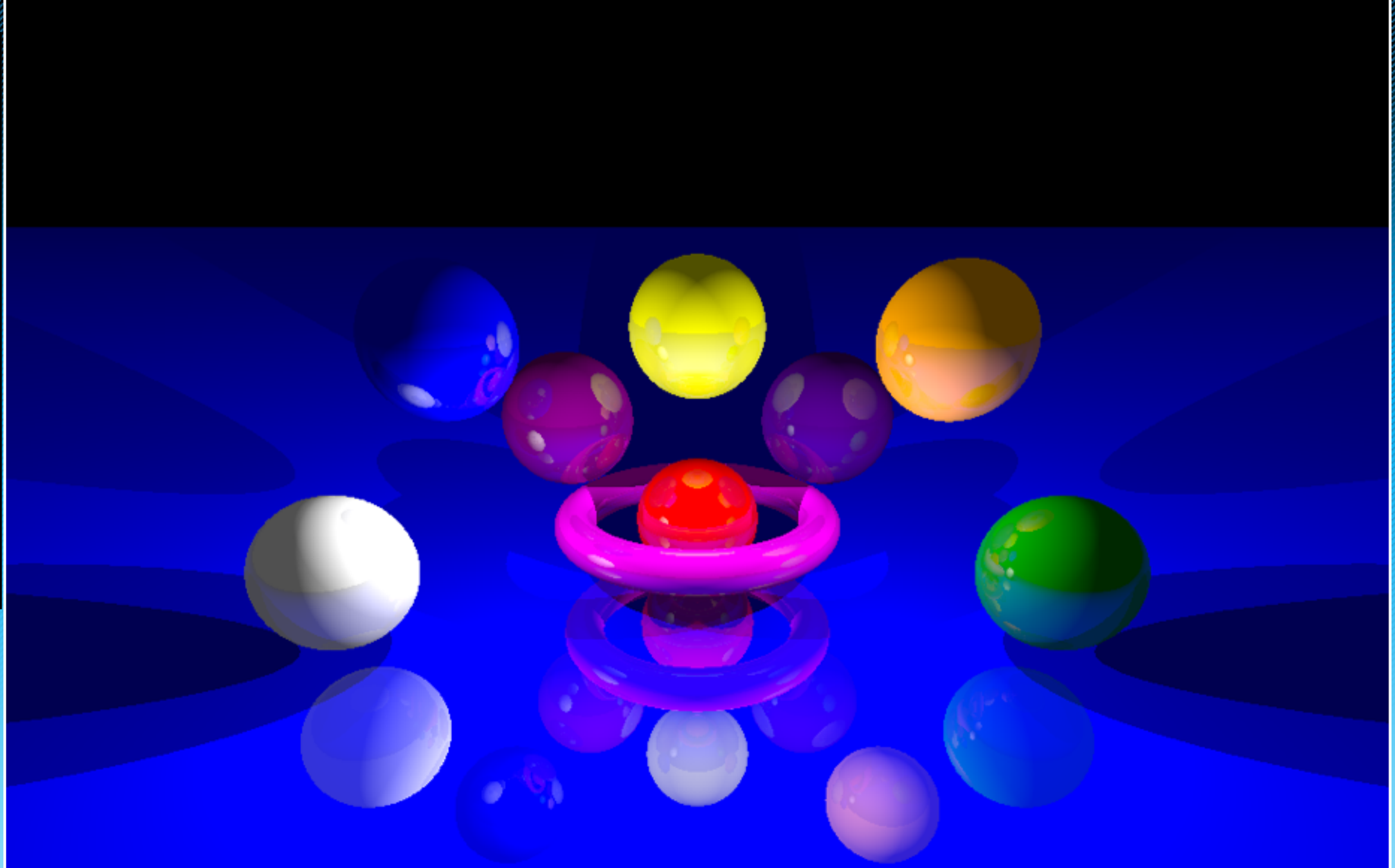


No Reflection



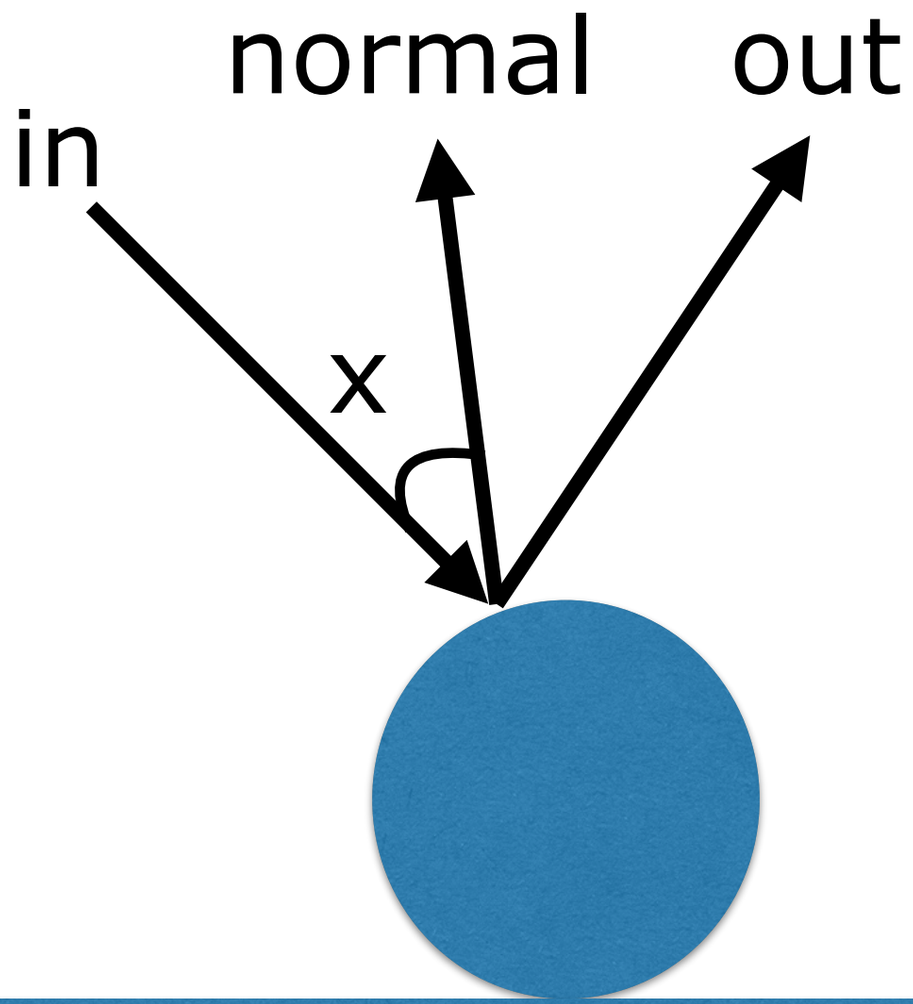
One Reflection

Two Reflections



Five Reflections

Calculating angles



x = in.direction * normal.direction
new direction = 2x
out = (in - 2x * normal)^

Mixing colours

To mix colours two colours with respect to a reflective surface scale the reflected colour with the reflective index, and your own colour with the index inverse

```
merge refl (r1, g1, b1) (r2, g2, b2) =
    (refl * r1 + (1 - refl) * r2,
     refl * g1 + (1 - refl) * g2,
     refl * b1 + (1 - refl) * b2)
```

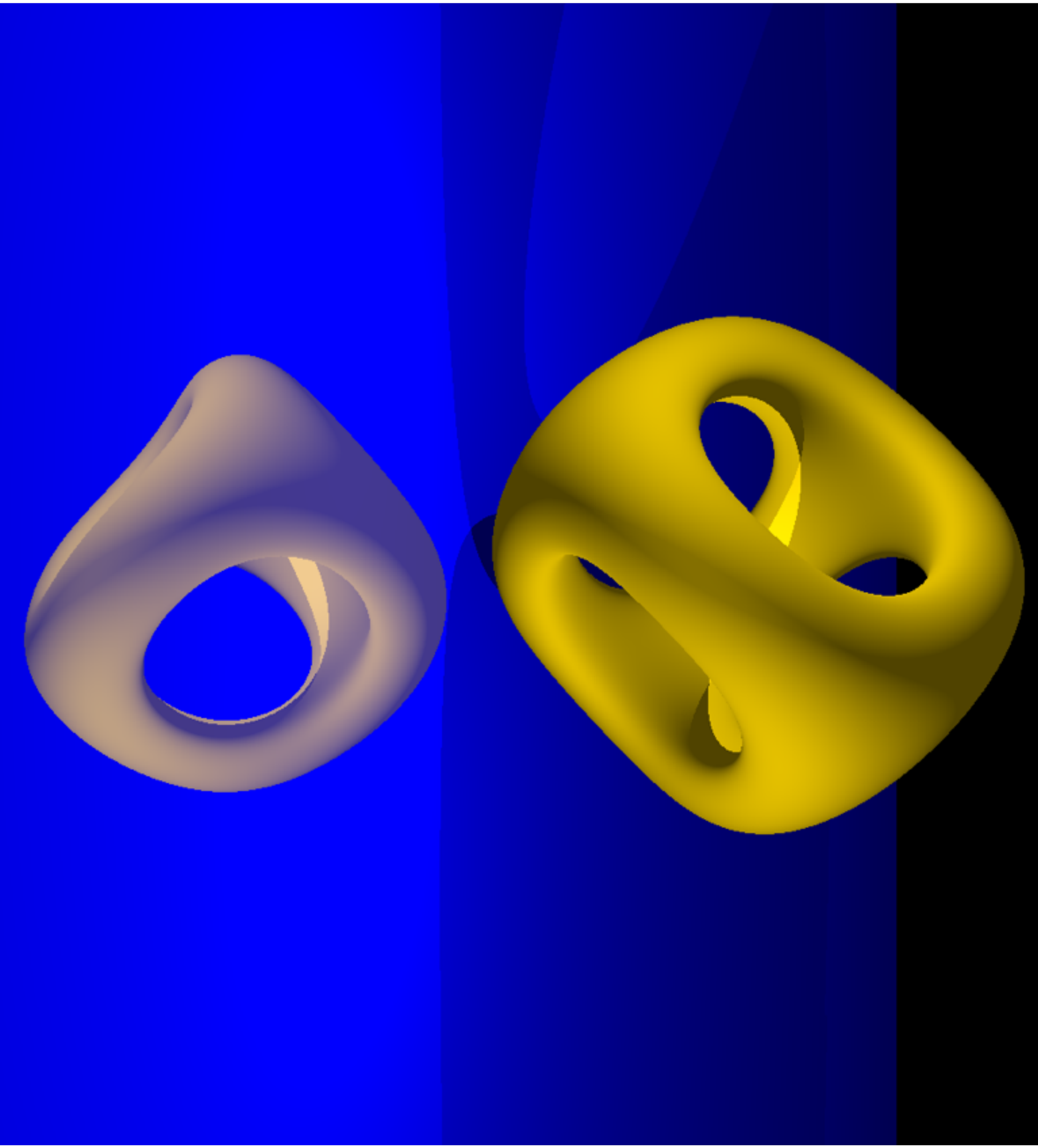
Recall that the reflective index is a value between zero and one

IT UNIVERSITY OF COPENHAGEN

Questions?

Ray Tracing

Implicit surfaces



Implicit surfaces

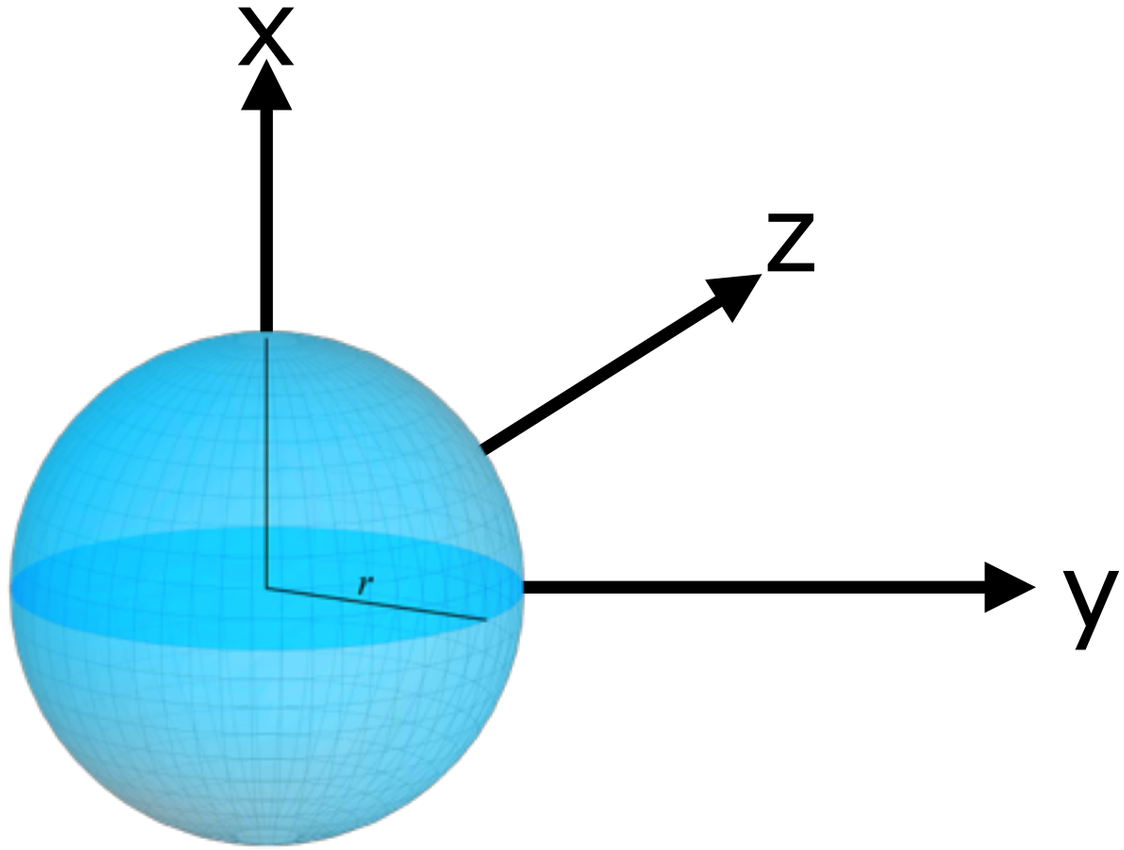
Implicit surfaces are equations in three dimensional space

$$x^2 + y^2 + z^2 - r^2 = 0$$

is the equation for a sphere

Implicit surfaces

$$x^2 + y^2 + z^2 - r^2 = 0$$



Implicit surfaces

$$x^2 + y^2 + z^2 - r^2 = 0$$

A ray has the equation

$$p + td$$

Where **p** is a point, **t** is a scalar distance (typically a double) and **d** is a normalised direction vector

Implicit surfaces

$$x^2 + y^2 + z^2 - r^2 = 0$$

A ray has the equation

$$p + td$$

If you squint hard enough you can see that the ray equation behaves like a combination of M (point movement) and multiplying a vector by a scalar

Implicit surfaces

$$x^2 + y^2 + z^2 - r^2 = 0$$

A ray has the equation

$$p + td$$

Where **p** is a point, **t** is a scalar distance (typically a double) and **d** is a normalised direction vector

Let’s calculate a hit function

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

A hit function takes an object and a ray where the point **p** and the direction vector **d** are known and computes a value for **t**. If such a non-zero value exists, then the ray will strike the object at a distance of **t** from the point **p** with **d** being the direction taken

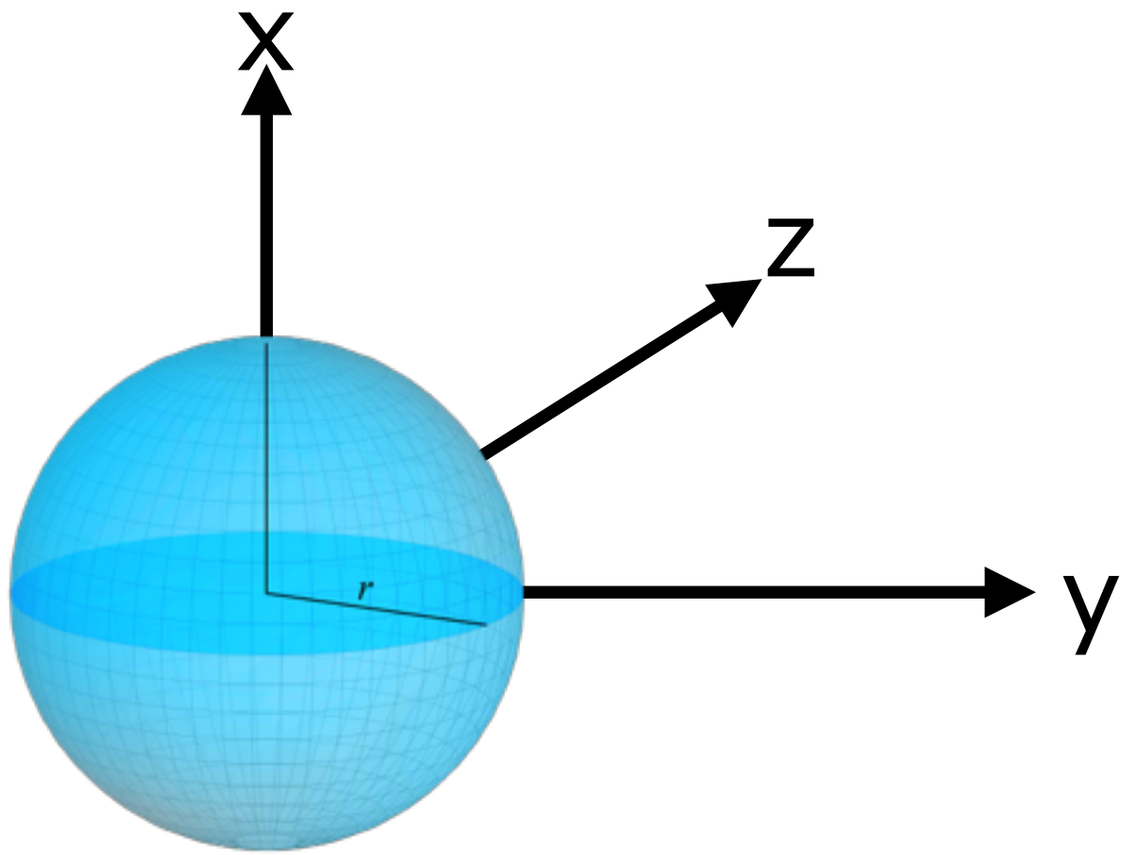
Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$



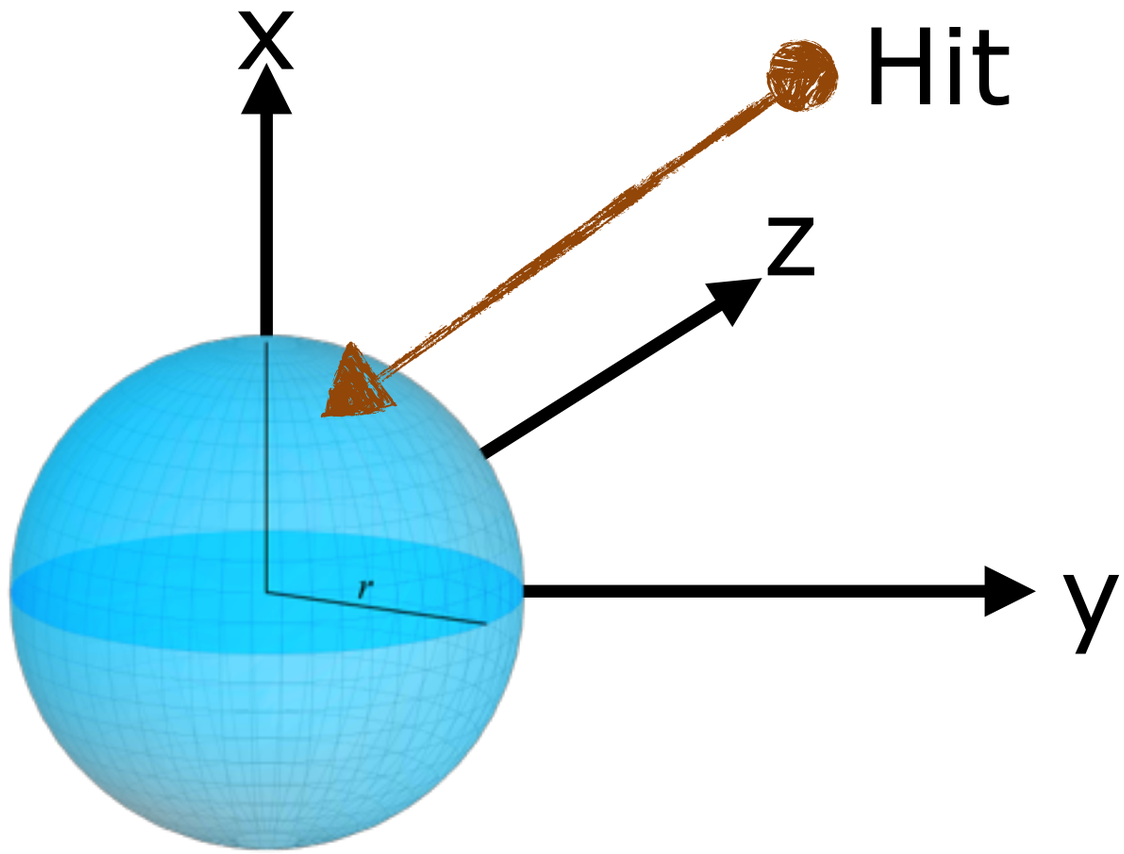
Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$



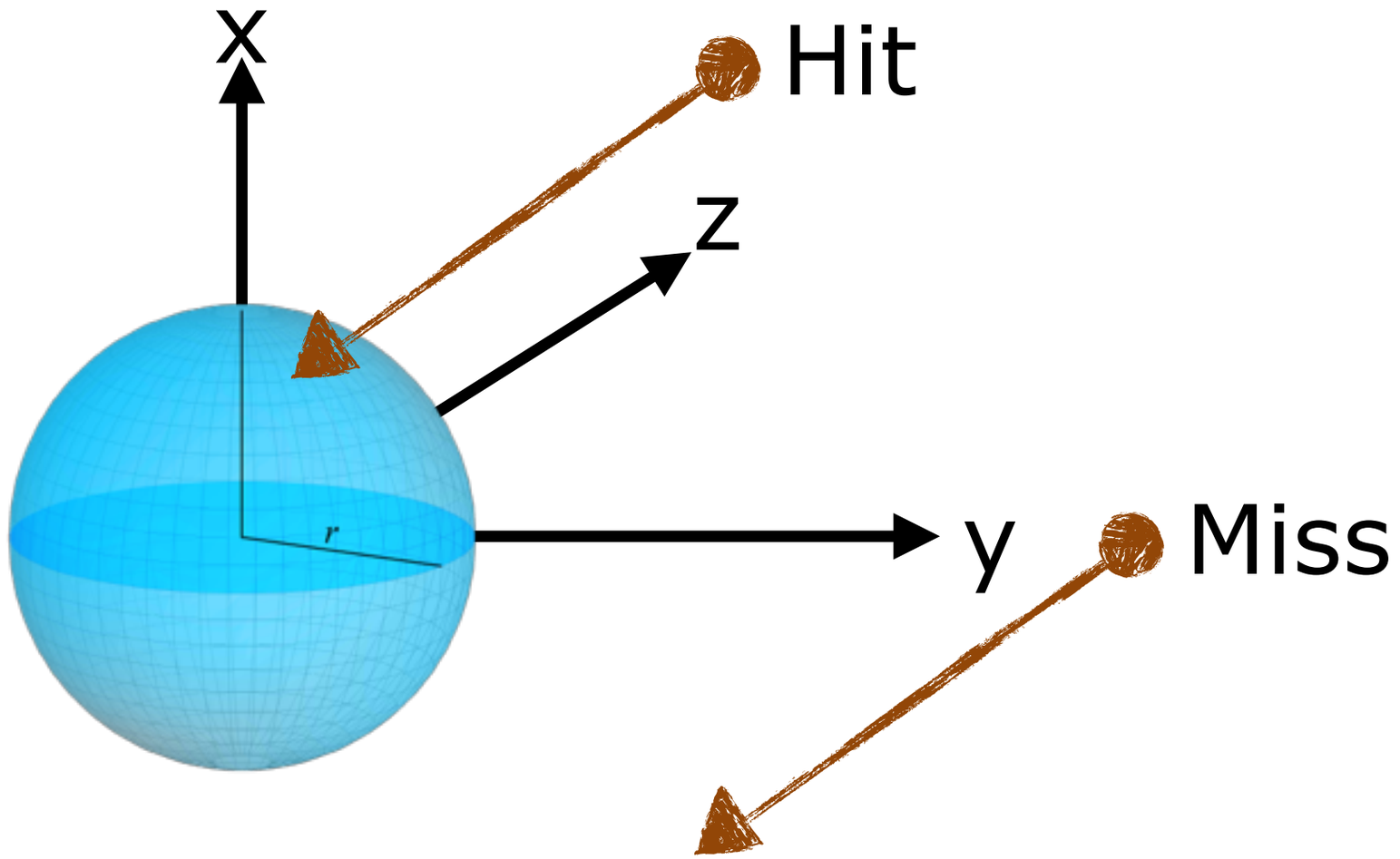
Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$



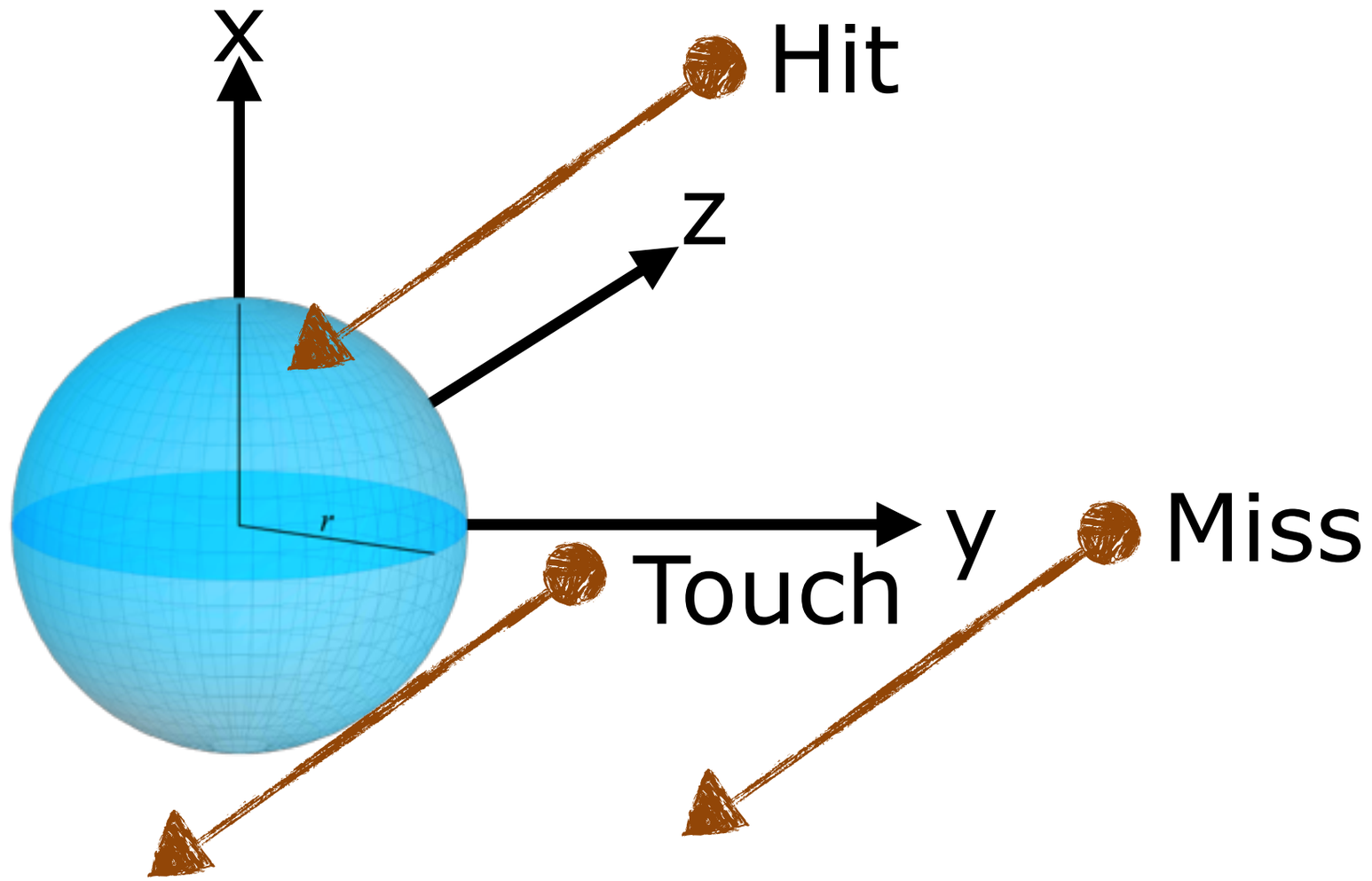
Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$



Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

Replace all occurrences of **x**, **y**, and **z** with the corresponding points from the point **p** and the vector **d**

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

Replace all occurrences of **x**, **y**, and **z** with the corresponding points from the point **p** and the vector **d**

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

$x^2 +$

$y^2 +$

$z^2 -$

r^2

Replace all occurrences of **x**, **y**, and **z** with the corresponding points from the point **p** and the vector **d**

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

$x^2 +$

$y^2 +$

$z^2 -$

r^2

Replace all occurrences of **x**, **y**, and **z** with the corresponding points from the point **p** and the vector **d**

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p_x + td_x$$

$x^2 +$

$y^2 +$

$z^2 -$

r^2

Replace all occurrences of **x**, **y**, and **z** with the corresponding points from the point **p** and the vector **d**

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

$$(p_x + td_x)^2 +$$

$$(p_y + td_y)^2 +$$

$$(p_z + td_z)^2 -$$

$$r^2$$

Replace all occurrences of **x**, **y**, and **z** with the corresponding points from the point **p** and the vector **d**

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

$$(p_x + td_x)^2 +$$

$$(p_y + td_y)^2 +$$

$$(p_z + td_z)^2 -$$

$$r^2$$

Simplify exponents

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

$$(p_x^2 + t^2 d_x^2) +$$

$$(p_y^2 + t^2 d_y^2) +$$

$$(p_z^2 + t^2 d_z^2) -$$

$$r^2$$

Simplify exponents

Implicit surfaces

Sphere

$$x^2 + y^2 + z^2 - r^2 = 0$$

Ray

$$p + td$$

$$(p_x^2 + 2p_xtd_x + t^2d_x^2) +$$

$$(p_y^2 + 2p_ytd_y + t^2d_y^2) +$$

$$(p_z^2 + 2p_ztd_z + t^2d_z^2) -$$

$$r^2$$

Simplify exponents

Implicit surfaces

$$(p_x^2 + 2p_x t d_x + t^2 d_x^2) +$$

$$(p_y^2 + 2p_y t d_y + t^2 d_y^2) +$$

$$(p_z^2 + 2p_z t d_z + t^2 d_z^2) -$$

$$r^2$$

This is a second degree polynomial

Implicit surfaces

$$(p_x^2 + 2p_x t d_x + t^2 d_x^2) +$$
$$(p_y^2 + 2p_y t d_y + t^2 d_y^2) +$$
$$(p_z^2 + 2p_z t d_z + t^2 d_z^2) -$$
$$r^2$$

This is a second degree polynomial

$$(d_x^2 + d_y^2 + d_z^2)t^2$$

Implicit surfaces

$$(p_x^2 + 2p_x t d_x + t^2 d_x^2) +$$
$$(p_y^2 + 2p_y t d_y + t^2 d_y^2) +$$
$$(p_z^2 + 2p_z t d_z + t^2 d_z^2) -$$
$$r^2$$

This is a second degree polynomial

$$(d_x^2 + d_y^2 + d_z^2)t^2 +$$
$$(2p_x d_x + 2p_y d_y + 2p_z d_z)t$$

Implicit surfaces

$$(p_x^2 + 2p_x t d_x + t^2 d_x^2) +$$
$$(p_y^2 + 2p_y t d_y + t^2 d_y^2) +$$
$$(p_z^2 + 2p_z t d_z + t^2 d_z^2) -$$
$$r^2$$

This is a second degree polynomial

$$(d_x^2 + d_y^2 + d_z^2)t^2 +$$
$$(2p_x d_x + 2p_y d_y + 2p_z d_z)t +$$
$$(p_x^2 + p_y^2 + p_z^2 - r^2)$$

Implicit surfaces

This is a second degree polynomial

$$(d_x^2 + d_y^2 + d_z^2)t^2 +$$
$$(2p_xd_x + 2p_yd_y + 2p_zd_z)t +$$
$$(p_x^2 + p_y^2 + p_z^2 - r^2)$$

Implicit surfaces

$at^2 + bt + c$

$$(d_x^2 + d_y^2 + d_z^2)t^2 + (2p_xd_x + 2p_yd_y + 2p_zd_z)t + (p_x^2 + p_y^2 + p_z^2 - r^2)$$

Solve the polynomial, and you have your hit function

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The smallest non-zero root (if any) is the distance the ray must travel before it intersects the sphere

Implicit surfaces

Get started!!!

Hard code the hit function for a sphere (you should anyway), point a camera at the origo and paint your favourite colour if you hit. If you have done everything correctly you should get a solid single-colour circle

WARNING!

Generating hit functions is expensive!
You must do this as a preprocessing step or your ray tracer will be unbearably slow

It is not uncommon for a hit function to be run hundreds of millions of times

Find the normal

Finding the normal of an implicit surface is relatively straightforward assuming you remember how to derive a polynomial

The normal of an implicit surface described by the polynomial p at a point [x, y, z] is the vector

{dp/dx, dp/dy, dp/dz}

Find the normal

Finding the normal of an implicit surface is relatively straightforward assuming you remember how to derive a

Th ace described by the polynomial p at a point [x, y, z] is the vector

{dp/dx, dp/dy, dp/dz}

The derivative of p with respect to x

Find the normal

Finding the normal of an implicit surface is relatively straightforward assuming you remember how to derive a

The derivative of p with respect to y

ace described by the polynomial p at a point [x, y, z] is the vector

{dp/dx, dp/dy, dp/dz}

Find the normal

Finding the normal of an implicit surface is relatively straightforward assuming you remember how to derive a

The derivative of p with respect to z

ace described by the polynomial p at a point [x, y, z] is the vector

{dp/dx, dp/dy, dp/dz}

The normal of a sphere

The equation for a sphere is

$$x^2 + y^2 + z^2 - r^2$$

Hence the equation for its normal is

$$\{2x, 2y, 2z\}$$

Once you have your hit point, substitute its coordinates into this vector to obtain the normal

The normal of a sphere

The equation for a sphere is

Key observation!!!!

You do not need to transform your equation to a polynomial to obtain the normal

the normal

Getting started

For polynomials with degrees smaller than four there exist formulas that solve them exactly. Use them!!!

For polynomials of degrees greater than four things get more difficult

Getting started

For polynomials with degrees smaller than four there exist formulas that solve them exactly. Use them!!!

For polynomials of degrees greater than four things get more difficult

High-degree polynomials

- Find the number of roots between zero and an arbitrarily large number (Sturm sequences)
- Iterate by binary partitioning until you find an interval containing the closest root
- Use Newton Raphson's method to converge on that root

Sturm sequences

- Obtain a sequence of polynomials of decreasing order (we will soon cover how)
- Evaluate each polynomial in the sequence for the minimum value of your interval and see how many times the result switches signs (a)
- Do the same for the maximum value (b)
- The number of roots is a - b

Obtain polynomials

$$p(x) = x^4 + x^3 - x - 1$$

The first member is the polynomial, the second is its derivative

$$p_0(x) = p(x) = x^4 + x^3 - x - 1$$
$$p_1(x) = p'(x) = 4x^3 + 3x^2 - 1$$

The rest of the chain is created using polynomial long division where

$$p_n = p_{(n-2)} / p_{(n-1)}$$

Polynomial long division

Polynomial long division works like regular long division

$$
\begin{array}{r}
x^2 + x + 3 \\
\hline
x-3 \,\big)\, x^3 - 2x^2 + 0x - 4 \\
\underline{x^3 - 3x^2} \\
+x^2 + 0x \\
\underline{+x^2 - 3x} \\
+3x - 4 \\
\underline{+3x - 9} \\
+5
\end{array}
$$

Polynomial long division

Polynomial long division works like regular long division

Hint

There is an algorithm on Wikipedia

+5

Obtain polynomials

$$p(x) = x^4 + x^3 - x - 1$$

The complete Sturm sequence is

$$p_0(x) = x^4 + x^3 - x - 1$$
$$p_1(x) = 4x^3 + 3x^2 - 1$$
$$p_2(x) = \tfrac{3}{16}x^2 + \tfrac{3}{4}x + \tfrac{15}{16}$$
$$p_3(x) = -32x - 64$$
$$p_4(x) = -\tfrac{3}{16}$$

In the interval -100 to 100 there exists 2 roots

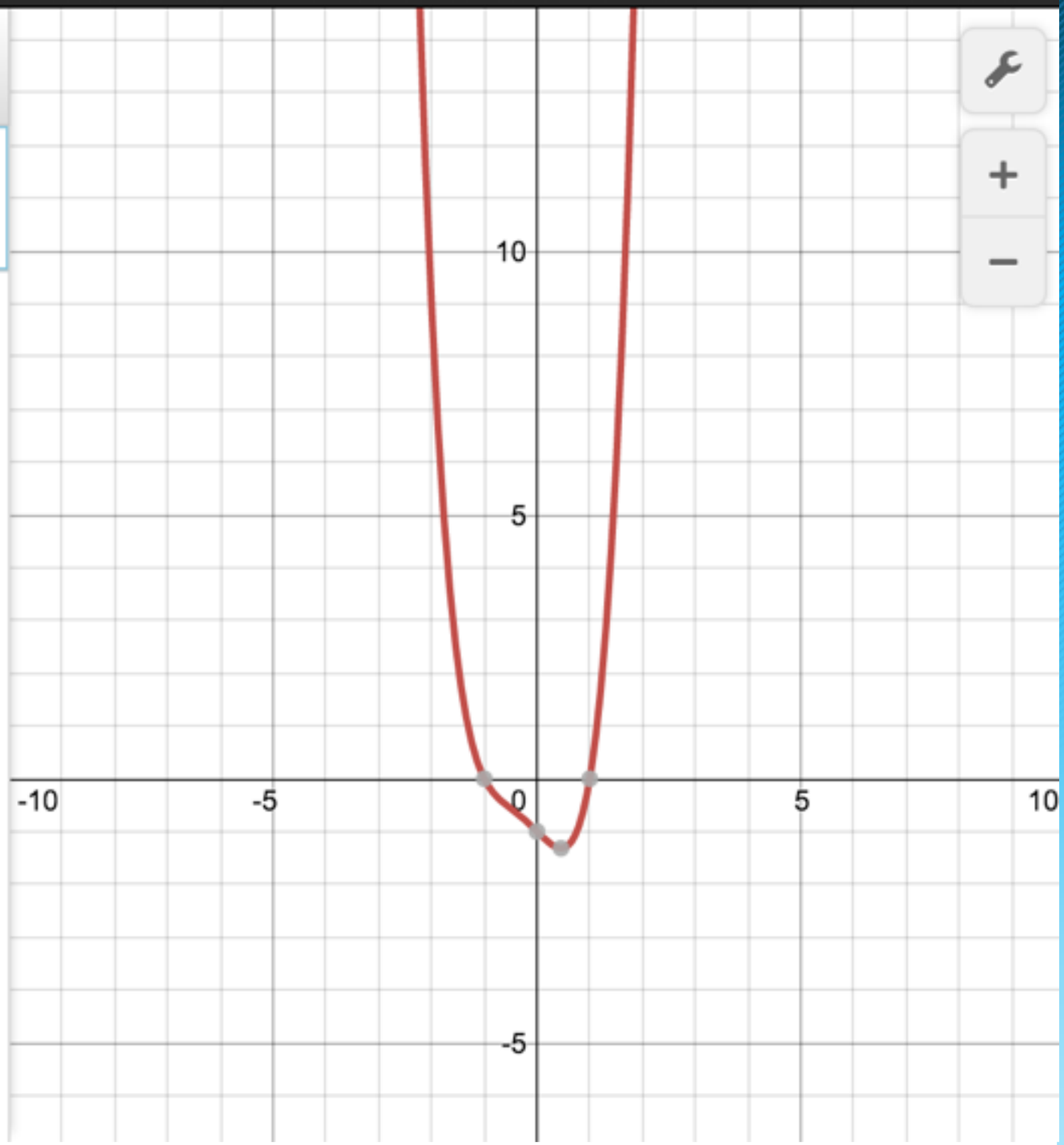
https://www.desmos.com/calculator

Untitled Graph

desmos

Create Account or Sign In

$x^4 + x^3 - x - 1$



IT UNIVERSITY OF COPENHAGEN

Questions?

Newton Raphson

Newton Raphson quickly converges on a root, provided you start "close enough"

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Newton Raphson

Newton Raphson quickly converges on a

Be careful!!!

You may converge on the wrong root
You may diverge
If any of this happen, start again with a better guess

Thursday

- Affine transformations
- Constructive Solid Geometry