# Fundamentals of Graphics Rendering

Computer Graphics - SET08116

EDINBURGH NAPIER UNIVERSITY

# Outline

# What do we mean by Real-Time Rendering?

- What do you think we mean by the term real-time? What systems are generally considered real-time?
- What do you think we mean by real-time rendering?
- What do you think we mean by interactive?
- Do you think there is a frame rate (Frames Per Second or FPS) we can use to definine interactive?
- Actually, there is. That frame rate is 6 FPS. Are you surprised about how low that number is?

# Frame Rates and Refresh Rates

- Who knows the frame rate (in FPS) of film?
- It is 24 FPS. This provides a sufficient frame rate to give the viewer the perception of motion.
- Who knows the typical frame rate (in FPS) associated with games?
- Typically games try and hit 30 FPS or 60 FPS.
- Does anyone know why games require a higher FPS than film?
- It is actually because standard cameras will capture motion blur. Motion blur provides the eye with information that it can use to fill in the blanks. Graphics rendering does not have motion blur - but we can fake it.
- Who knows what we mean by refresh rate for monitors?
- What is the typical refresh rate of a monitor? What does this mean?
- Typcially, monitors operate at 60Hz (or 60 FPS). This is as fast as the monitor can refresh the image on the screen.

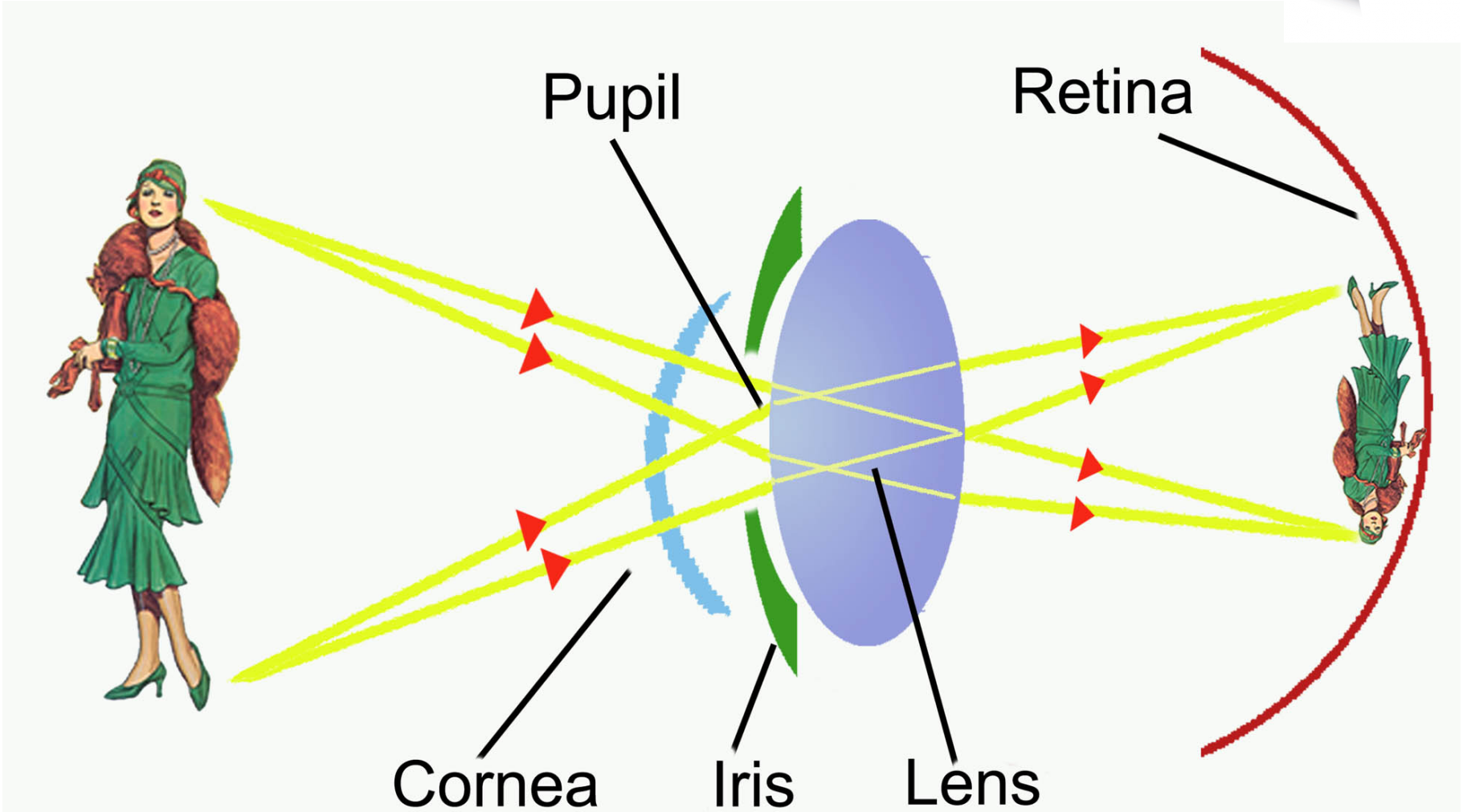
# Operations per Frame

- Let's say that our application runs at the maximum refresh rate for a common monitor. How many milliseconds (ms) per frame do we have?
- 16.66ms
- Let us now consider a pretty powerful CPU running at 3GHz. This means there are 3 billion operations per second. How many instructions per frame do we have?
- We divide the 3 billion by 60 to give ourselves 50 million instructions. This seems like a lot to play with.
- OK, now what is the standard resolution of a 1080p HD screen?
- It is $1920 \times 1080$. This is 2073600 pixels. So how many instructions per pixel do we have?
- We only end up with 24 instructions per pixel! Not very many to play with.

# The Eye

- Who knows how the eye works?
- The eye takes in light through the pupil. The light travels through the eye before hitting the retina (the back of the eye). The retina is covered with rods and cones, which are activated by the light providing our brain with the information to create an image.
- What are the main components of the eye which allow us to perceive an image?
- There are two main parts
  1. The pupil which captures light through a lens
  2. The retina which receives the light to create an image
- We therefore have a lens and a sensor.

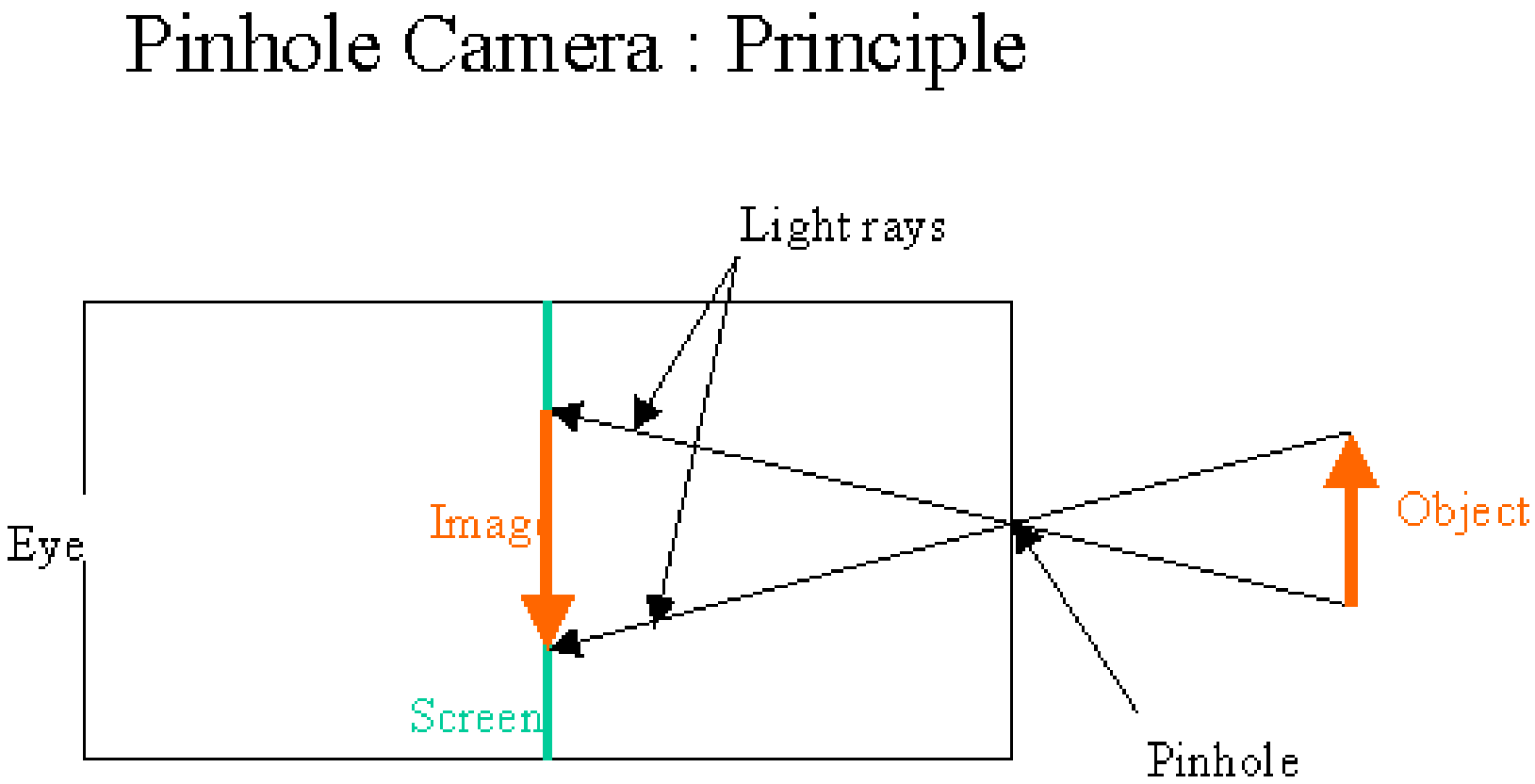
# How the Eye Works



# Pinhole Camera

- Now let us look at cameras and how they relate to the eye. What are some of the basic components of a camera?
- The two we are most interested in are the lens and the image sensor. These directly relate to our simplified understanding of the eye.
- Now let us consider a pinhole camera (we will look at an image shortly).
- A pinhole camera has a pinhole in the front to receive light. This light is then projected onto the back of the camera, creating an image.
- In other words we have a lens (the pinhole) and a sensor (the back of the camera where the image is projected).
- We are going to start calling the lens and sensor the terms we will use for the rest of the module - the view and the projection.
- Read Section 1.4 of Interactive Computer Graphics for more details on pinhole cameras.

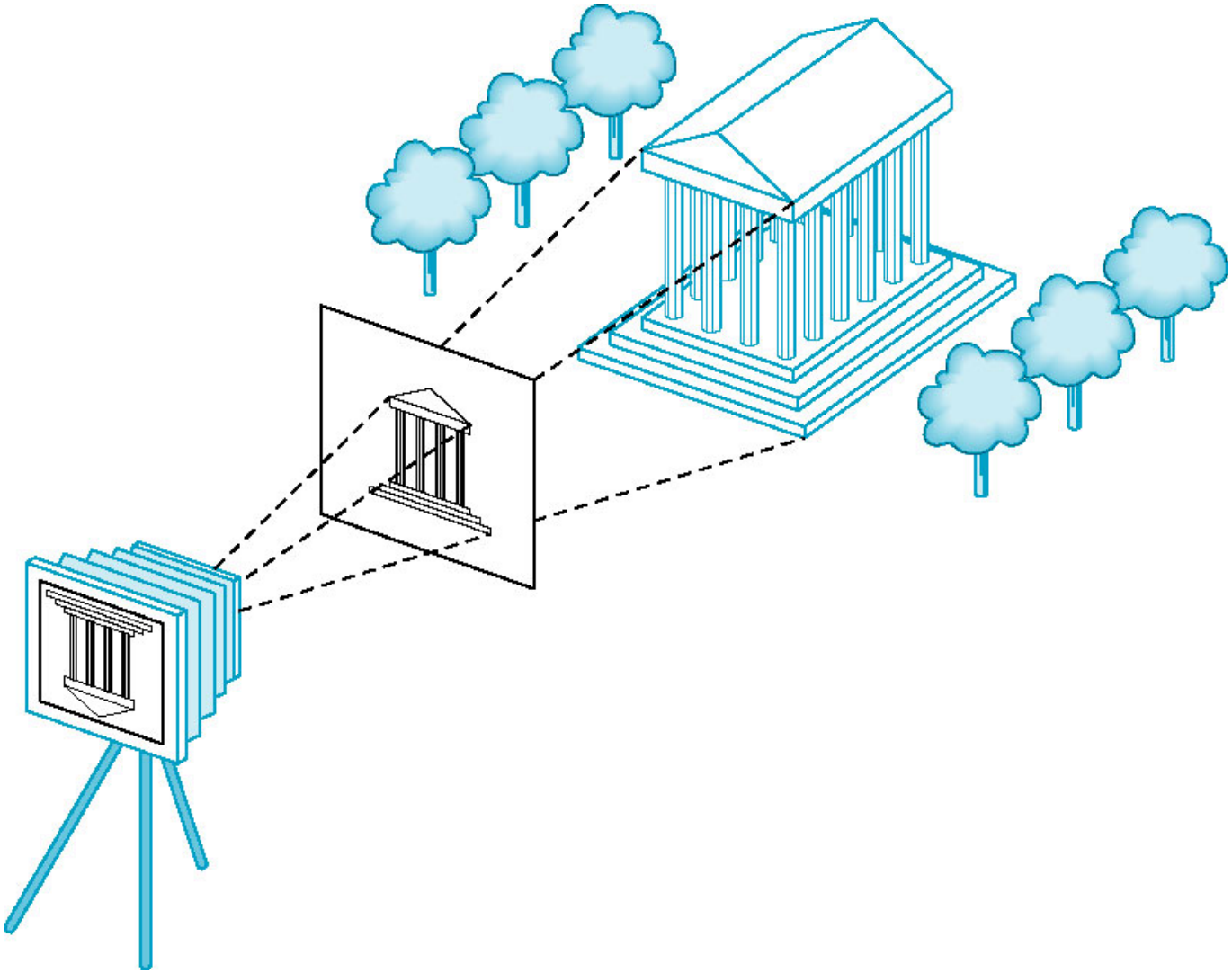
# Pinhole Camera



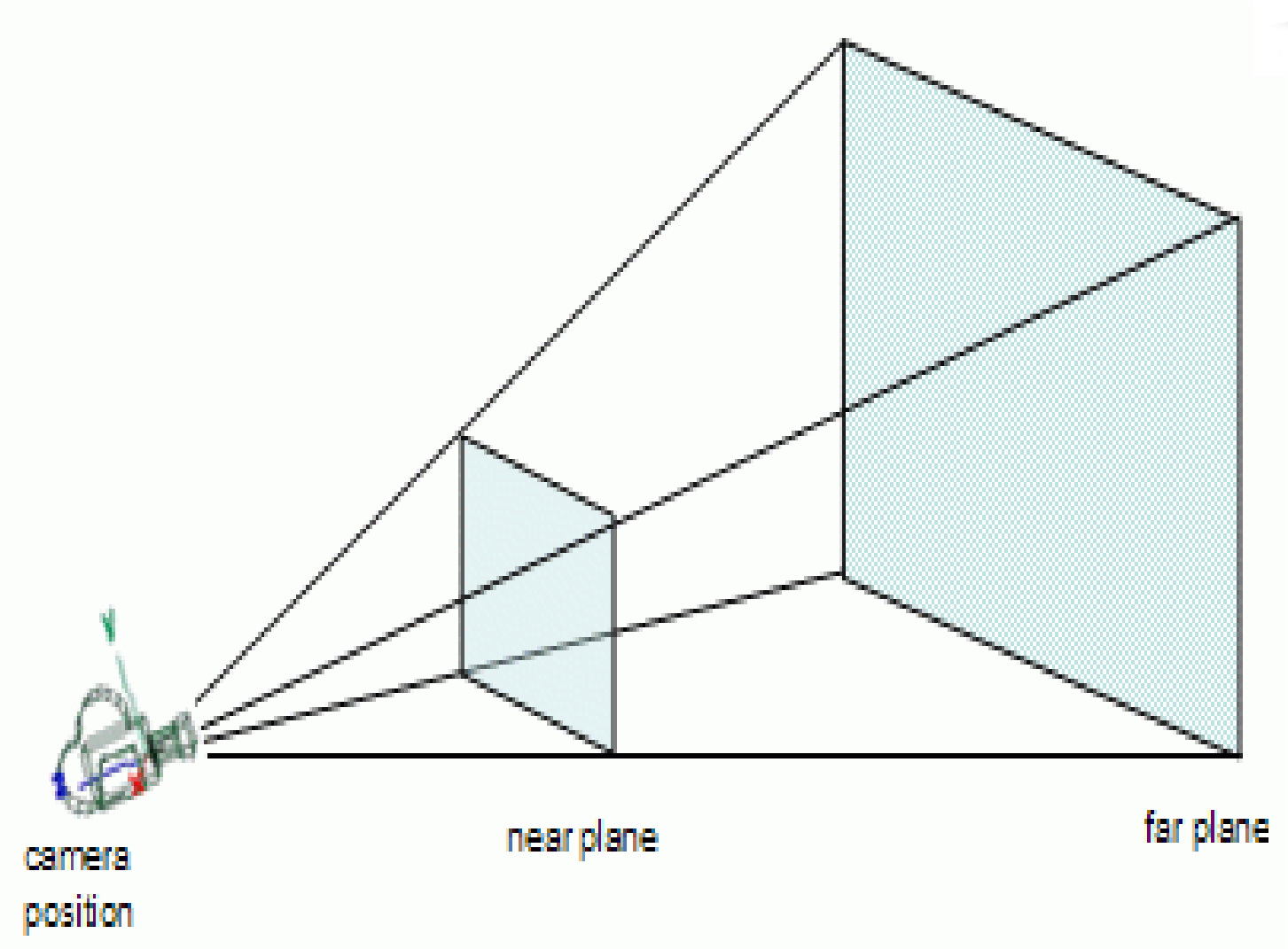
# Synthetic Camera Model

- One thing you might have noticed when we have described the eye and a camera is that the projected imaged always forms upside down. This is due to the way light travels from the eye to the projector.
- For the eye, the brain inverts the image for us. For a camera, the film can be flipped or the image inverted on the device.
- For 3D rendering, we take a different approach - the image isn't projected past the lens but in front of it - the image on the next slide will explain.
- Knowing now what you do about the eye and cameras, can you guess what information we need to have to define the lens?
- We need its world position, the direction it is facing, and its orientation. Can you guess the information required for the projection?
- This does depend somewhat on the projection technique used, but for our purposes we need the field of view (the angle it can see), the aspect ratio, the near plane position (where we are projection to) and the far plane position (how far can we see).

# Synthetic Camera Model



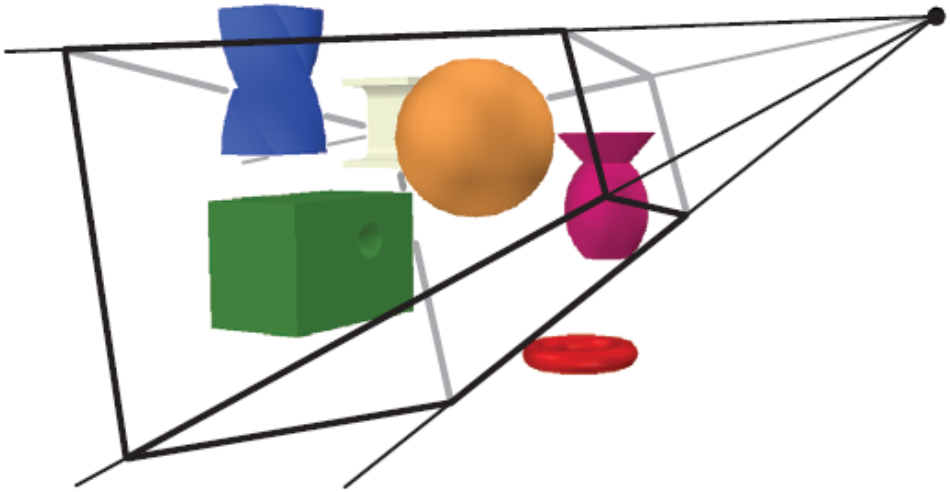
# View Frustum

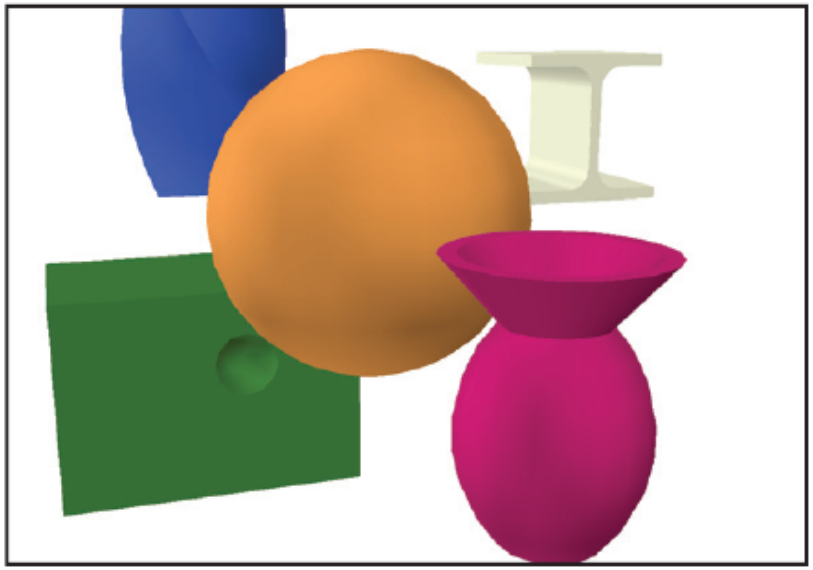


# Objects and Viewers

- Now that we have defined what we mean by a camera in a 3D scene, we can move onto a definition for an object.
- What information do you think we need to define an object in a 3D scene?
- Some of these you will get, some you won't:
  - The geometry that makes up the object.
  - The position, orientation and scale of the geometry in the world. This is what we traditionally call the transform (either model or world) of the object.
  - The material of the object.
- Combining these three ideas together gives us what we will call a mesh. A mesh is a combination of geometry, transform and material.
- We are actually concerned with how our objects look from the point of view of the camera. This means we perform another transform to move the objects into the camera's viewpoint.
- The final stage is to take the camera's view and project it onto the screen. This requires another transform. The combined transform through model, view and projection is normally combined. The is called the model-view-projection transform (or world-view-projection in DirectX).

# Objects and Viewers





# 3D Scene

- A 3D scene is just a collection of meshes and a camera to view them with. It is really that simple. The rest is really about how the object looks (the material).
- In general, artists generate geometry and the materials to go with them. As a programmer your main concern is with importing these assets and placing them into the world.
- An understanding of geometry is useful, and becomes essential for some of the more advanced rendering techniques.
- An understanding of lighting, texturing and other physical material concepts is also useful, and again becomes essential for many rendering techniques.
- We will be returning to geometry and materials many times in the module to build up your understanding. However, you should have an understanding of the fundamentals after today.

# 3D Scene

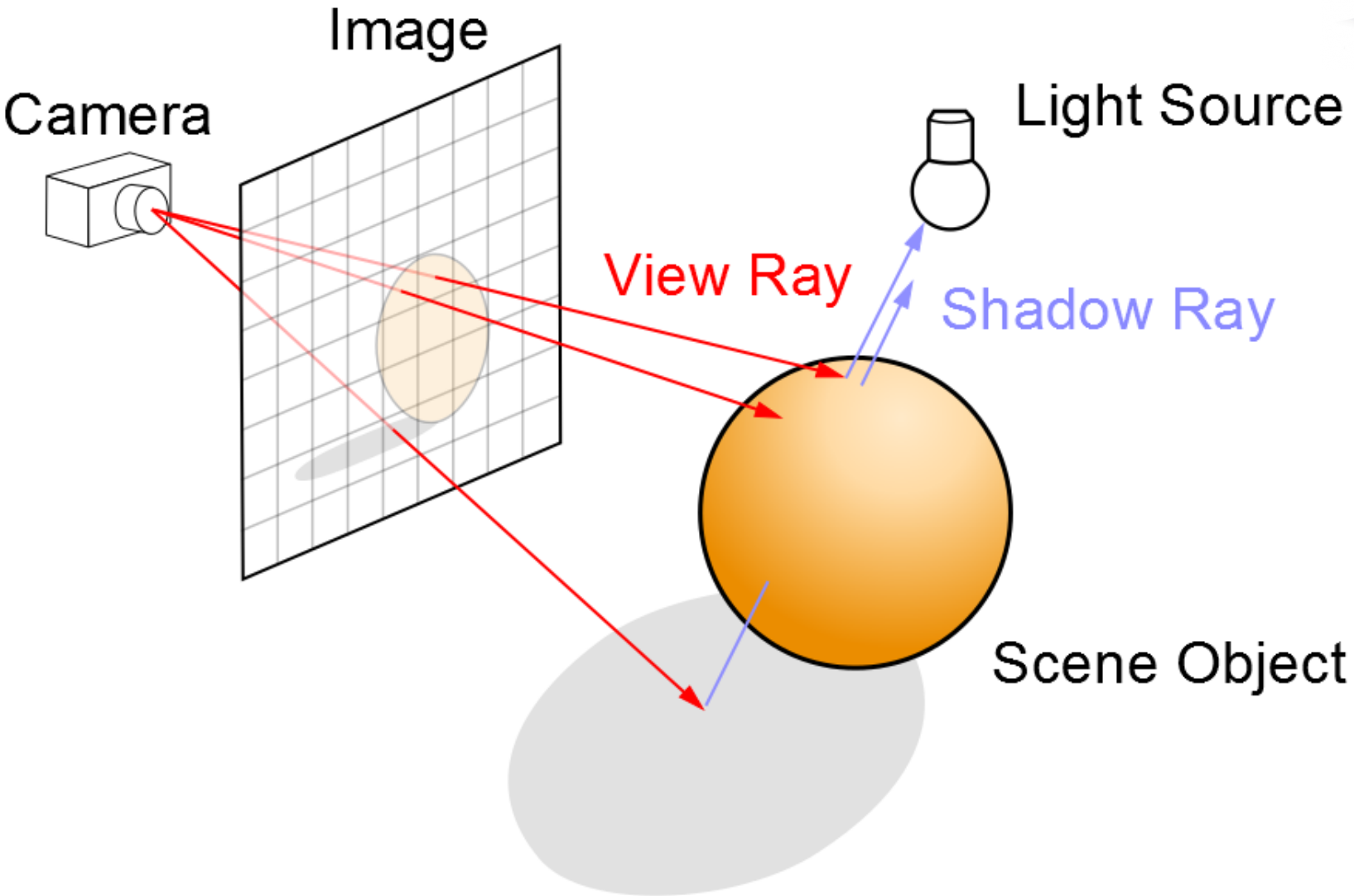
# Light and Viewers

- One of the most important parts of a 3D scene is the lighting
- Lighting, and in particular how light reacts with an object, is very important for scene realism.
- Light is esentially photons interacting with objects and eventually entering our eye, providing us with an image.
- One method we could use to calculate scene lighting is to track all the photons in a scene and determine the colour of the light that enters the eye.
- As a quick aside, does anyone know how many photons are generated by a standard 40 watt lightbulb?
- A 40 watt lightbulb generates $10^{19}$ photons. How many computers do you think it will take to compute the light for a scene using this approach?
- Hundreds of thousands of worlds worths of computers. It really is a crazy computationalal job.

# Reversing the Process

- So computing all the photons in a scene is a crazy idea.
  So how do we go about calculating the light for a scene?
- First of all, all we really care about are the photons that enter the eye. That will significantly reduce the number of photons we have to calculate.
- Since we are only concerned with the photons that enter the eye, we can effectively work in reverse. We send rays out from the eye (each pixel on the final screen) and determine what colour it should be.
- The colour is based on the object closest to the eye and how it is interacting with the lights from the viewer's point of view.
- Doing this properly (ray casting) will give us a very realistic looking scene. In real-time rendering we simply take each object and project it onto the screen, using the information we have to determine the individual pixel colours (more on this in later lectures).
- We essentially render each object to the screen, colouring in pixels as we go. This provides us with the final output image we expect

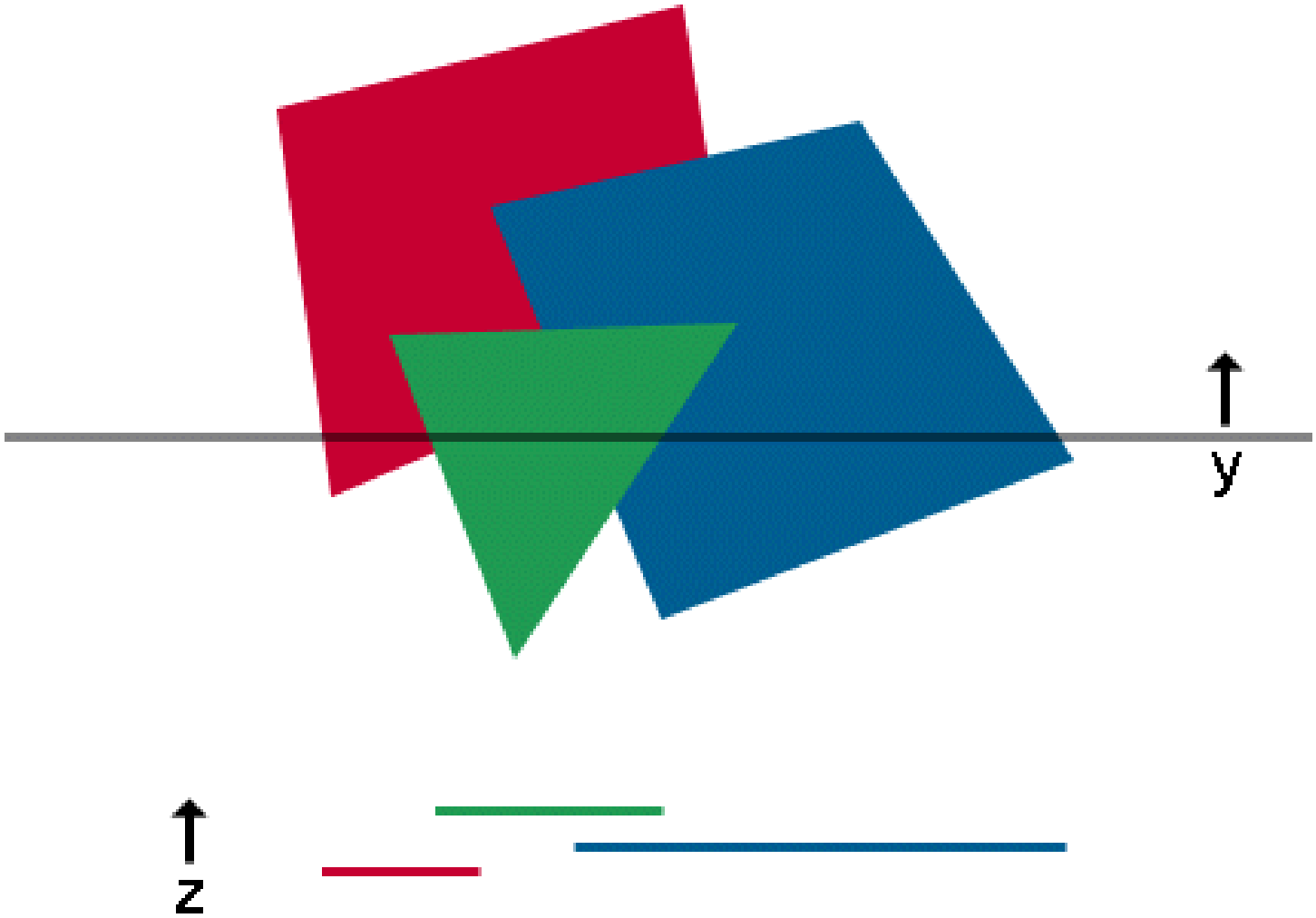
# Reversing the Process



# Painter's Algorithm

- Our current understanding is that we simply render each object to the screen one at a time. Our final image is based on the pixel colours generated.
- So how does this work in practice. Let us look at a simple algorithm.
- First of all, we could just render everything in any order? What do you think is the problem with doing this?
- We have no idea of what order everything should be drawn in. Something at the back could be rendered last, overwriting objects in front of it. So how do we overcome this?
- We can use a technique called the painter's algorithm. Here, objects are sorted into order, the furthest away being rendered first and the nearest last. This means that a pixel colour will be represented by the nearest object.
- Do you see any problems with this approach?
- What if objects are overlap each other in depth? How do we overcome this?

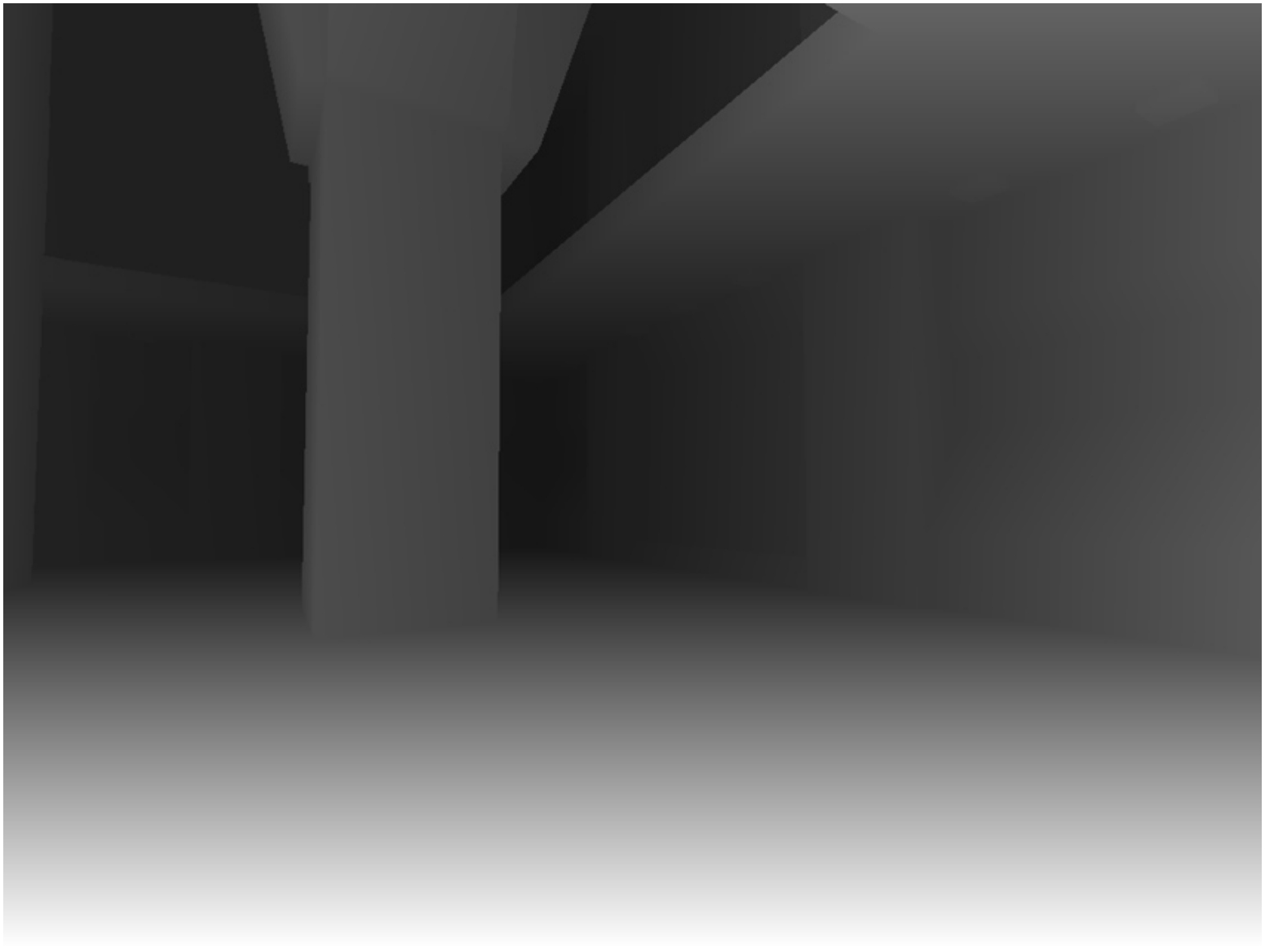
# Painter's Algorithm



# Depth Buffer

- Graphics rendering uses a technique called a depth buffer (or Z-buffer).
- As well as storing the colour of a pixel, the current depth of the pixel is also stored. This means we can check the current depth of any pixel we try and write to and determine whether to overwrite the colour.
- The depth buffer contains values ranging from 0.0 (near the eye) to 1.0 (far from the eye). This means that for every pixel, when using depth buffering we have 5 floating point values (4 for colour and 1 for depth).
- Do you think there are any further optimisations we can think about when using a depth buffer?
- It is quicker to check the depth buffer and perform no write than to check the depth buffer and write a new colour and depth value. What does that mean for the order of drawing?
- We should draw nearest first, the reverse of the painter's algorithm.

# Depth Buffer



# Summary

- You should now have a fundamental understanding of graphics rendering basics. In particular, our focus has been on real-time rendering. You should have a basic understanding of what we mean by real-time in a number of contexts.
- You should also have an understanding of the eye and cameras and how they are related. In particular, you should know what we mean by the synthetic camera model.
- You should also have an understanding of how a 3D scene is generated and how we go about defining objects in our scenes.
- You should also know what we mean by lighting and an initial knowledge of how lighting works in graphics rendering.
- Finally, you should have an understanding of depth and how we solve depth issues in 3D rendering.

# Recommended Reading

- Chapter 1 of Interactive Computer Graphics by Edward Angel. This will give you a solid understanding of computer graphics basics.
- Chapter 1 of Real-Time Rendering. This again will introduce you to some of the basic concepts we have covered today.