Game Engines

CMPM 164, F2019

Prof. Angus Forbes (instructor)
angus@ucsc.edu

Montana Fowler (TA)
mocfowle@ucsc.edu

Website: creativecoding.soe.ucsc.edu/courses/cmpm164
Slack: https://ucsccmpm164.slack.com

Lab Sessions

Lab:

Ming Ong Computer Center, Windows Lab – Merrill 103

Tuesdays 11am-12noon
Wednesdays 3pm-4pm
Thursdays 1pm-2pm
(**Montana's office hours are also held during the Thursday lab**)

Lab website / Direction:

https://its.ucsc.edu/computer-labs/descriptions/mingong.html

Ray Tracing

1968 - Arthur Appel, first ray casting implementation

1979 - Turner Whitted, first recursive ray tracing algorithm, including additional bounces to calculate reflection, refraction, and shadows

1982 - Ohmura Kouichi, Shirakawa Isao and Kawata Toru, first animated ray tracing prototype on a supercomputer

1984 - Robert Cook, Tom Porter, and Loren Carpenter extended ray tracing to render “fuzzy” phenomena, such as motion blur and soft shadows

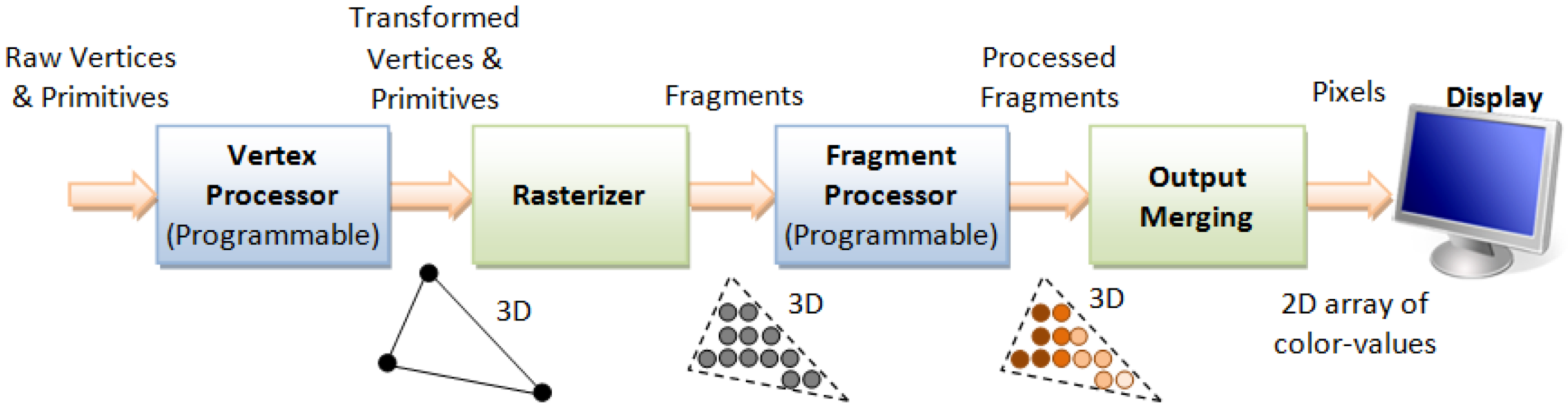
2018 - Nvidia introduces consumer GPUs that support real-time ray tracing

Difference between Rasterization and Ray Tracing

Rasterization:

The ubiquitous rendering pipeline that is implemented by all graphics frameworks and game engines. For every object in the 3D world space that is visible to the camera, the object is projected onto the 2D image plane (using a vertex shader) and then a fragment shader decides what color to "shade" the pixel (based on information about the world and the object that's been passed into the shader – e.g., lights, texture maps, etc).

That is, processing starts from the *object*, whereas in ray tracing, processing starts from a *ray*.


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
3D
Fragments
Fragment Processor (Programmable)
3D
Processed Fragments
Output Merging
3D
Pixels
Display
2D array of color-values

3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

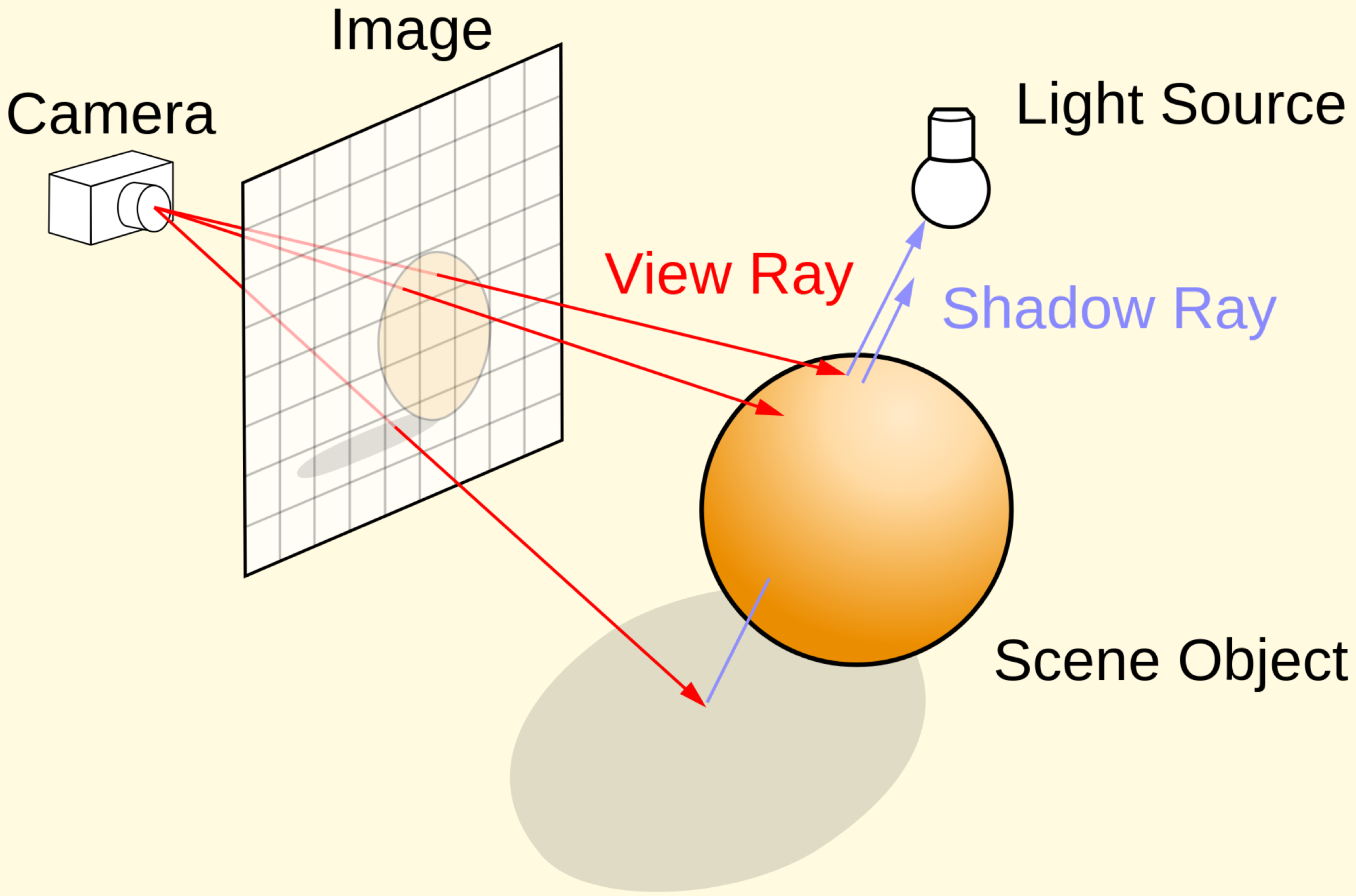


Image
Camera
Light Source
View Ray
Shadow Ray
Scene Object

Definitions

Ray Casting:

Finding the closest object along a ray, e.g., from the camera into a scene to see what object is visible to the viewer, or from an object toward a light source, to check if that object is in shadow.

Ray Marching:

Finds the intersection with an implicitly defined object along a ray, and then "marches" along the ray, sampling the values inside the object to render a volume.

Definitions

Ray Tracing:

Uses ray casting to recursively spawn and then gather contributions of light from reflective and refractive objects. The rays are evaluated to determine the final color of the pixel on the image plane that cast the first ray.

“Classic” Ray Tracing:

A single ray is cast for each pixel in the image plane, all surfaces are perfectly shiny or smooth, and all lights are point lights.

Stochastic Ray Tracing:

Multiple rays are spawned at each bounce, e.g., to model area lights which produce soft shadows, and to model more complex materials.

Definitions

Path Tracing:

Provides a more physically accurate representation of light, which can bounce off of surfaces as well as emanate directly from light sources, and models materials uses a BRDF to describe the probability of a photon to bounce in a particular direction.

BRDF:

The Bidirectional reflection distribution function takes an incoming light direction and an outgoing direction, and returns the amount of reflected radiance exiting along the outgoing direction, relative to the irradiance from the incoming light direction.

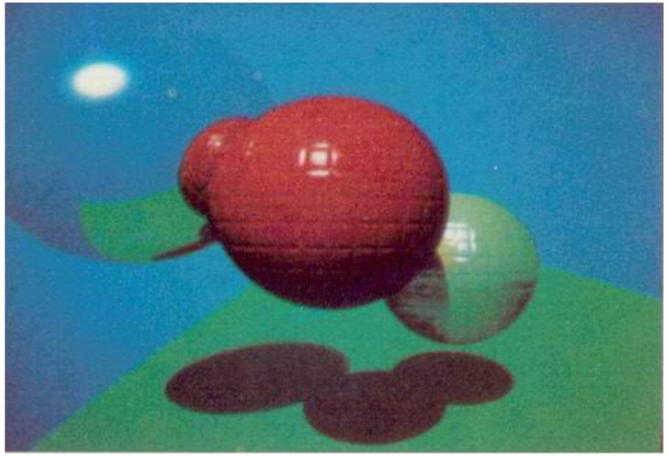


Figure 5

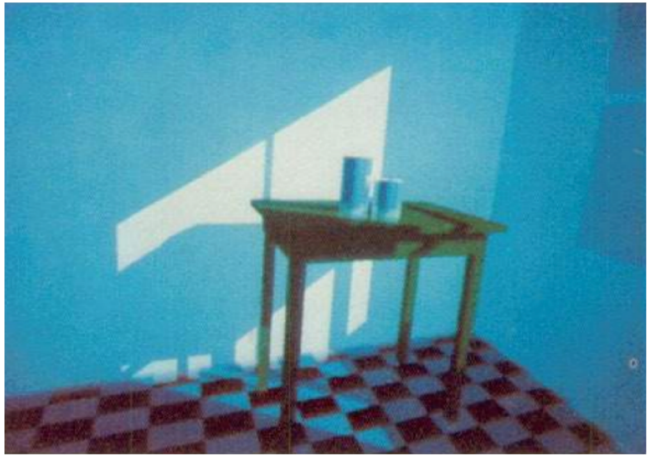


Figure 7

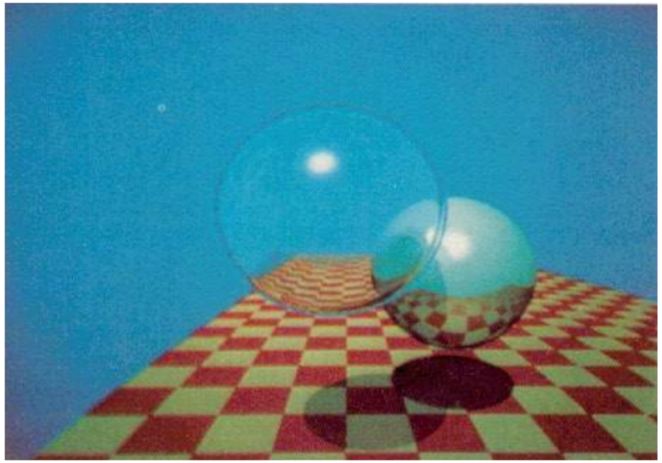


Figure 6

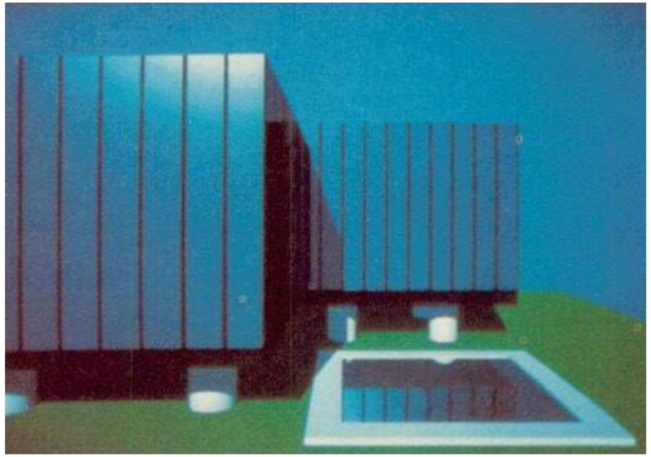
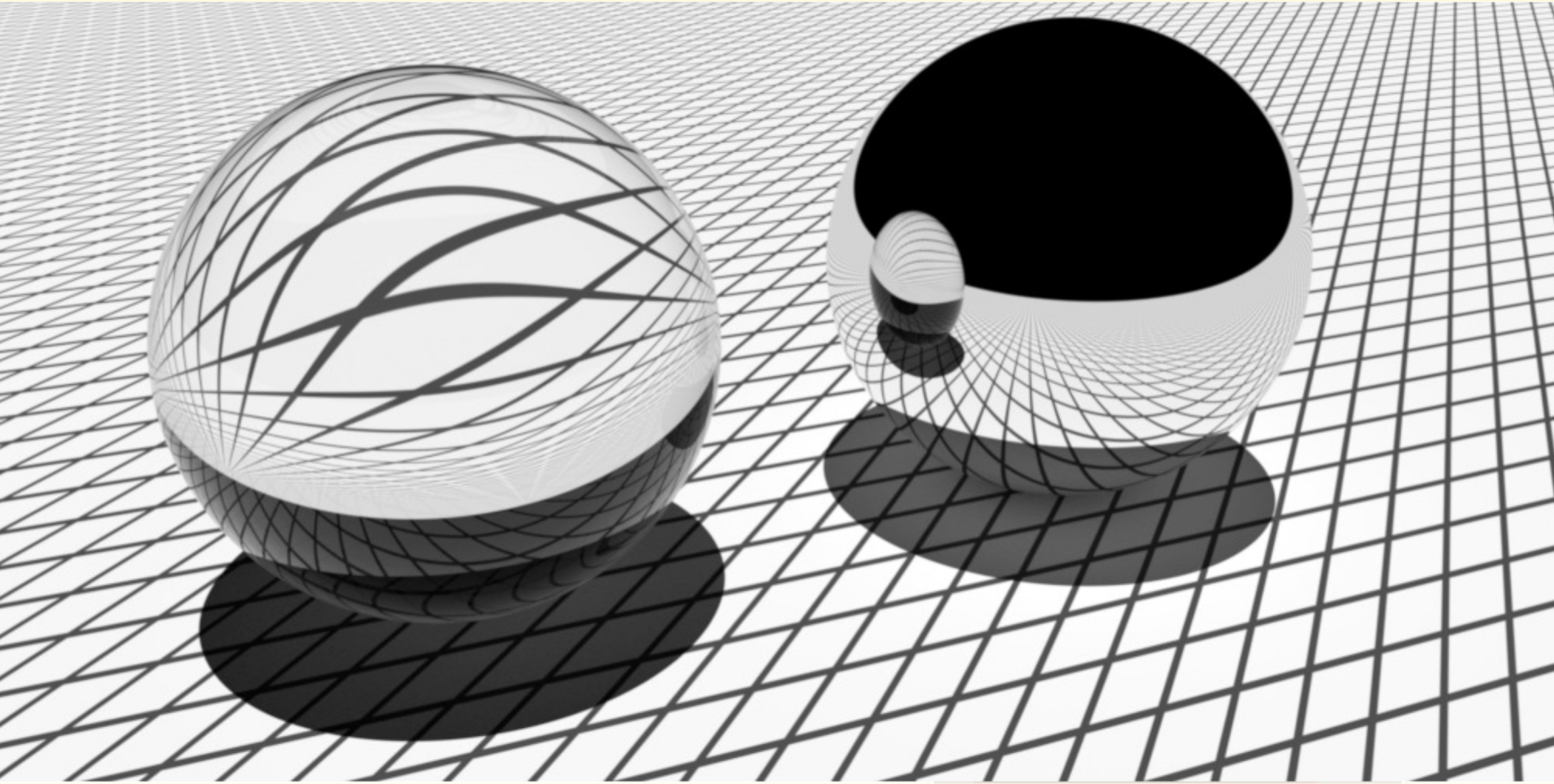
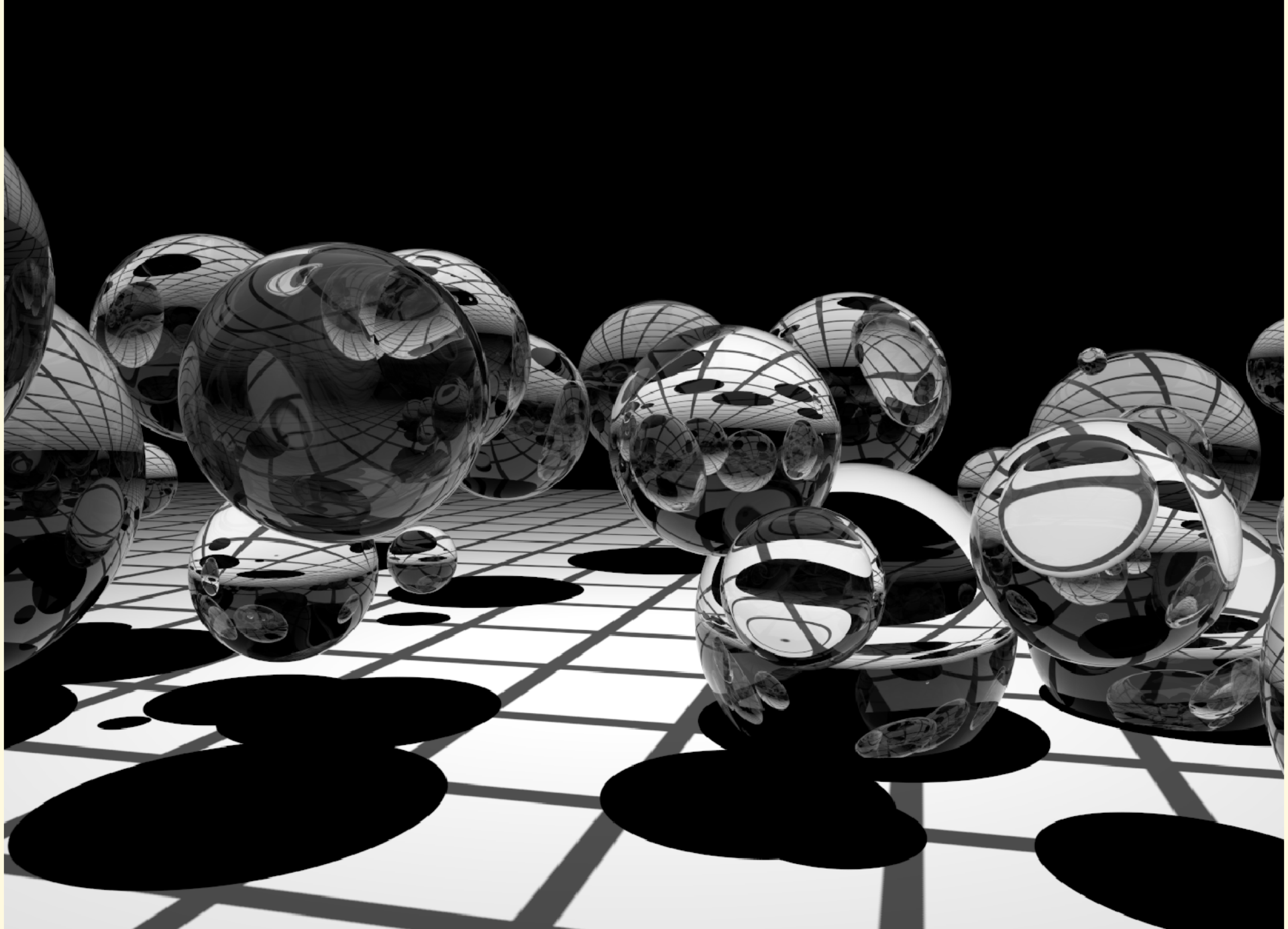


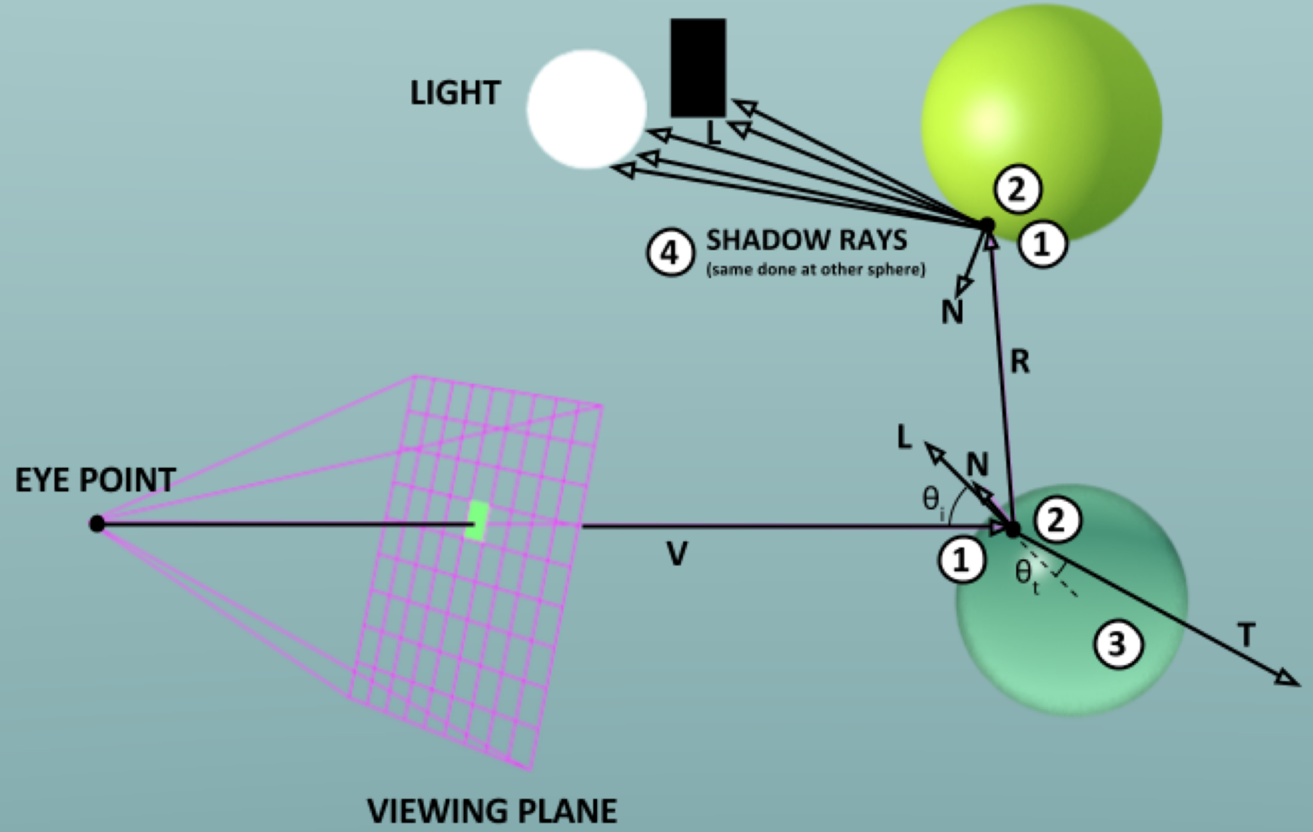
Figure 8





RAY TRACING

(for one pixel up to first bounce)



1

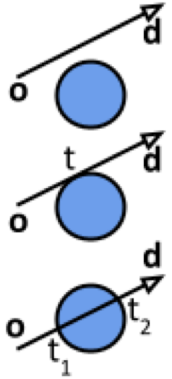
Sphere equation: $(\vec{p}-\vec{c})\cdot(\vec{p}-\vec{c}) = r^2$

Ray equation: $\vec{r}(t) = \vec{o} + t\vec{d}$

Intersection:

$$\left(\vec{o}+t\vec{d}-\vec{c}\right)\cdot\left(\vec{o}+t\vec{d}-\vec{c}\right) = r^2$$

$$t^2\left(\vec{d}\cdot\vec{d}\right) + 2\left(\vec{o}-\vec{c}\right)t\vec{d} + \left(\vec{o}-\vec{c}\right)\cdot\left(\vec{o}-\vec{c}\right) - r^2 = 0$$



2

Illuminiation Equation (Blinn–Phong) with recursive Transmitted and Reflected Intensity:

$$I = k_a I_a + I_i\left(k_d\left(\vec{L}\cdot\vec{N}\right) + k_s\left(\vec{V}\cdot\vec{R}\right)^n\right) + \underbrace{k_t I_t + k_r I_r}_{recursion}$$

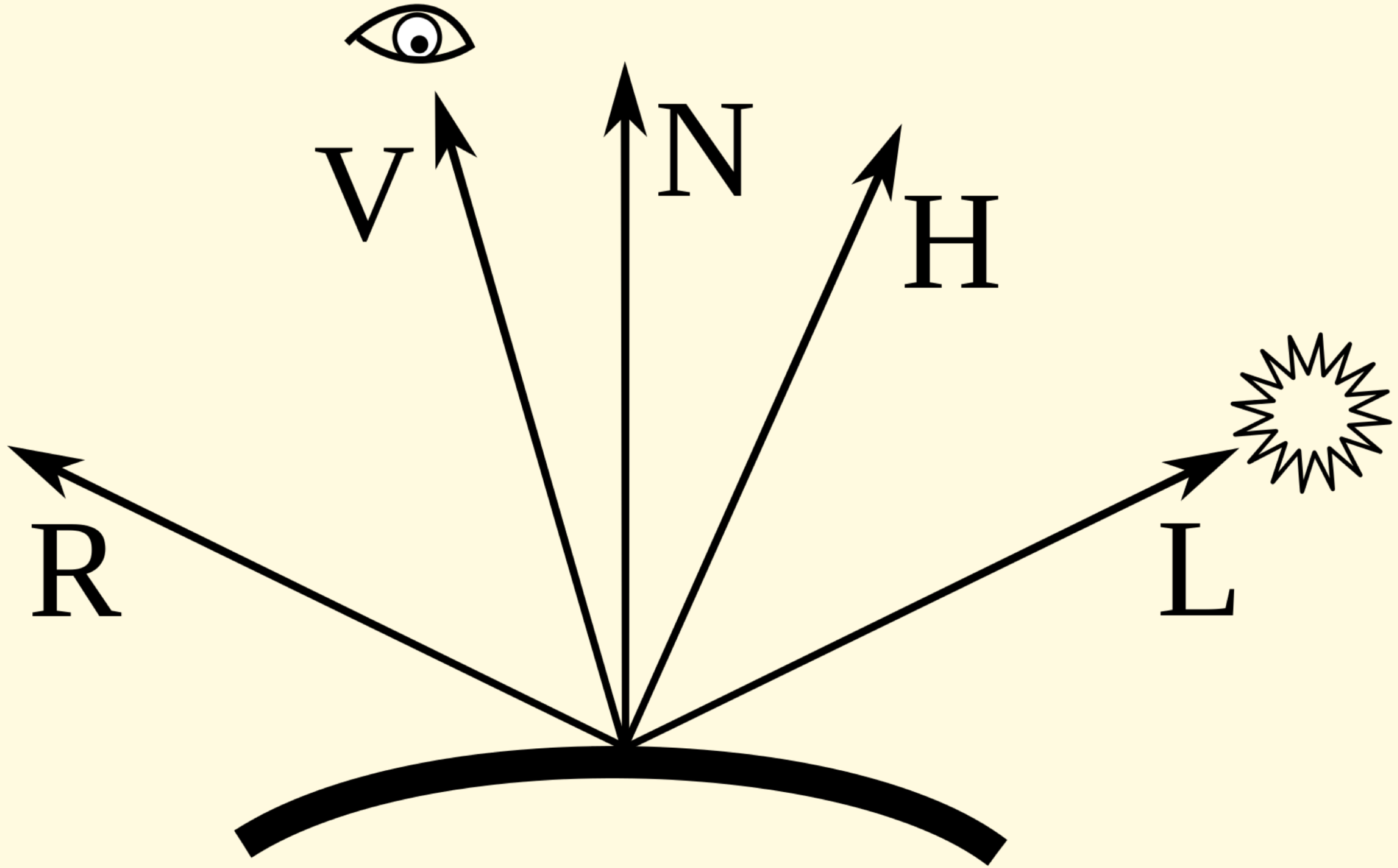
3

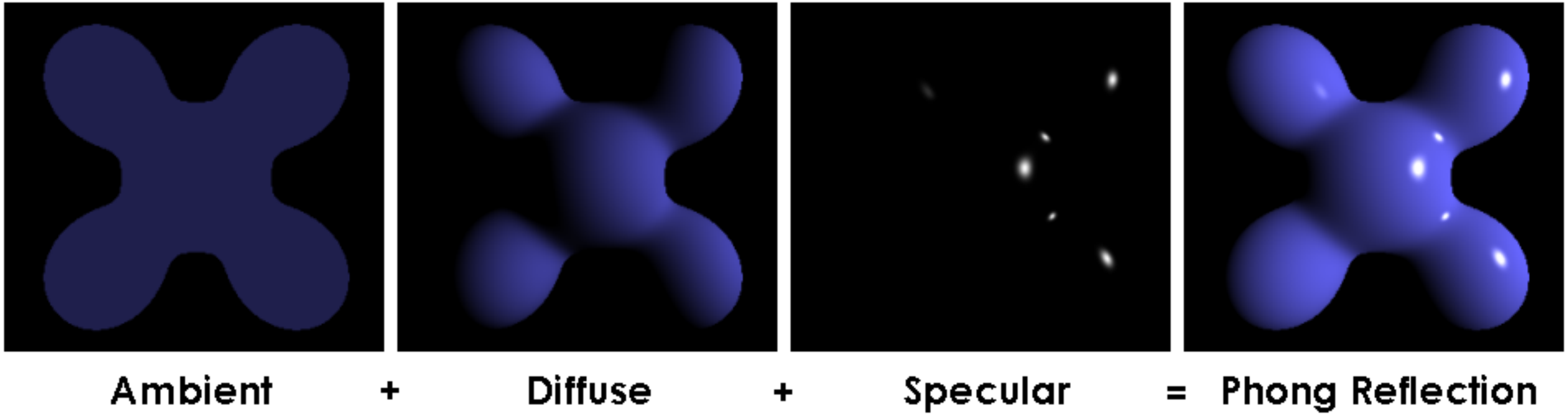
Snell's law: $\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}$ $\quad n_{air}\sin\theta_i = n_{glass}\sin\theta_t$

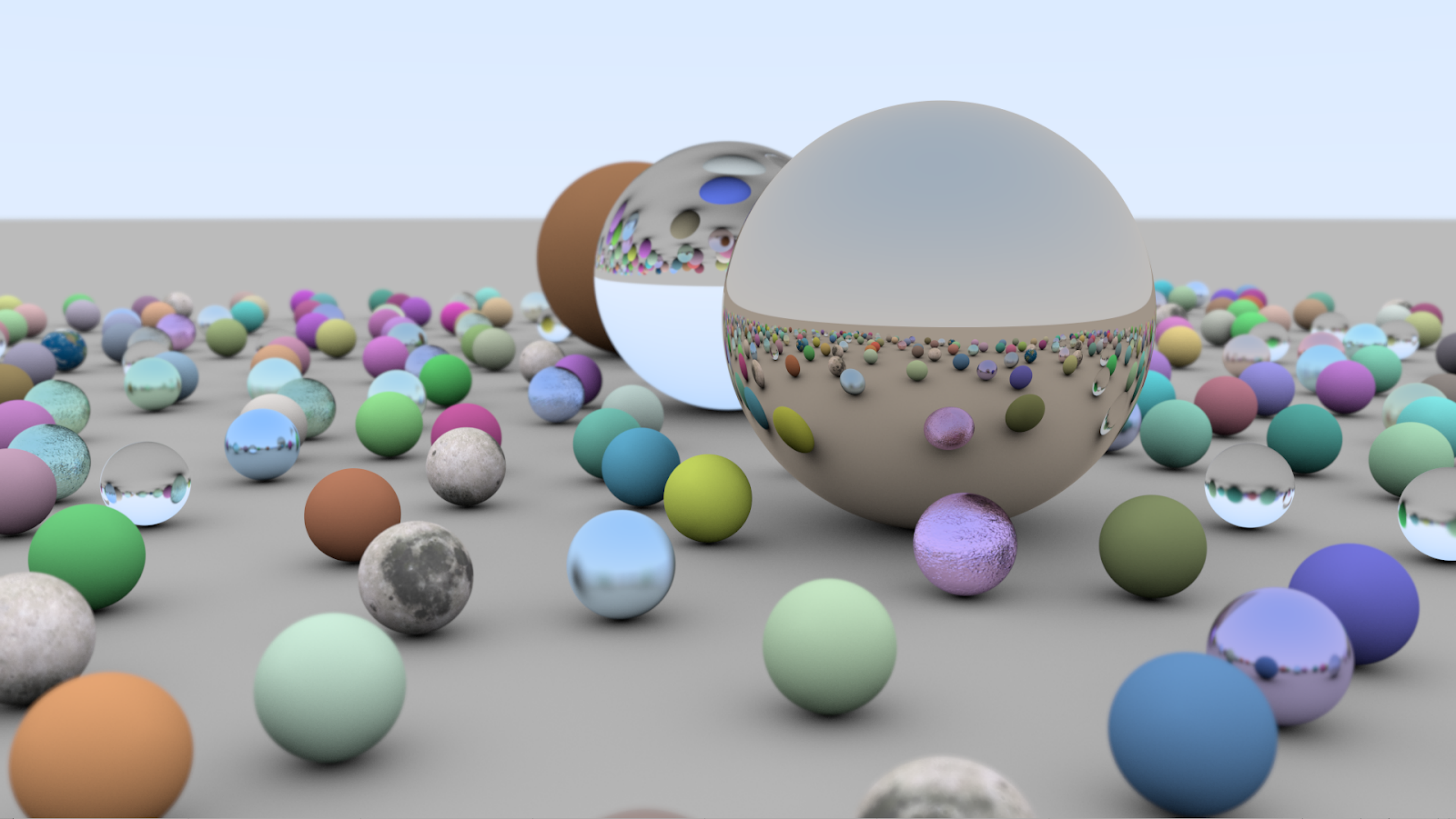
refraction coefficients:
$n_{air} = 1$, $n_{glass} = 1.5$

4

Area Light Simulation: $I_{light}\frac{\#\,(\text{visible shadow rays})}{\#\,(\text{all shadow rays})}$


V
N
H
R
L


Ambient
+
Diffuse
+
Specular
=
Phong Reflection



Homework 2

1: Create a simple scene using Unreal Engine (due 10/6)

2: Implement a classic Whitted recursive ray tracer (due 10/15)

Homework 2

- 1. Calculate Rays from Camera through each pixel of the Image Plane into 3D Scene
- 2. Calculate intersection point with closest 3D object
- 3. Depending on the object's material:
    - Calculate color at point by casting a ray towards each light to determine diffuse and specular contributions
    - Cast a reflection ray, go to step 2
    - Cast a refraction ray, go to step 2
- 4. Combine information from recursively cast rays, check if object is in shadow, return final color

Homework 2

- 3. Depending on the object’s material:
    - Calculate color at point by casting a ray towards each light
    - Cast a reflection ray, go to step 2
    - Cast a refraction ray, go to step 2

Next week, we’ll go over how to calculate the diffuse + specular components & how to calculate the reflection and refraction rays

Homework 2

PPM files:

```
P3
3 2
255
# The part above is the header
# "P3" means this is a RGB color image in ASCII
# "3 2" is the width and height of the image in pixels
# "255" is the maximum value for each color
# The part below is image data: RGB triplets
255   0   0     0 255   0     0   0 255
255 255   0   255 255 255     0   0   0
```

You'll see a lot of examples that use "P6" - an RGB color image in bytes
"If the PPM magic identifier is "P6" then the image data is stored in byte format, one byte per colour component (r,g,b)" - Paul Bourke