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)

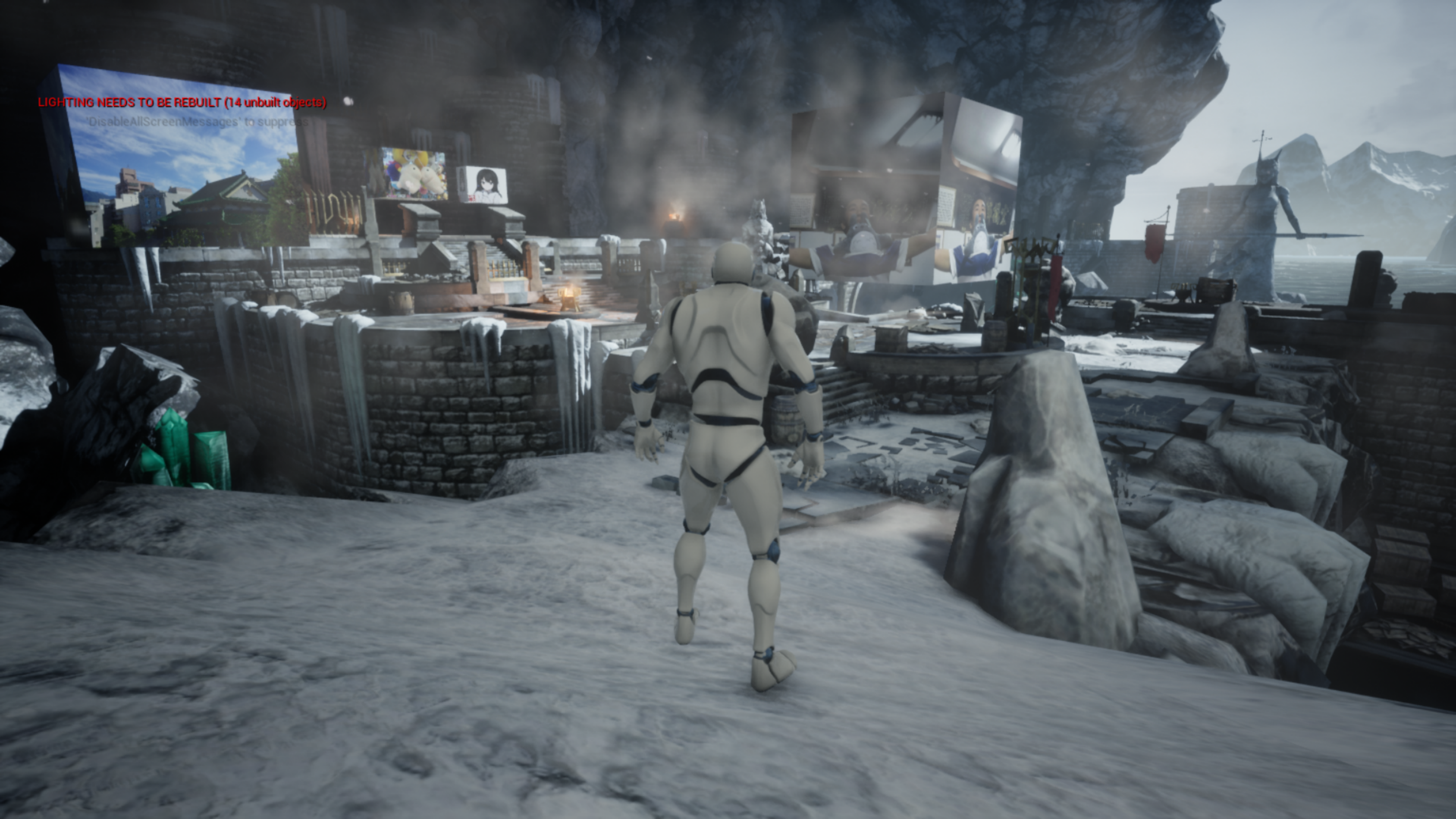
Lab website / Direction:

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

Homework 2

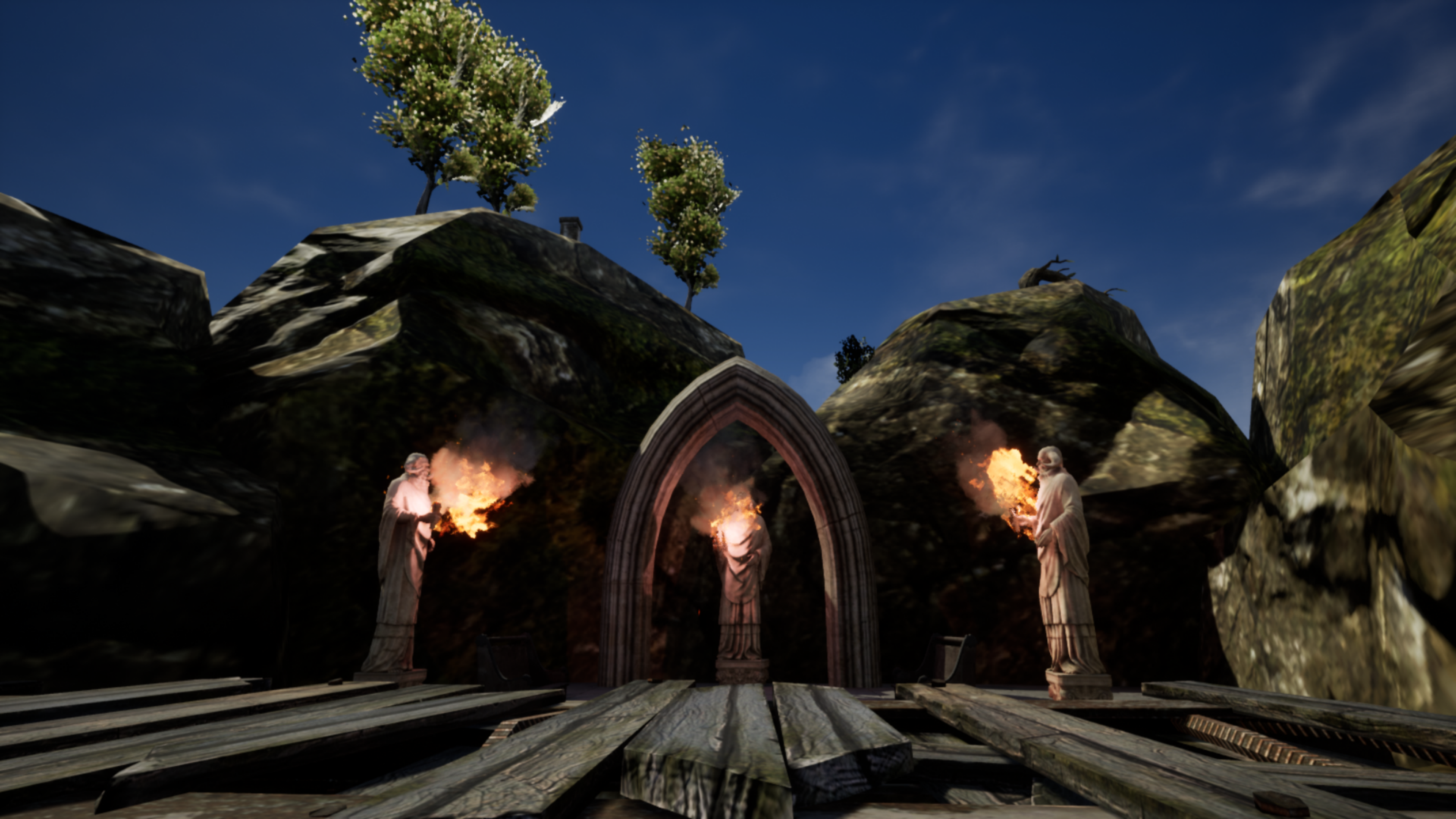
1: Create a simple scene using Unreal Engine (due 10/6)

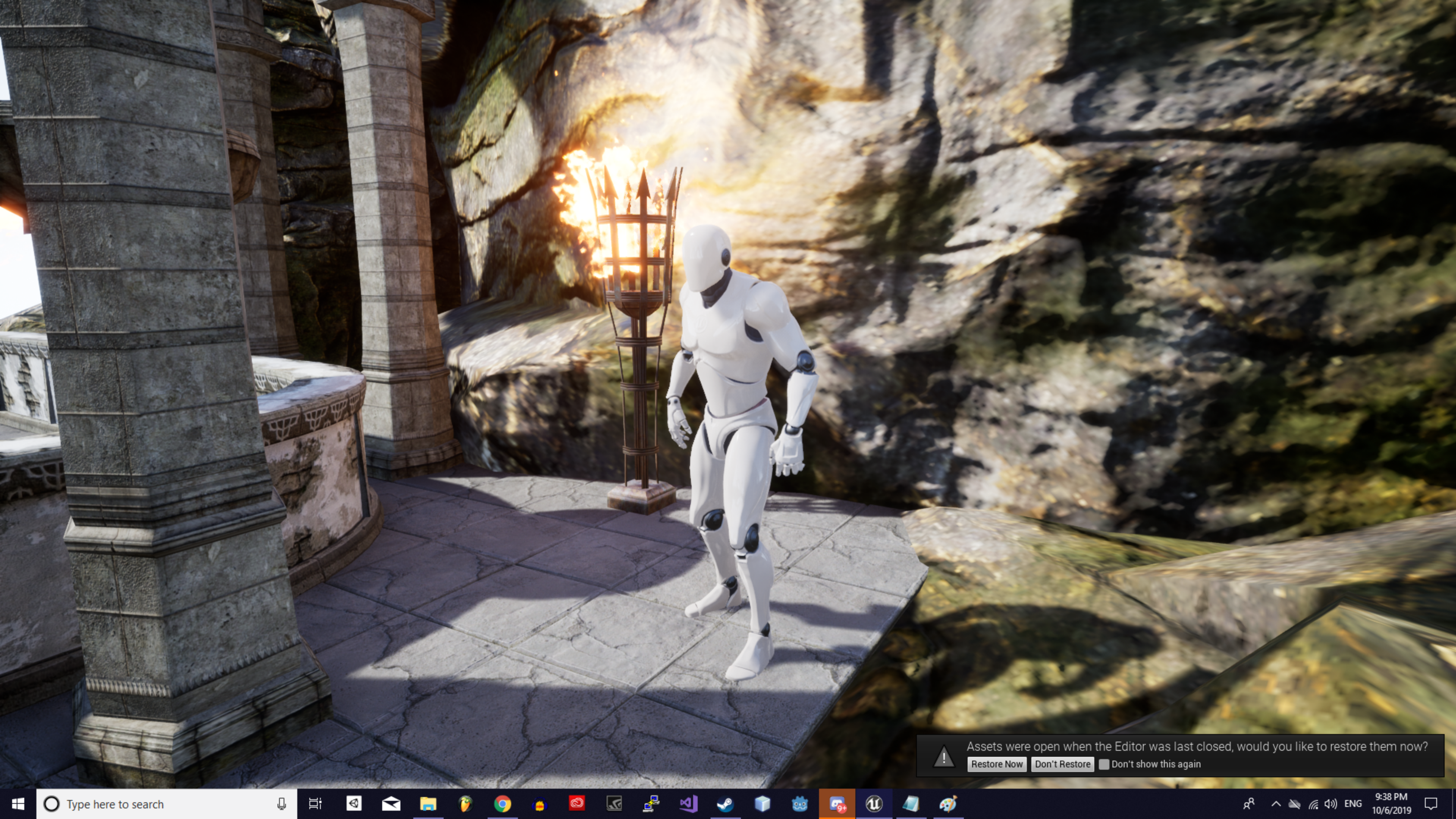
2: Implement a classic Whitted recursive ray tracer (due 10/15)



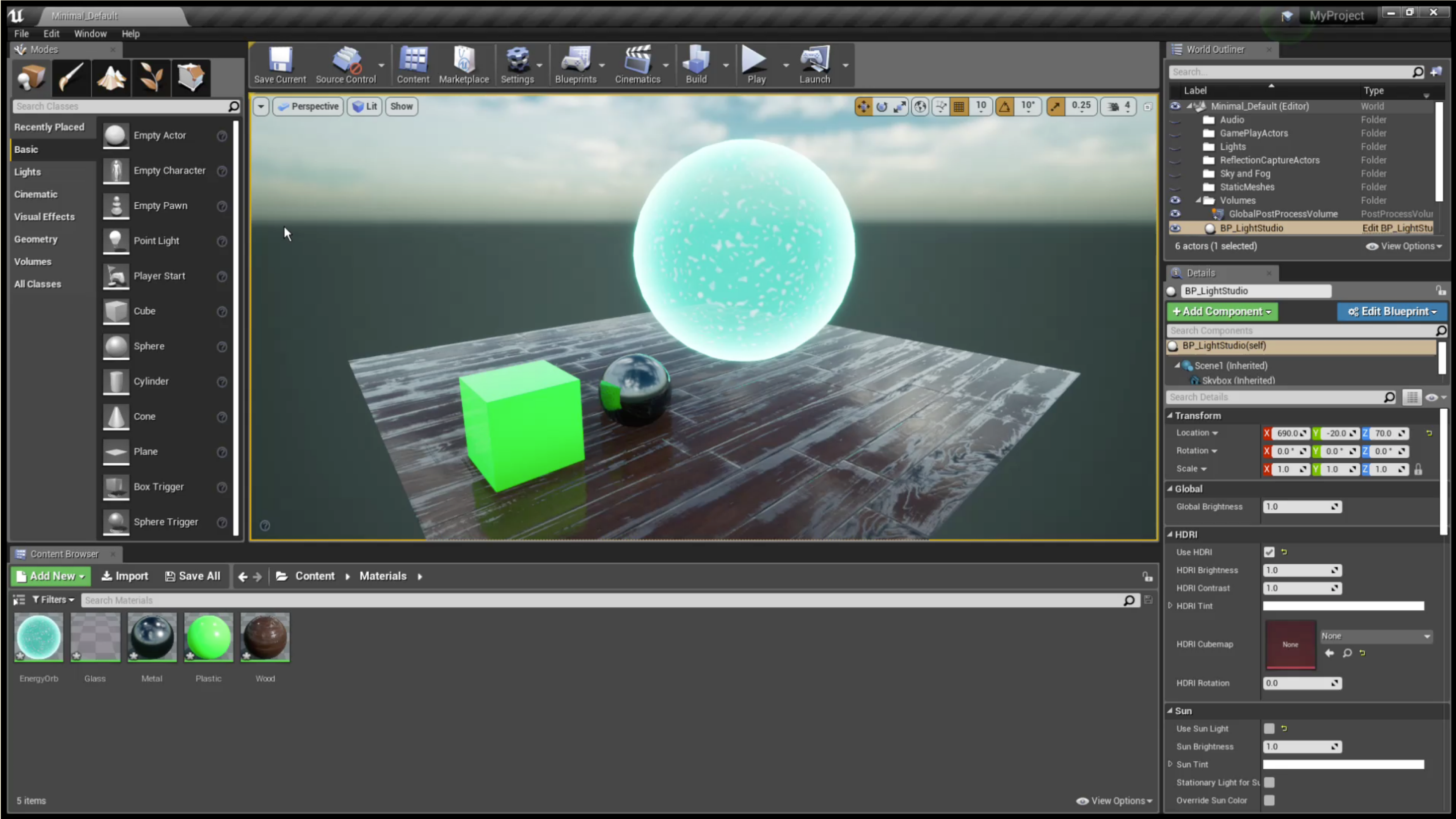
LIGHTING NEEDS TO BE REBUILT (14 unbuilt objects)

'DisableAllScreenMessages' to suppress





Assets were open when the Editor was last closed, would you like to restore them now?



Label	Type
Minimal_Default (Editor)	World
Audio	Folder
GamePlayActors	Folder
Lights	Folder
ReflectionCaptureActors	Folder
Sky and Fog	Folder
StaticMeshes	Folder
Volumes	Folder
GlobalPostProcessVolume	PostProcessVolu
BP_LightStudio	Edit BP_LightStu

6 actors (1 selected)

BP_LightStudio

BP_LightStudio(self)
Scene1 (Inherited)
Skybox (Inherited)

Transform

	X	Y	Z
Location	690.0	-20.0	70.0
Rotation	0.0°	0.0°	0.0°
Scale	1.0	1.0	1.0

Global

Global Brightness 1.0

HDRI

Use HDRI ☑

HDRI Brightness 1.0

HDRI Contrast 1.0

HDRI Tint

HDRI Cubemap None

HDRI Rotation 0.0

Sun

Use Sun Light □

Sun Brightness 1.0

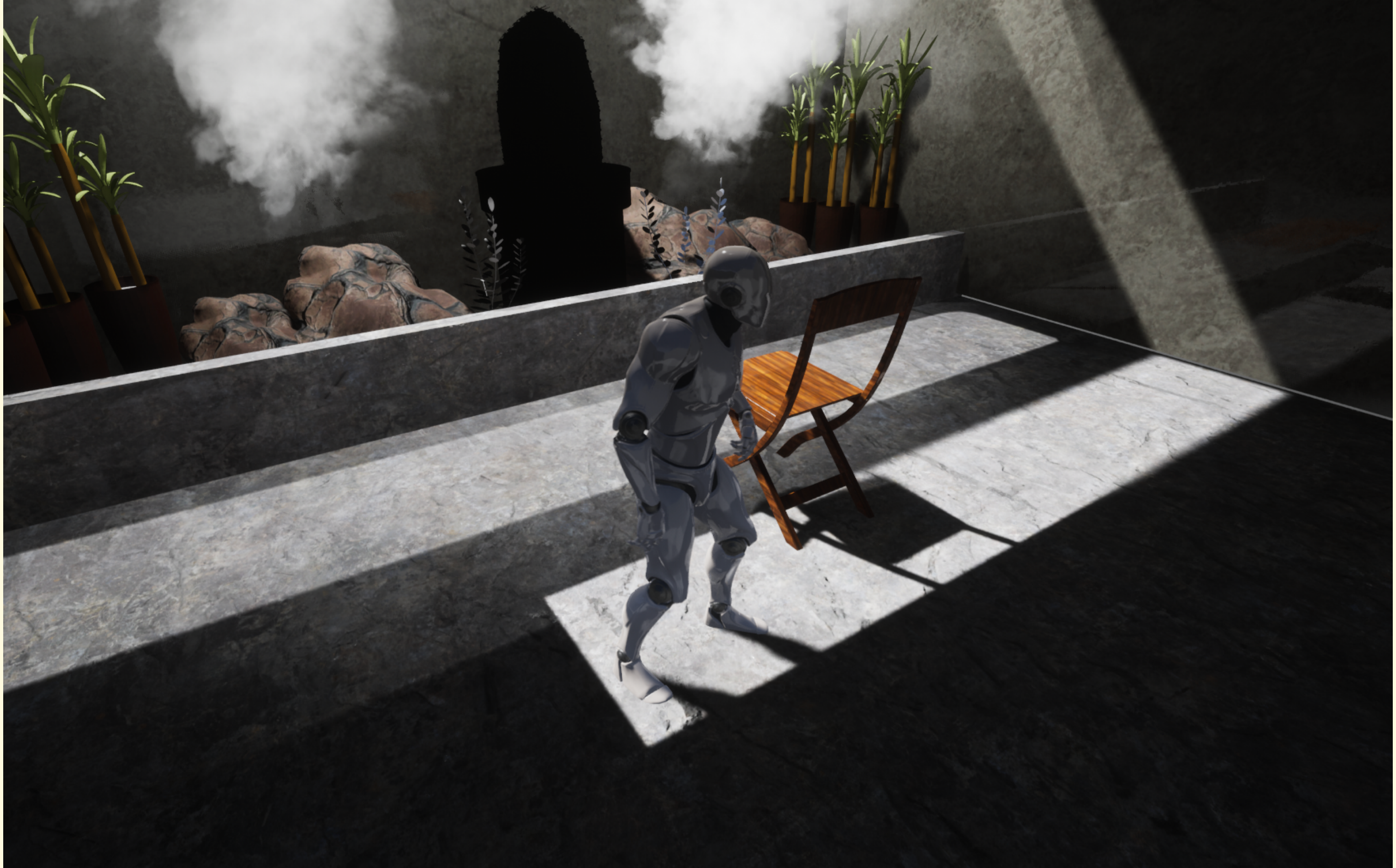
Sun Tint

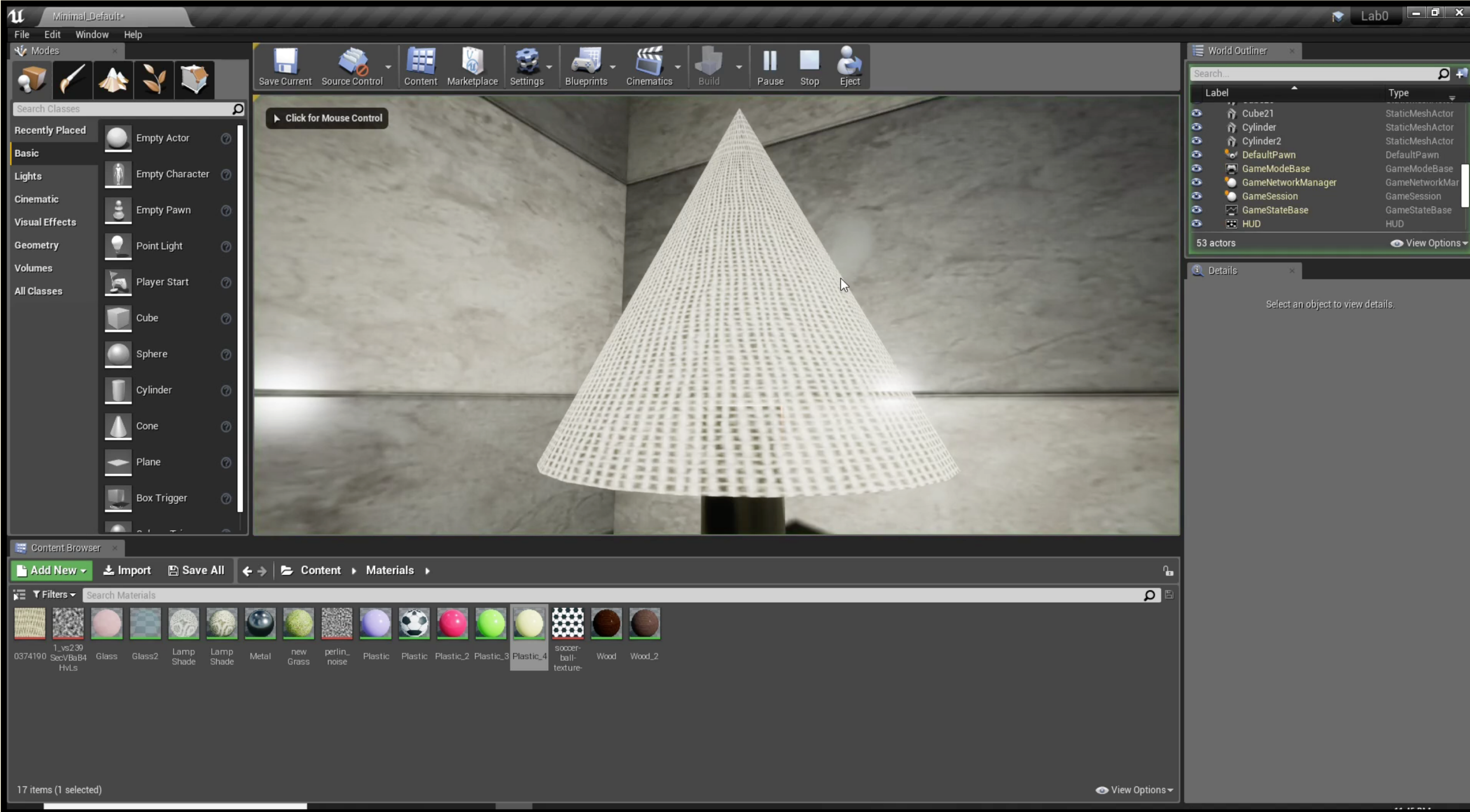
Stationary Light for Su □

Override Sun Color □

EnergyOrb Glass Metal Plastic Wood

5 items



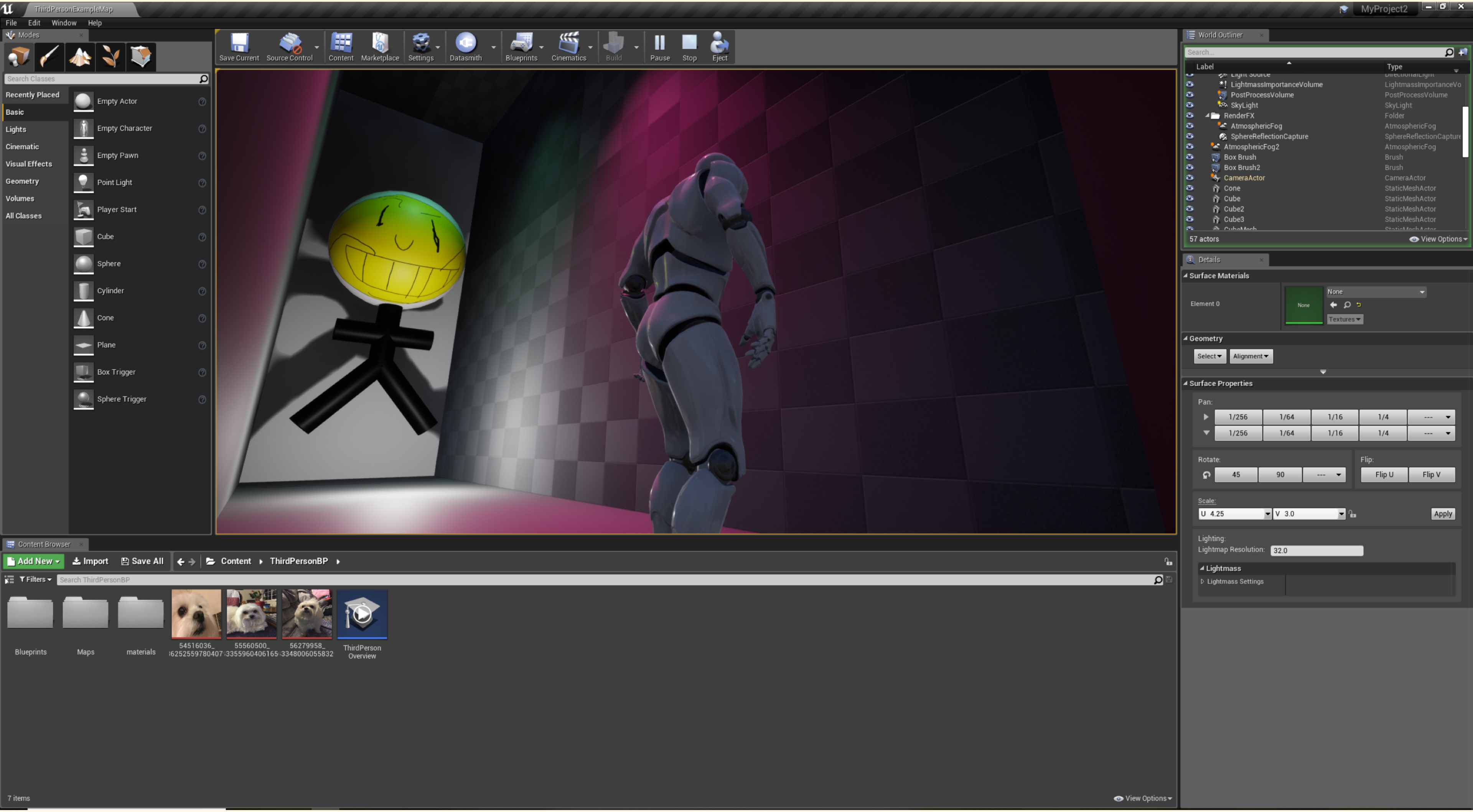


Label	Type
Cube21	StaticMeshActor
Cylinder	StaticMeshActor
Cylinder2	StaticMeshActor
DefaultPawn	DefaultPawn
GameModeBase	GameModeBase
GameNetworkManager	GameNetworkMar
GameSession	GameSession
GameStateBase	GameStateBase
HUD	HUD

53 actors

Select an object to view details.

17 items (1 selected)



Label	Type
Light Source	DirectionalLight
LightmassImportanceVolume	LightmassImportanceVo
PostProcessVolume	PostProcessVolume
SkyLight	SkyLight
RenderFX	Folder
AtmosphericFog	AtmosphericFog
SphereReflectionCapture	SphereReflectionCapture
AtmosphericFog2	AtmosphericFog
Box Brush	Brush
Box Brush2	Brush
CameraActor	CameraActor
Cone	StaticMeshActor
Cube	StaticMeshActor
Cube2	StaticMeshActor
Cube3	StaticMeshActor
CubeMesh	StaticMeshActor

57 actors

Surface Materials

Element 0: None

Textures

Geometry

Select

Alignment

Surface Properties

Pan: 1/256 1/64 1/16 1/4 ---

1/256 1/64 1/16 1/4 ---

Rotate: 45 90 ---

Flip: Flip U Flip V

Scale: U 4.25 V 3.0 Apply

Lighting:

Lightmap Resolution: 32.0

Lightmass

Lightmass Settings

Blueprints

Maps

materials

54516036_ 6252559780407

55560500_ 3355960406165

56279958_ 3348006055832

ThirdPerson Overview

7 items

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

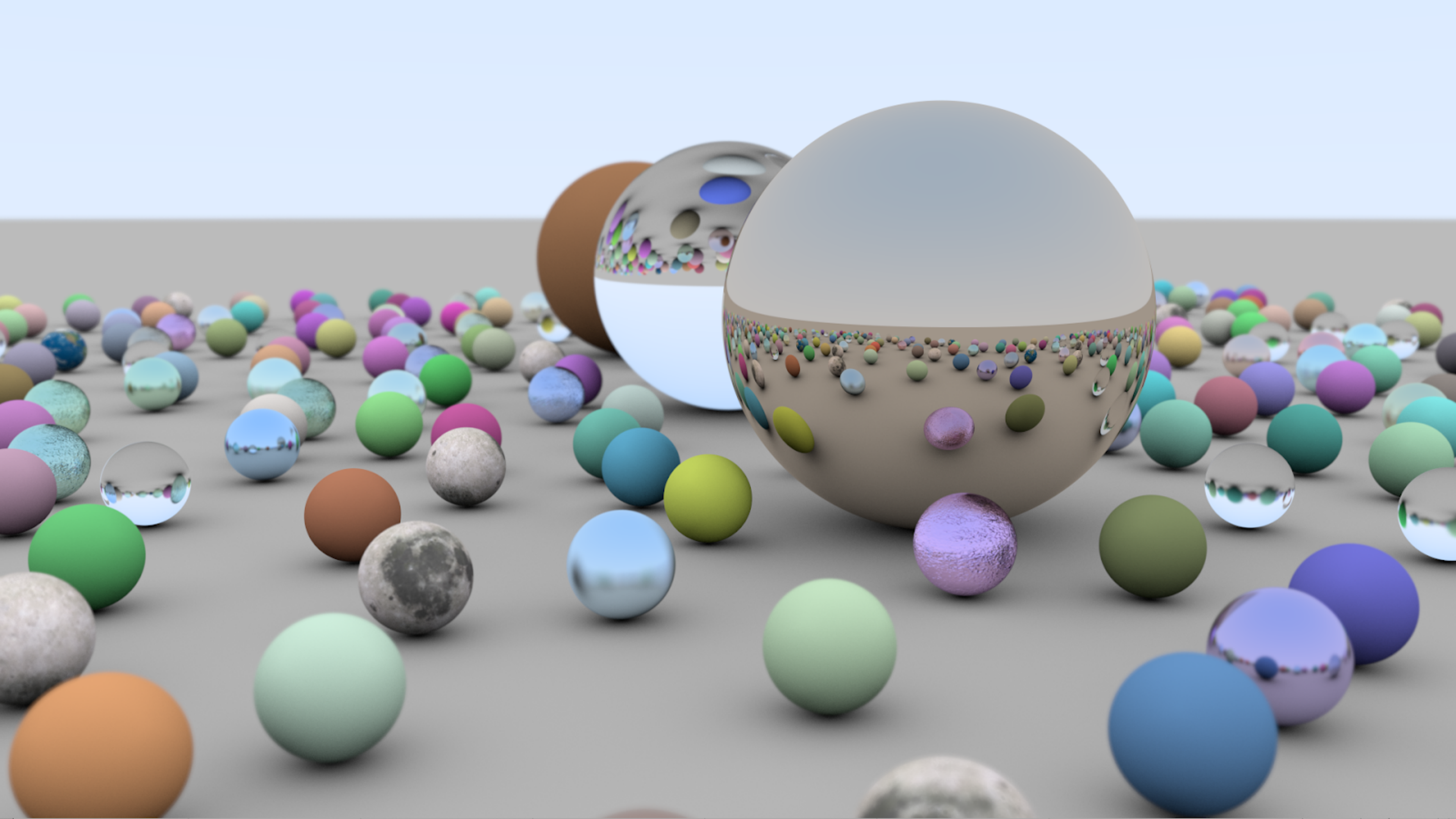
Homework 2

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
```

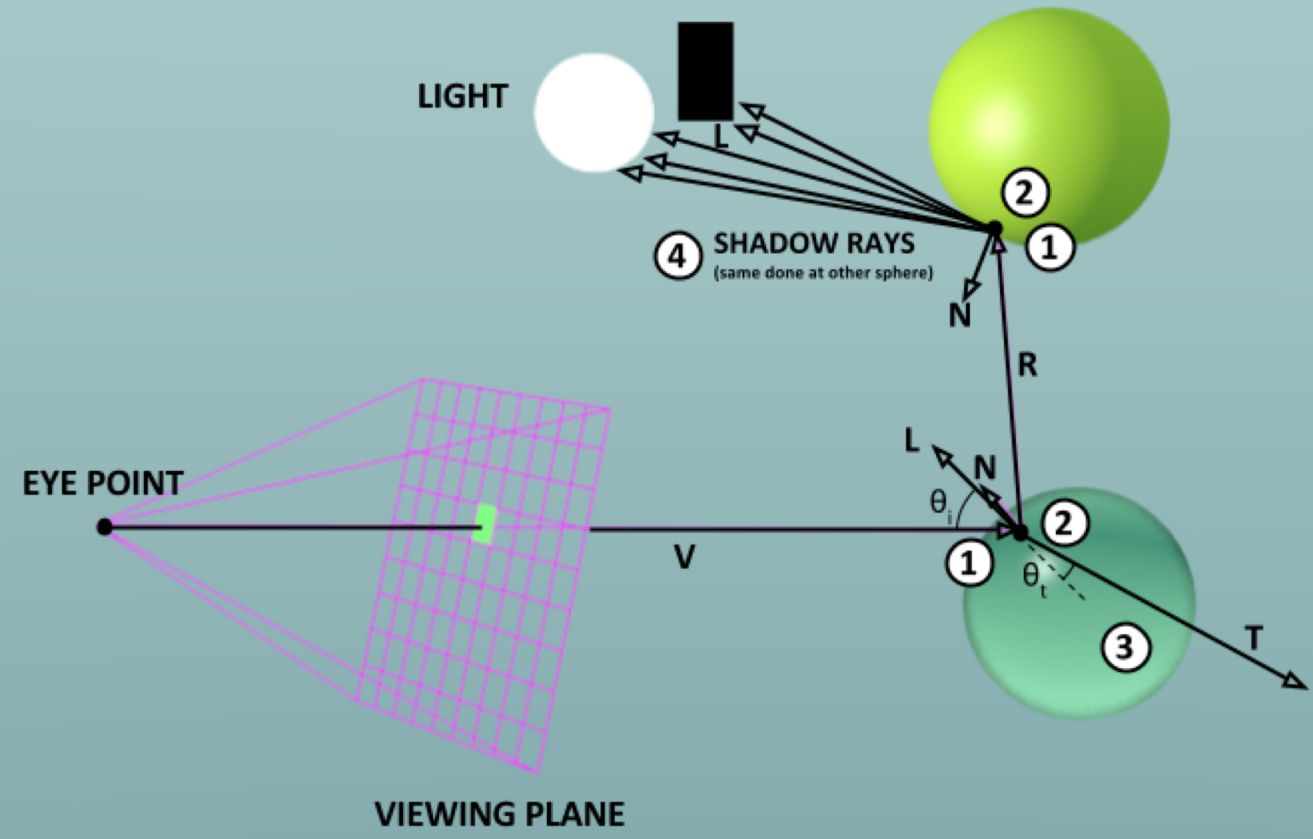
You'll see a lot of examples that use "P6" - an RGB color image in bytes
"If the PPM magic identifier is "P6" then the image data is stored in byte format, one byte per colour component (r,g,b)" - Paul Bourke

See https://www.scratchapixel.com/code.php?id=3 for an example C++ code to write out to a PPM file



RAY TRACING

(for one pixel up to first bounce)



①

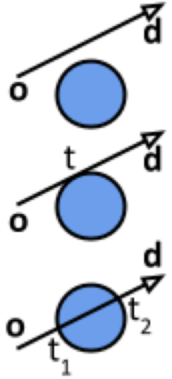
Sphere equation: $(\vec{p}-\vec{c})\cdot(\vec{p}-\vec{c})=r^2$

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:

$$I=k_aI_a+I_i\left(k_d\left(\vec{L}\cdot\vec{N}\right)+k_s\left(\vec{V}\cdot\vec{R}\right)^n\right)+\underbrace{k_tI_t+k_rI_r}_{recursion}$$

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

Pixels to World Space

Given an eye position and an image plane, how do you calculate the ray from the eye through each pixel?

Another way of saying this is: How do you move between raster space or pixel space to world space?

(in class explanation on chalkboard…)

Once you have the pixels in World Space, you can create a ray from the Camera through the pixel into the 3D scene.

Calculating Intersections

- Ray-Sphere intersection

- Ray-Triangle intersection

(in class explanation on chalkboard…)

See:

https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection

https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/why-are-triangles-useful

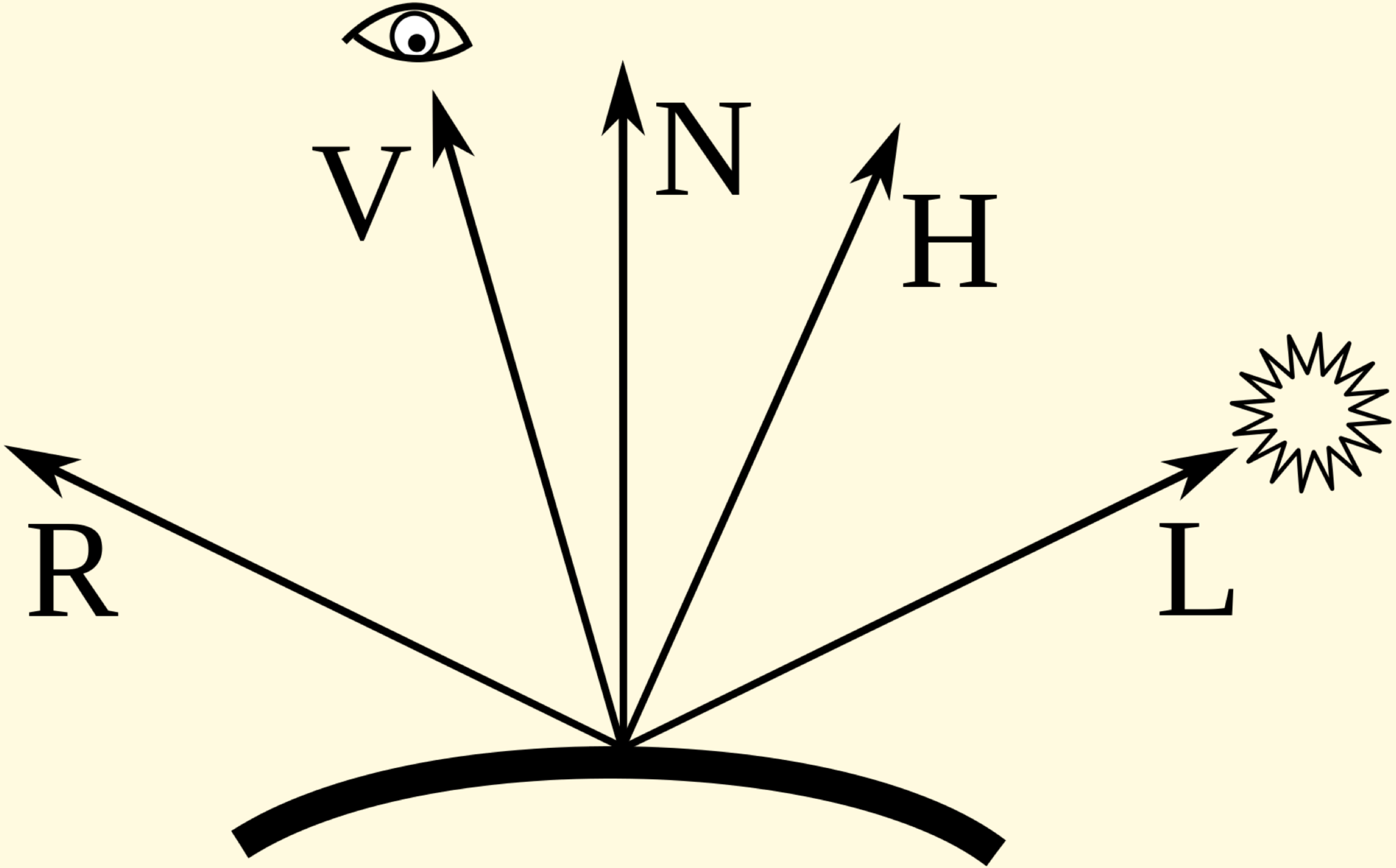
https://www.iquilezles.org/www/articles/intersectors/intersectors.htm

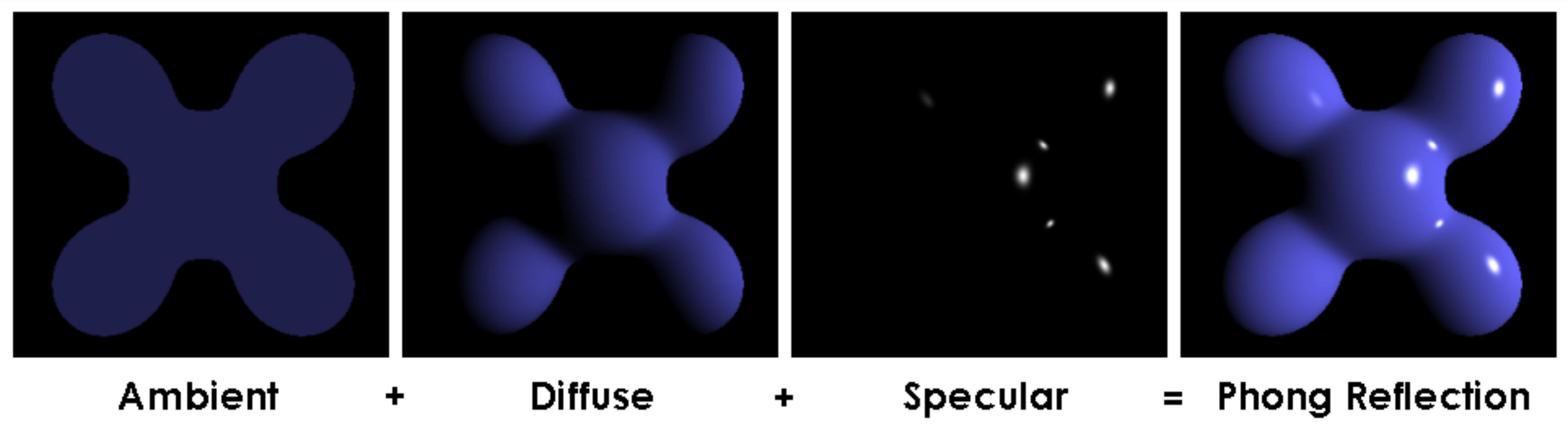
Calculating color at intersections

Once we've detected an intersection point, how do we figure out what color that point is?

Phong shading

A computationally inexpensive approximation of light that includes both diffuse and specular components


V
N
H
R
L


Ambient
+
Diffuse
+
Specular
=
Phong Reflection

Calculating color at intersections

Diffuse calculation:

Requires *Normal* vector and *Light* vector

```
DiffuseContribution =
    diffuseColor * max( (Normal . Light), 0.0 )
```

The diffuse term is largest when the normal and the light are parallel. The diffuse term is 0 when the normal and the light are >= perpendicular.

Calculating color at intersections

Specular calculation:

Requires *Reflect* vector and *View* vector. That is, it can change based on the position of the camera

SpecularContribution =
    specularColor * (Reflect . View)^shininessFactor

The specular term is large only when the viewer direction is aligned with the reflection direction. The highlights are sharper the greater the shininessFactor is.

(*Reflect* vector is calculated using *Light* vector and *Normal* vector)

Calculating color at intersections

Alternative Specular calculation:

Requires *Normal* vector and a *Half* vector that is in between the *View* and the *Light* vectors.

SpecularContribution =
    specularColor * (Normal . Half)^shininessFactor

The specular term is large only when the viewer direction is aligned with the reflection direction. The highlights are sharper the greater the shininessFactor is.

Calculating color at intersections

Calculate diffuse and specular contribution for each light and add them together

Check if light is not occluded by another object, otherwise move to next to light. If all lights are occluded, then the point is completely in shadow.

Next class

Reflection, Refraction, Recursion …