INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 12: “Post-processing”

Welcome!

Today's Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

```
rics
& (depth < MAXD

t = inside / 1
nt = nt / nc, ddn
os2t = 1.0f - nnt
D, N );
0)

at a = nt - nc, b = nt
at Tr = 1 - (R0 + (1 - R0)
Tr) R = (D * nnt - N * (ddn

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH

D, N );
refl * E * diffuse;
= true;


MAXDEPTH)

survive = SurvivalProbability( diffuse
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurv
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (

andom walk - done properly, closely following
vive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2,
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

```
& (depth < MAXDEPTH)

c = inside
nt = nt / nc, ddn
os2t = 1.0f - nnt * nnt
D, N );
0)

at a = nt - nc, b = nt
at Tr = 1 - (R0 + (1 - R0)
Tr) R = (D * nnt - N * (ddn

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)

D, N );
refl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) &&

v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) *

andom walk - done properly, closely following
vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Introduction

Post Processing

Operations carried out on a rendered image.

Purposes:

- Simulation of camera effects
- Simulation of the effects of HDR
- Artistic tweaking of look and feel, separate from actual rendering
- Calculating light transport in space
- Anti-aliasing

Post processing is handled by the *post processing pipeline*.

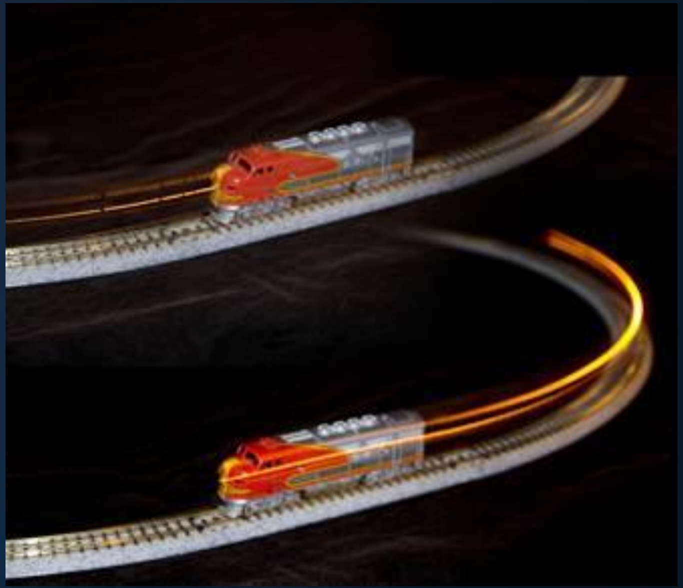
Input: rendered image, in linear color format;
Output: image ready to be displayed on the monitor.

Camera Effects

Purpose: simulating camera / sensor behavior

Bright lights:

- Lens flares
- Glow
- Exposure adjustment
- Trailing / ghosting



Camera Effects

Purpose: simulating camera / sensor behavior

Camera imperfections:

- Vignetting
- Chromatic aberration
- Noise / grain

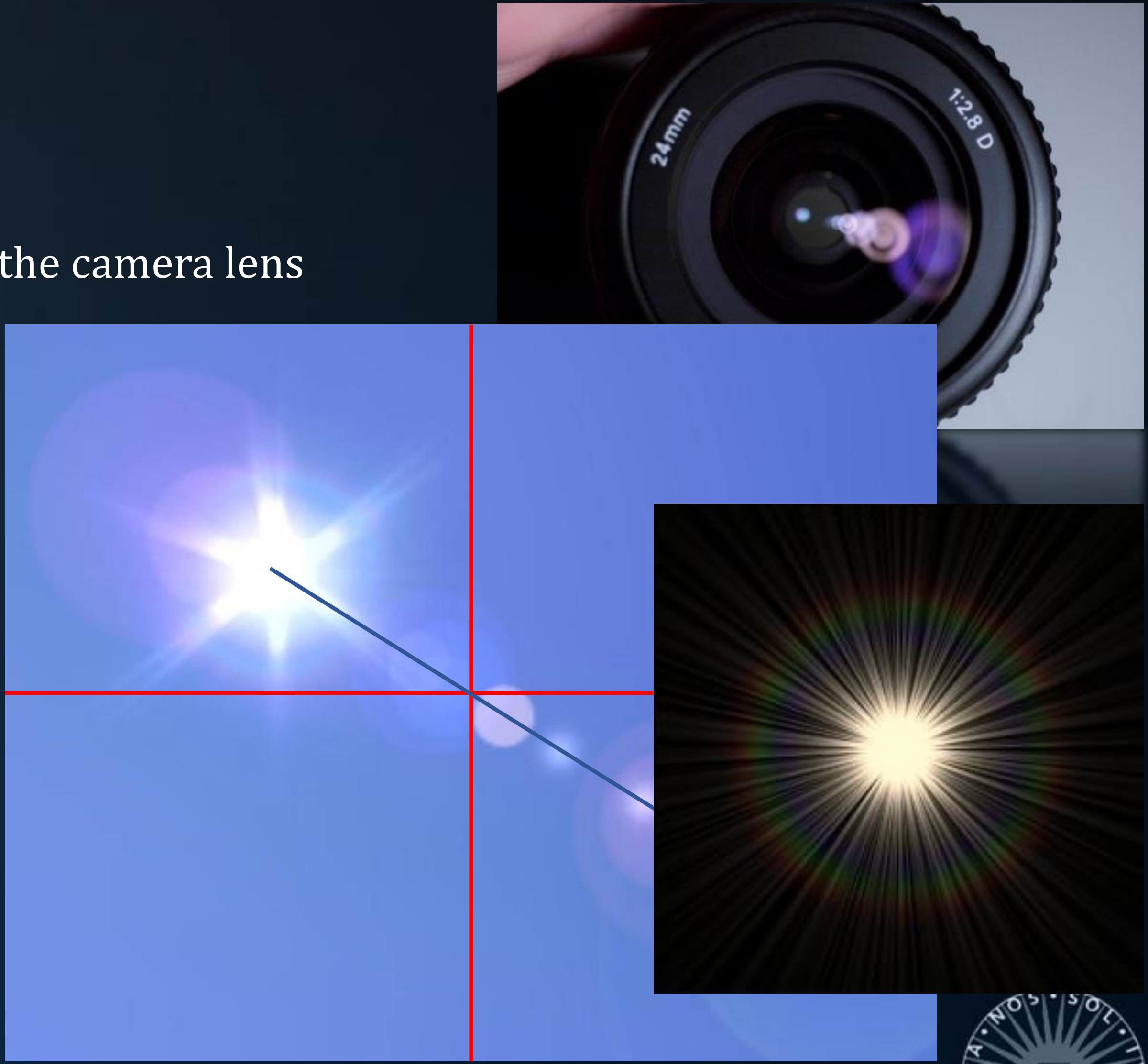
Camera Effects

Lens Flares

Lens flares are the result of reflections in the camera lens system.

Lens flares are typically implemented by drawing sprites, along a line through the center of the screen, with translucency relative to the brightness of the light source.

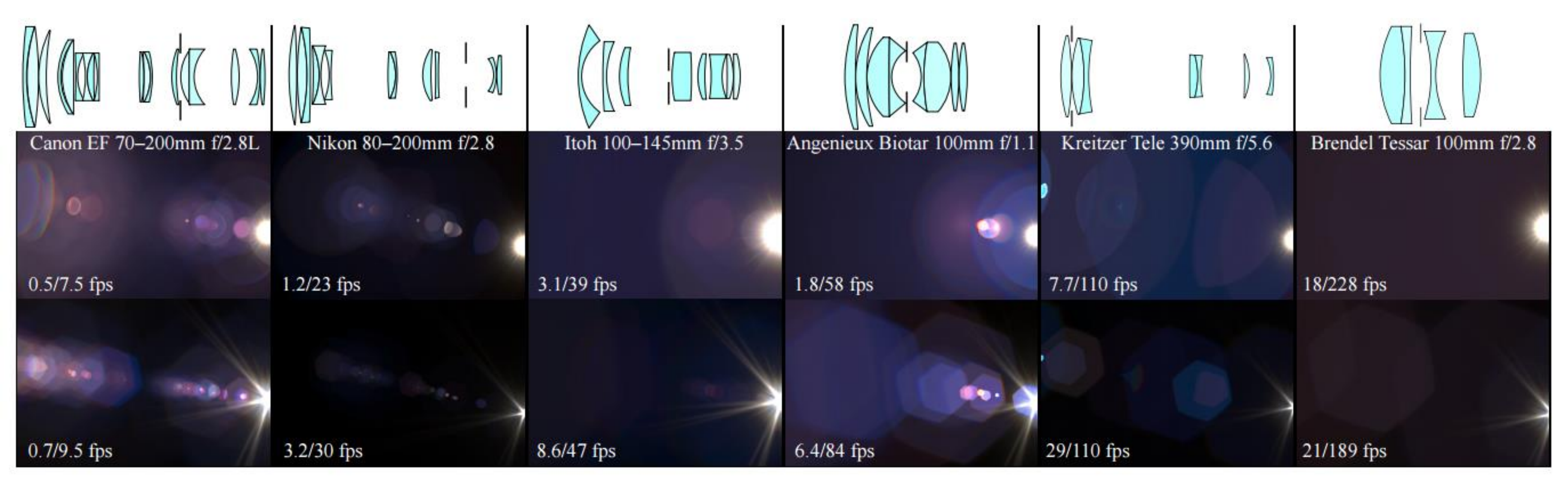
Notice that this type of lens flare is specific to cameras; the human eye has a drastically different response to bright lights.



Camera Effects

Lens Flares

“Physically-Based Real-Time Lens Flare Rendering”, Hullin et al., 2011




Skyrim

Camera Effects

Lens Flares

From: www.alienscribbleinteractive.com/Tutorials/lens_flare_tutorial.html

Camera Effects

Vignetting

Cheap cameras often suffer from vignetting: reduced brightness of the image for pixels further away from the center.


SELF STORAGE

```
& (depth < MAXDE
c = inside / 1
nt = nt / nc, ddn
os2t = 1.0f - nnt * nnt
O, N );
0)

at a = nt - nc, b = nt
at Tr = 1 - (R0 + (1 - R0
Tr) R = (D * nnt - N * (ddn

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH

O, N );
refl * E * diffuse;
= true;

MAXDEPTH)

survive = SurvivalProbability( diffu
estimation - doing it properly, clos
df;
radiance = SampleLight( &rand, I, &L,
e.x + radiance.y + radiance.z) > 0) &

w = true;
at brdfPdf = EvaluateDiffuse( L, N )
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / direct

andom walk - done properly, closely f
vive)

at3 brdf = SampleDiffuse( diffuse, N,
urvive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

Camera Effects

Vignetting

Cheap cameras often suffer from vignetting: reduced brightness of the image for pixels further away from the center.

In a renderer, subtle vignetting can add to the mood of a scene.

Vignetting is simple to implement: just darken the output based on the distance to the center of the screen.

Camera Effects

Chromatic Aberration

This is another effect known from cheap cameras.

A camera may have problems keeping colors for a pixel together, especially near the edges of the image.

In this screenshot (from “Colonial Marines”, a CryEngine game), the effect is used to suggest player damage.

Unreal Tournament Outpost 23

Camera Effects

Chromatic Aberration

Calculating chromatic aberration:

Use a slightly different distance from the center of the screen when reading red, green and blue.

Camera Effects

Noise / Grain

Adding (on purpose) some noise to the rendered image can further emphasize the illusion of watching a movie.

Blair witch project

Camera Effects

Noise / Grain

Adding (on purpose) some noise to the rendered image can further emphasize the illusion of watching a movie.

Film grain is generally not static and changes every frame. A random number generator lets you easily add this effect (keep it subtle!).

When done right, some noise reduces the ‘cleanness’ of a rendered image.

Today's Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

HDR

HDR Bloom

A monitor generally does not directly display HDR images. To suggest brightness, we use hints that our eyes interpret as the result of bright lights:

- Flares
- Glow
- Exposure control

HDR

HDR Bloom

A monitor generally does not directly display HDR images. To suggest brightness, we use hints that our eyes interpret as the result of bright lights:

- Flares
- Glow
- Exposure control

HDR

HDR Bloom

Calculation of HDR bloom:

1. For each pixel, subtract (1,1,1) and clamp to 0
   (this yields an image with only the bright pixels)
2. Apply a Gaussian blur to this buffer
3. Add the result to the original frame buffer.

Unreal Engine 4

HDR

Exposure Control / Tone Mapping

Our eyes adjust light sensitivity based on the brightness of a scene.

Exposure control simulates this effect:

1. Estimate brightness of the scene;
2. Gradually adjust 'exposure';
3. Adjust colors based on exposure.

Exposure control happens *before* the calculation of HDR bloom.

Today’s Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

Color Grading

Color Correction

Changing the color scheme of a scene can dramatically affect the mood.

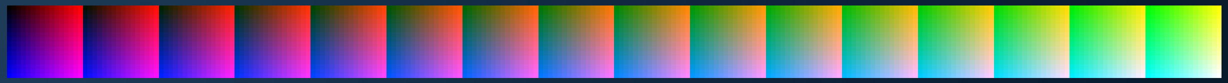
(in the following movie, notice how often the result ends up emphasizing blue and orange)

Color Grading

Color Correction

Color correction in a real-time engine:

1. Take a screenshot from within your game
2. Add a color cube to the image
3. Load the image in Photoshop
4. Apply color correction until desired result is achieved
5. Extract modified color cube
6. Use modified color cube to lookup colors at runtime.




CC
Bloom
Bloom
CC
Niether
Warframe


00'00''00

Today’s Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

Gamma Correction

Concept

Monitors respond in a non-linear fashion to input.

Gamma Correction

Concept

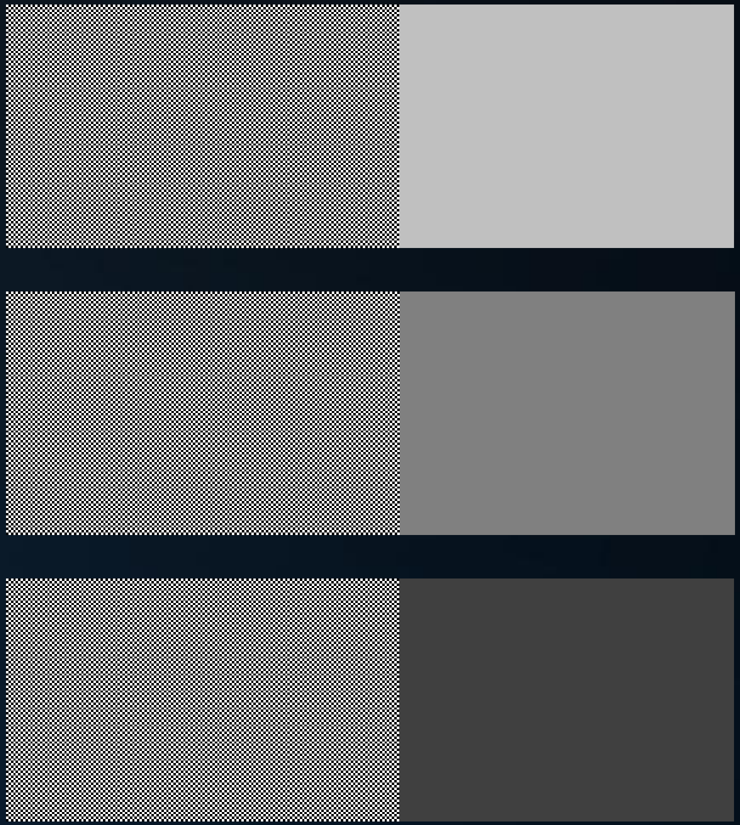
Monitors respond in a non-linear fashion to input:

Displayed intensity $I = a^{\gamma}$

Example for γ=2: $a = \left\{0, \frac{1}{2}, 1\right\} \rightarrow I = \{0, \frac{1}{4}, 1\}$

Let’s see what γ is on the beamer. ☺

On most monitors, $\gamma \approx 2$.



Gamma Correction

How to deal with $\gamma \approx 2$

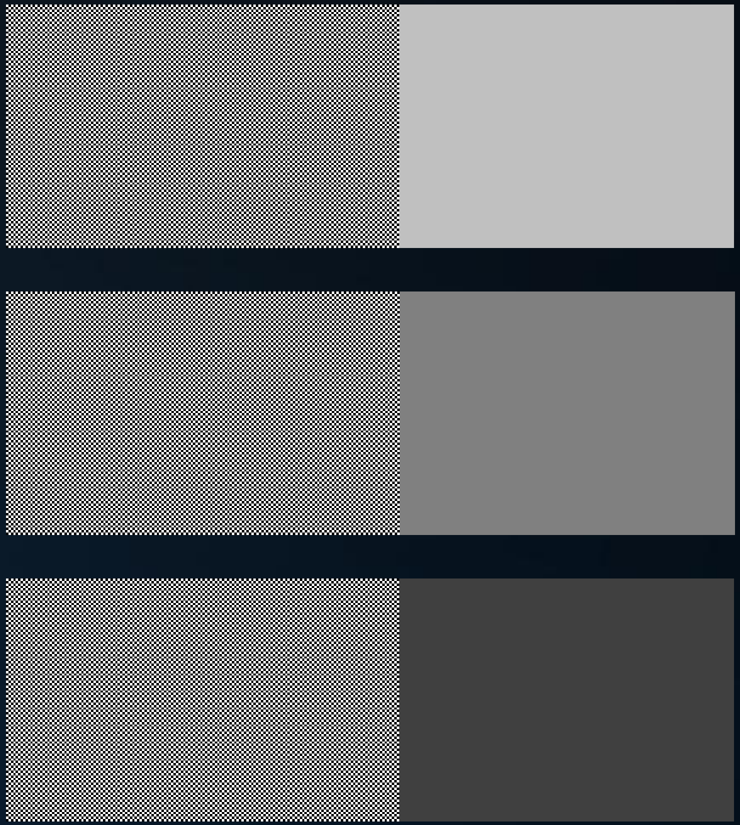
First of all: we will want to do our rendering calculations in a linear fashion.

Assuming that we did this, we will want an intensity of 50% to show up as 50% brightness.

Knowing that $I = a^{\gamma}$,

we adjust the input: $a' = a^{\frac{1}{\gamma}}$ (for γ=2, $a' = \sqrt{a}$),

so that $I = a'^{\gamma} = (a^{\frac{1}{\gamma}})^{\gamma} = a$.



Gamma Correction

How to deal with $\gamma \approx 2$

Apart from ‘gamma correcting’ our output, we also need to pay attention to our input.

This photo looks as good as it does because it was adjusted for screens with $\gamma \approx 2$.

In other words: the intensities stored in this image file have been processed so that a^{γ} yields the intended intensity; i.e. linear values a have been adjusted: $a' = a^{\frac{1}{\gamma}}$.

We restore the linear values for the image as follows:

$$a = a'^{\gamma}$$

Gamma Correction

Linear workflow

To ensure correct (linear) operations:

1. Input data a' is linearized: $a = a'^{\gamma}$
2. All calculations assume linear data
3. Final result is gamma corrected: $a' = a^{\frac{1}{\gamma}}$
4. The monitor applies a non-linear scale to obtain the final linear result a.

Interesting fact: modern monitors have no problem at all displaying linear intensity curves: they are forced to use a non-linear curve because of legacy…

Today’s Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

Depth of Field

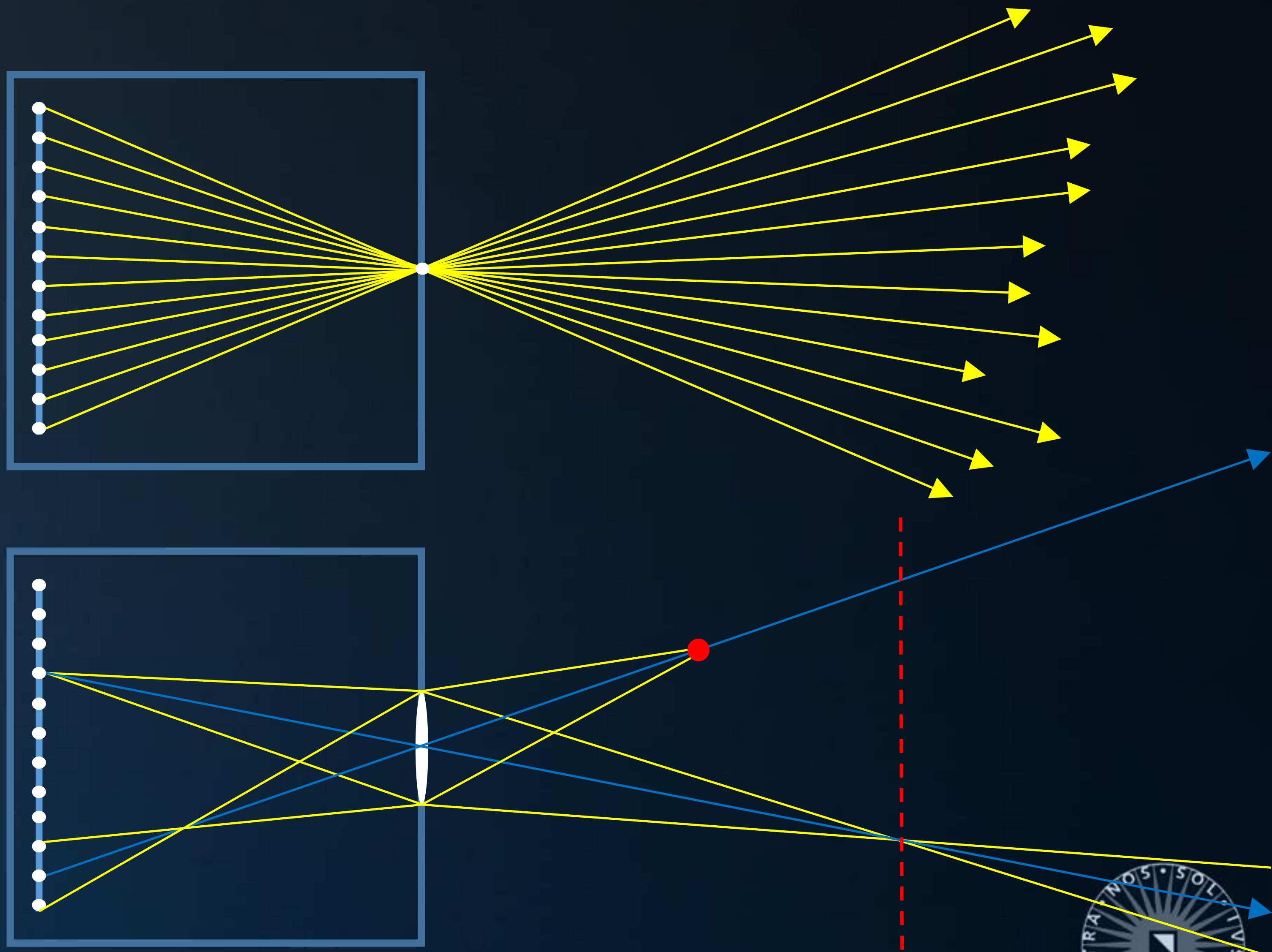
A pinhole camera maps incoming directions to pixels.

Pinhole: aperture size = 0

For aperture sizes > 0, the lens has a focal distance.

Objects not precisely at that distance cause incoming light to be spread out over an area, rather than a point on the film.

This area is called the ‘circle of confusion’.



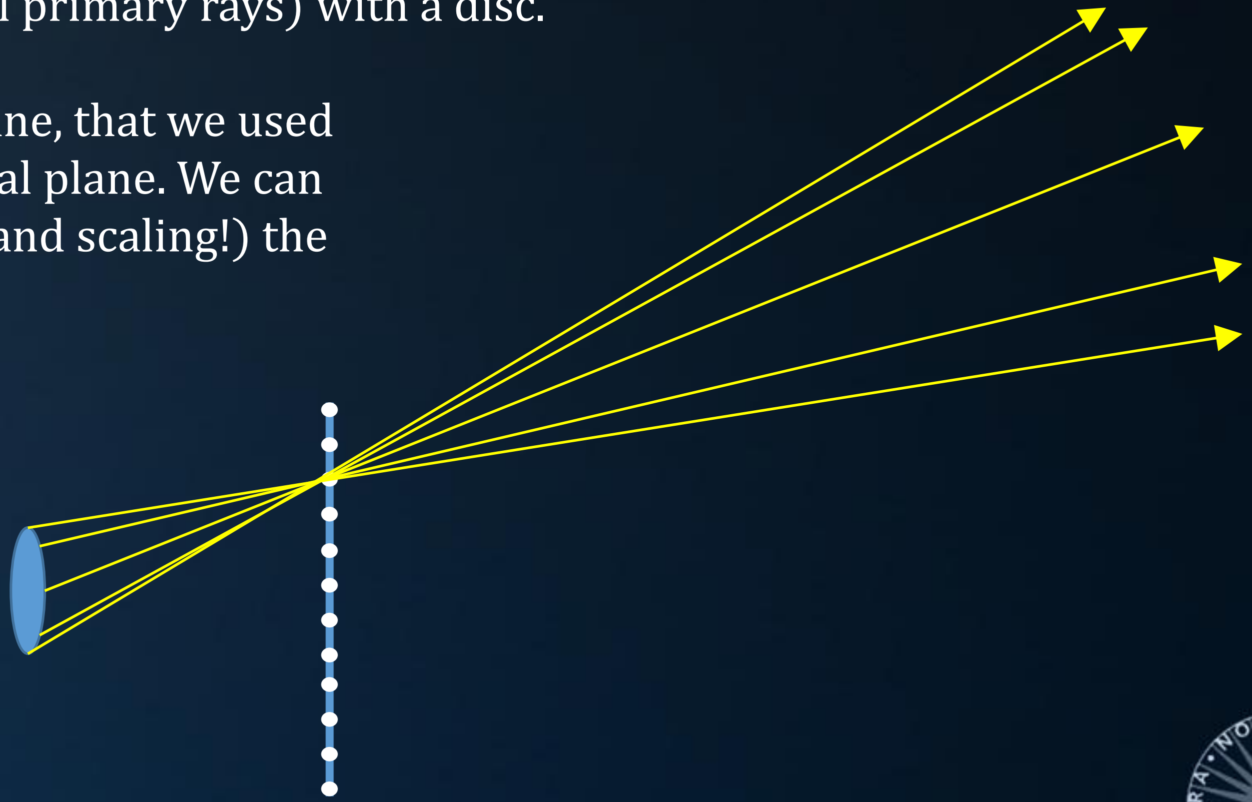
Depth of Field

Depth of Field in a Ray Tracer

To model depth of field in a ray tracer, we exchange the pinhole camera (i.e., a single origin for all primary rays) with a disc.

Notice that the virtual screen plane, that we used to aim our rays at, is now the focal plane. We can shift the focal plane by moving (and scaling!) the virtual plane.

We generate primary rays, using Monte-Carlo, on the ‘lens’.



Depth of Field

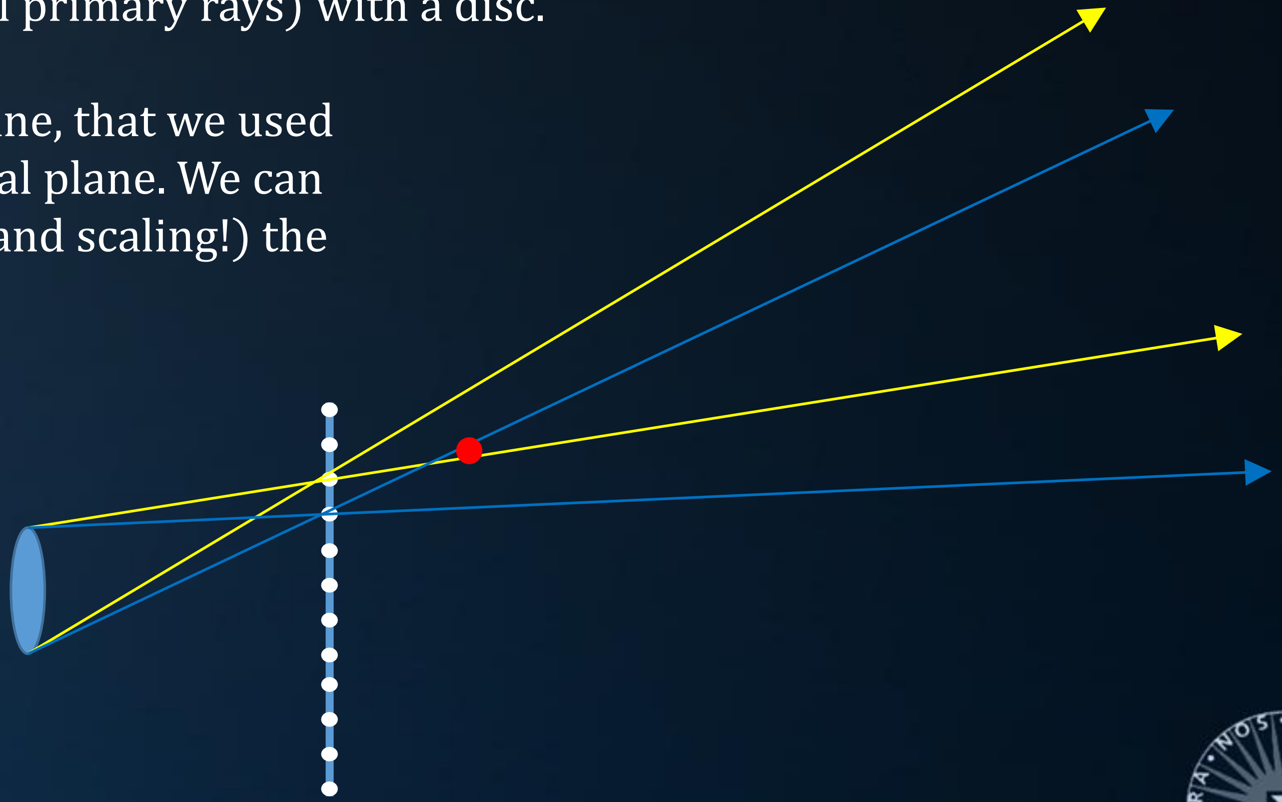
Depth of Field in a Ray Tracer

To model depth of field in a ray tracer, we exchange the pinhole camera (i.e., a single origin for all primary rays) with a disc.

Notice that the virtual screen plane, that we used to aim our rays at, is now the focal plane. We can shift the focal plane by moving (and scaling!) the virtual plane.

We generate primary rays, using Monte-Carlo, on the ‘lens’.

The red dot is now detected by two pixels.



Depth of Field

Depth of Field in a Rasterizer

Depth of field in a rasterizer can be achieved in several ways:

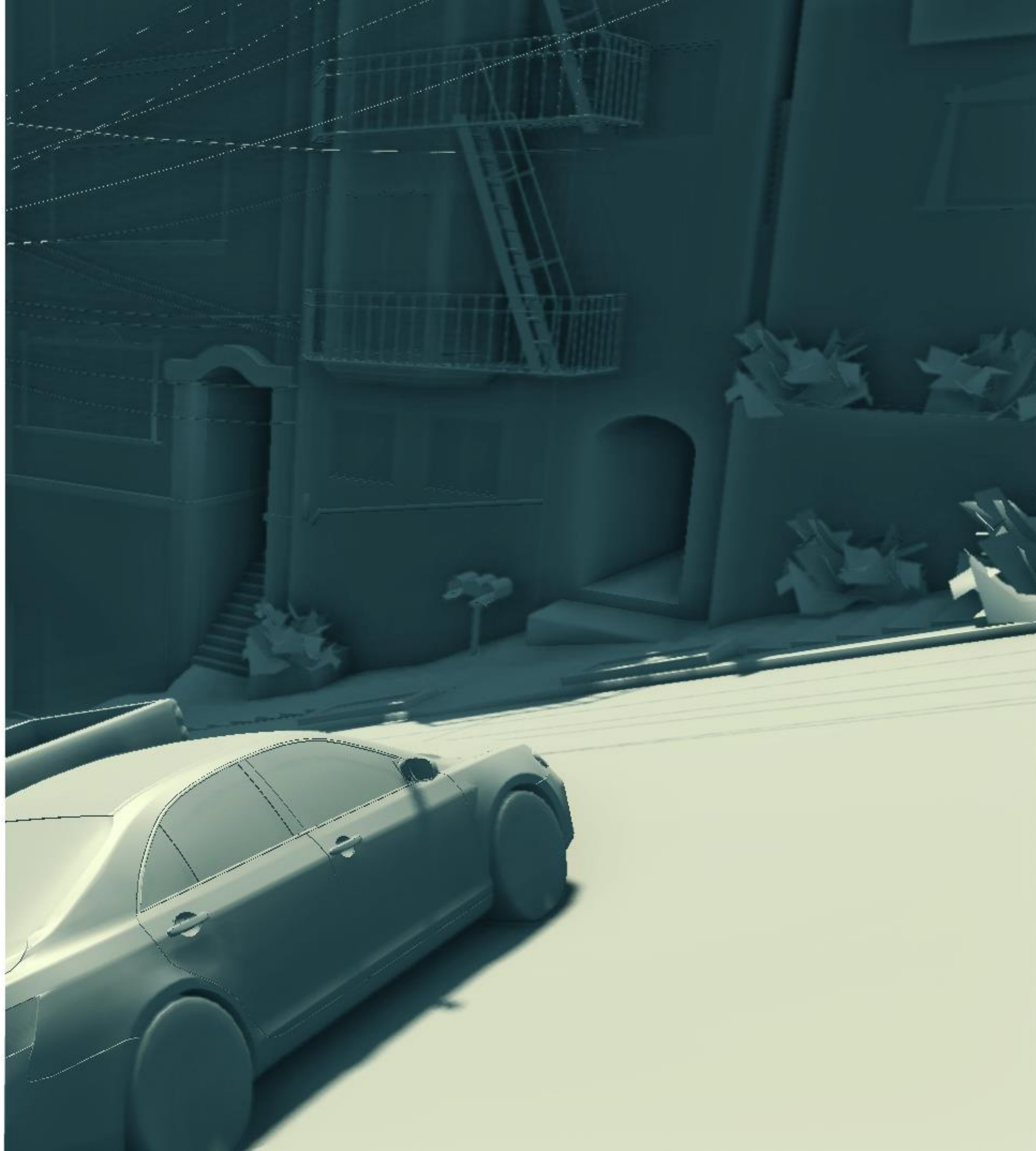
1. Render the scene from several view points, and average the results;
2. Split the scene in layers, render layers separately, apply an appropriate blur to each layer and merge the results;
3. Replace each pixel by a disc sprite, and draw this sprite with a size matching the circle of confusion;
4. Filter the 'in-focus' image to several buffers, and blur each buffer with a different kernel size. Then, for each pixel select the appropriate blurred buffer.
5. As a variant on 4, just blend between a single blurred buffer and the original one.

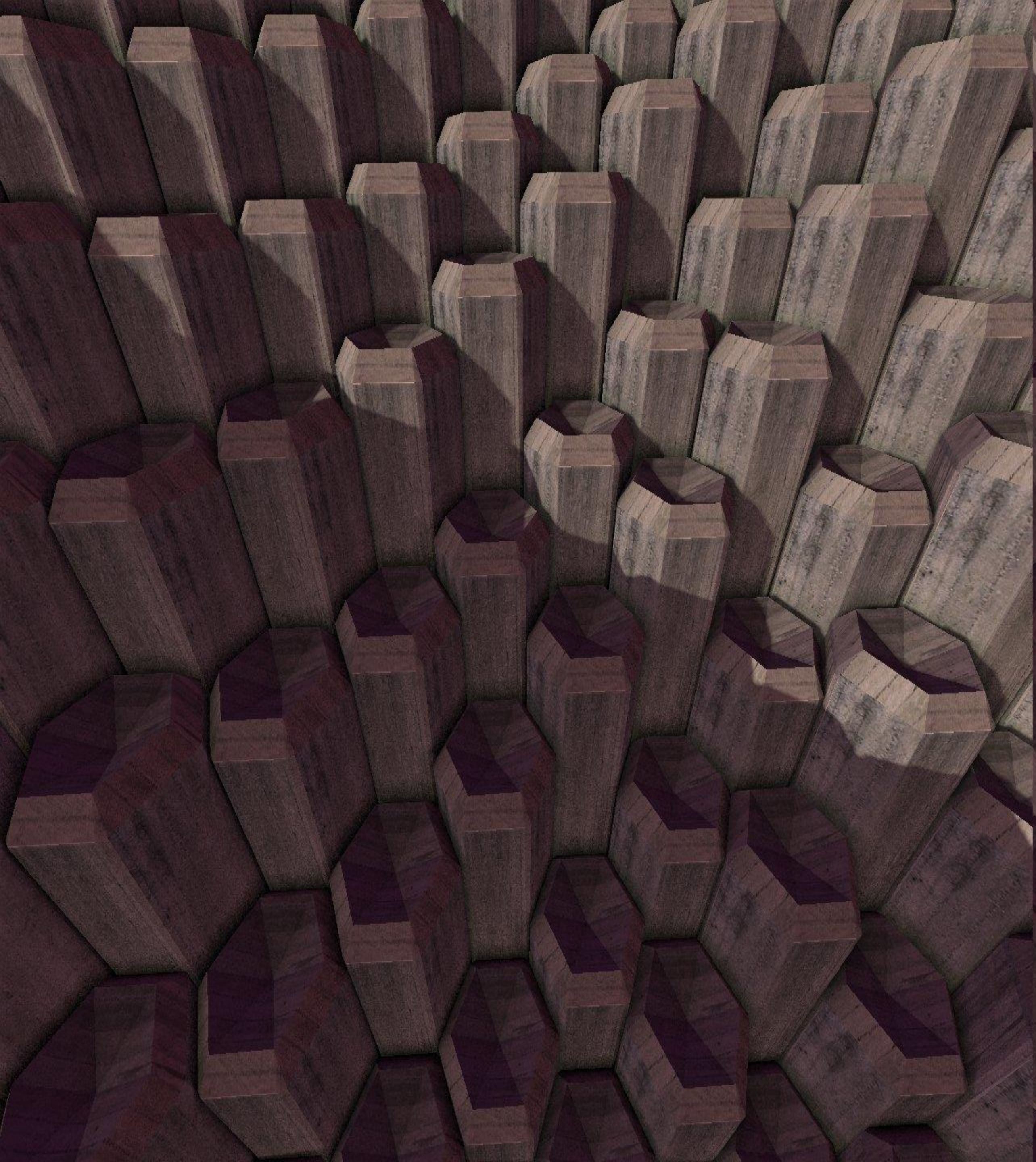
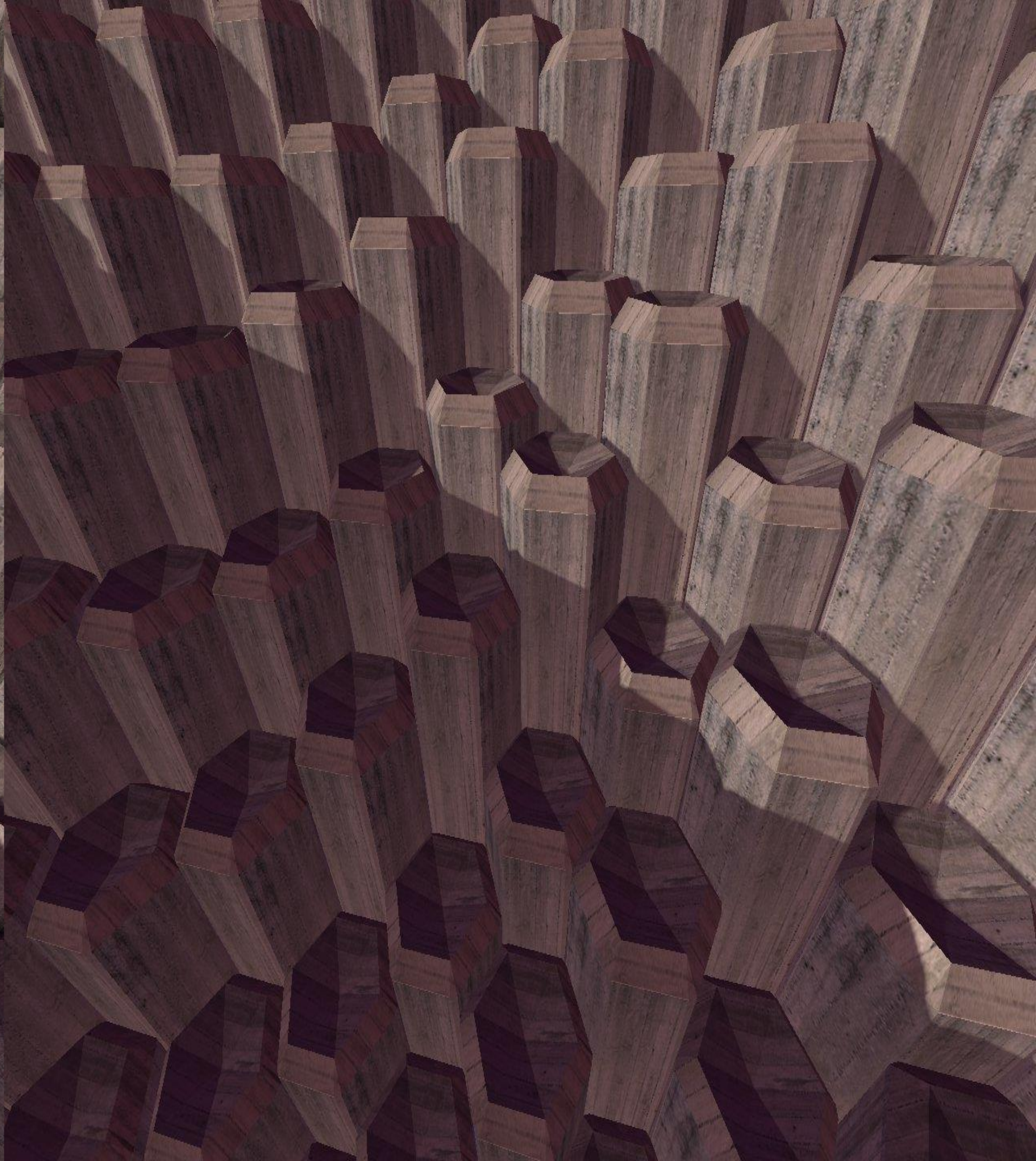
Note that in all cases (except 1), the input is still an image generated by a pinhole camera.


HEALTH
100

Today’s Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections


GPU1 : 42 °C, 9 %, 95 %, 324 MHz
GPU2 : 66 °C, 84 %, 95 %, 1071 MHz
MEM1 : 324 MHz, 790 MB
MEM2 : 3005 MHz, 790 MB
D3D11 : 119.9 FPS
WAVE 1
TAKE DOWN ALL HOSTILES
HOSTILES REMAINING
5
[SPACE] CLIMB
CLIMB
10M
STICKY NOISEMAKER
3
11 30
SILENCED


GPU1 : 41 °C, 7 %, 95 %, 324 MHz
GPU2 : 68 °C, 95 %, 95 %, 1019 MHz
MEM1 : 324 MHz, 838 MB
MEM2 : 3005 MHz, 838 MB
D3D11 : 82.6 FPS
HOSTILES REMAINING
5
[SPACE] CLIMB
[SPACE] CLIMB
10M
STICKY NOISEMAKER
03
11 30
SILENCED

Ambient Occlusion

Concept

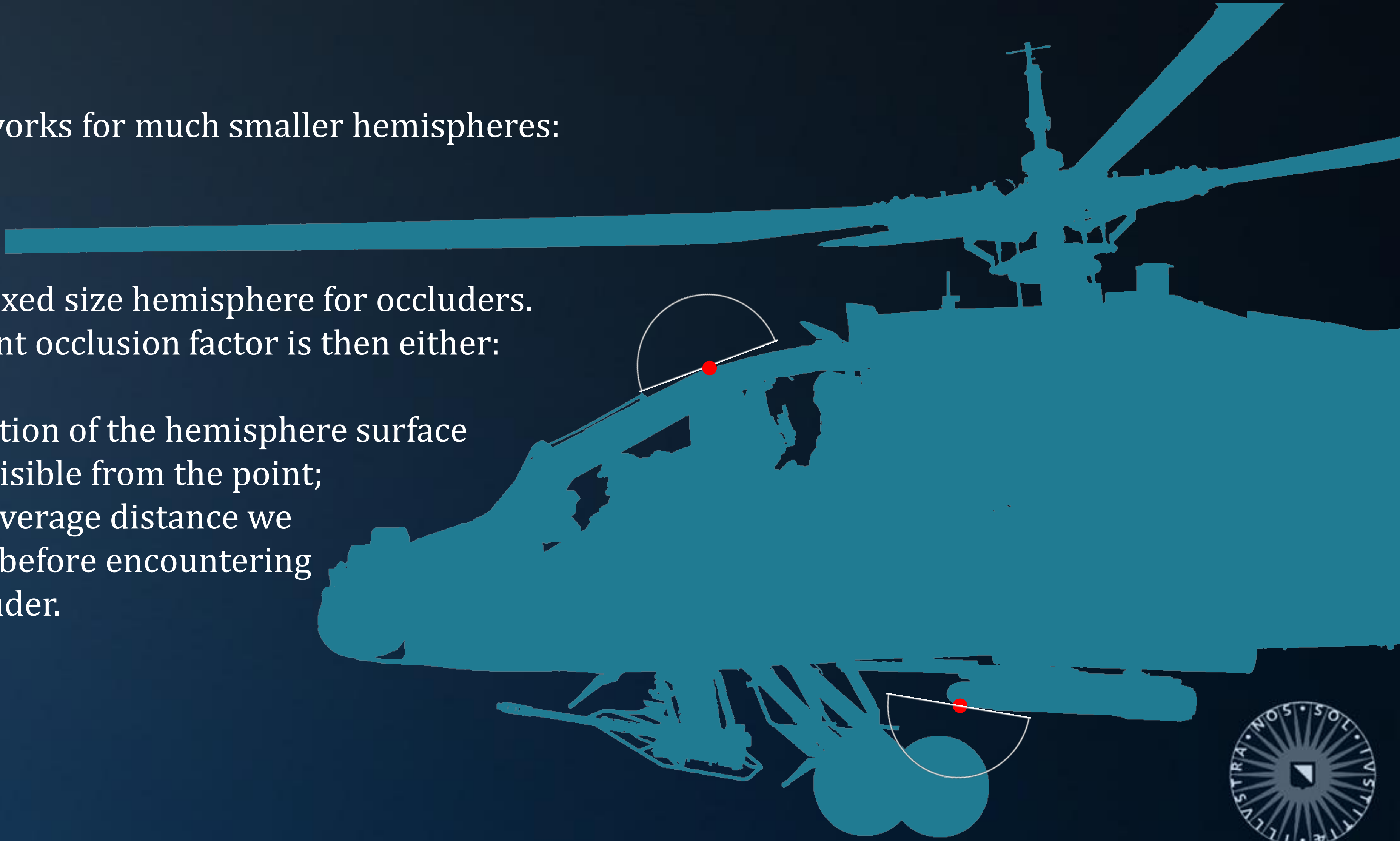
Ambient occlusion was designed to be a scale factor for the ambient factor in the Phong shading model.

A city under a skydome: assuming uniform illumination from the dome, illumination of the buildings is proportional to the visibility of the skydome.

Ambient Occlusion

Concept

This also works for much smaller hemispheres:



We test a fixed size hemisphere for occluders.
The ambient occlusion factor is then either:

- The portion of the hemisphere surface that is visible from the point;
- Or the average distance we can see before encountering an occluder.

Ambient Occlusion

Concept

Ambient occlusion is generally determined using Monte Carlo integration, using a set of rays.

$$AO = \frac{1}{N}\sum_{i=1}^{N} V_{P,\vec{w}}(\vec{N}\cdot\vec{w})$$

where V is 1 or 0, depending on the visibility of points on the hemisphere at a fixed distance.

or

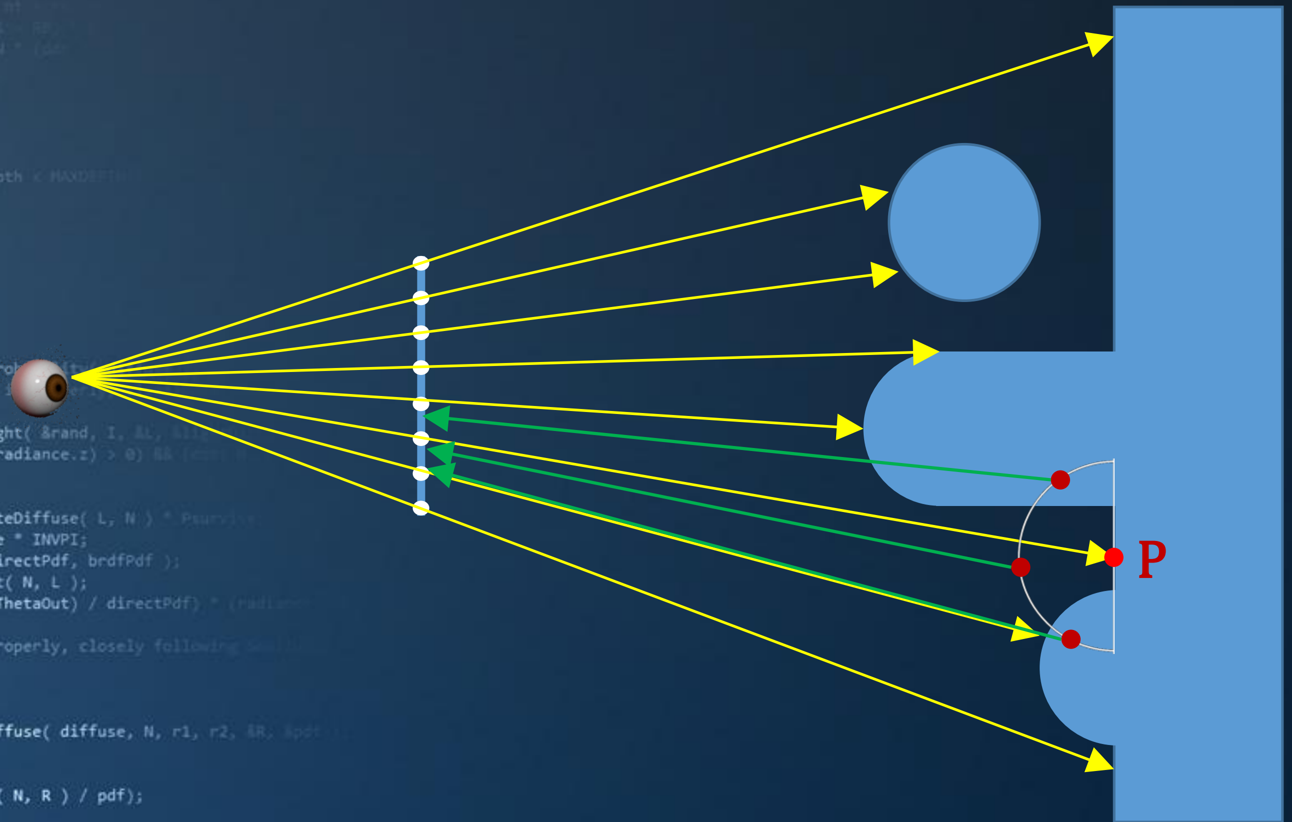
$$AO = \frac{1}{N}\sum_{i=1}^{N} \frac{D_{P,\vec{w}}}{D_{max}}(\vec{N}\cdot\vec{w})$$

where $D_{P,\vec{w}}$ is the distance to the first occluder or a point on a hemisphere with radius $D_{\max}$.

Ambient Occlusion

Screen Space Ambient Occlusion

We can approximate ambient occlusion in screen space, i.e., without actual ray tracing.



1. Using the z-buffer and the view vector, reconstruct a view space coordinate P
2. Generate N random points $S_{1..i}$ around P
3. Project each $S_{1..i}$ back to 2D screen space coordinate S', and lookup z for S'
4. We can now compare S_z to S'_z to estimate occlusion for S.

Ambient Occlusion

Filtering SSAO

Applying the separable Gaussian blur you implemented already is insufficient for filtering SSAO: we don’t want to blur AO values over edges.

We use a *bilateral filter* instead.

Such a filter replaces each value in an image by a weighted average of nearby pixels. Instead of using a fixed weight, the weight is computed on the fly, e.g. based on the view space distance of two points, or the dot between normals for the two pixels.

Today’s Agenda:

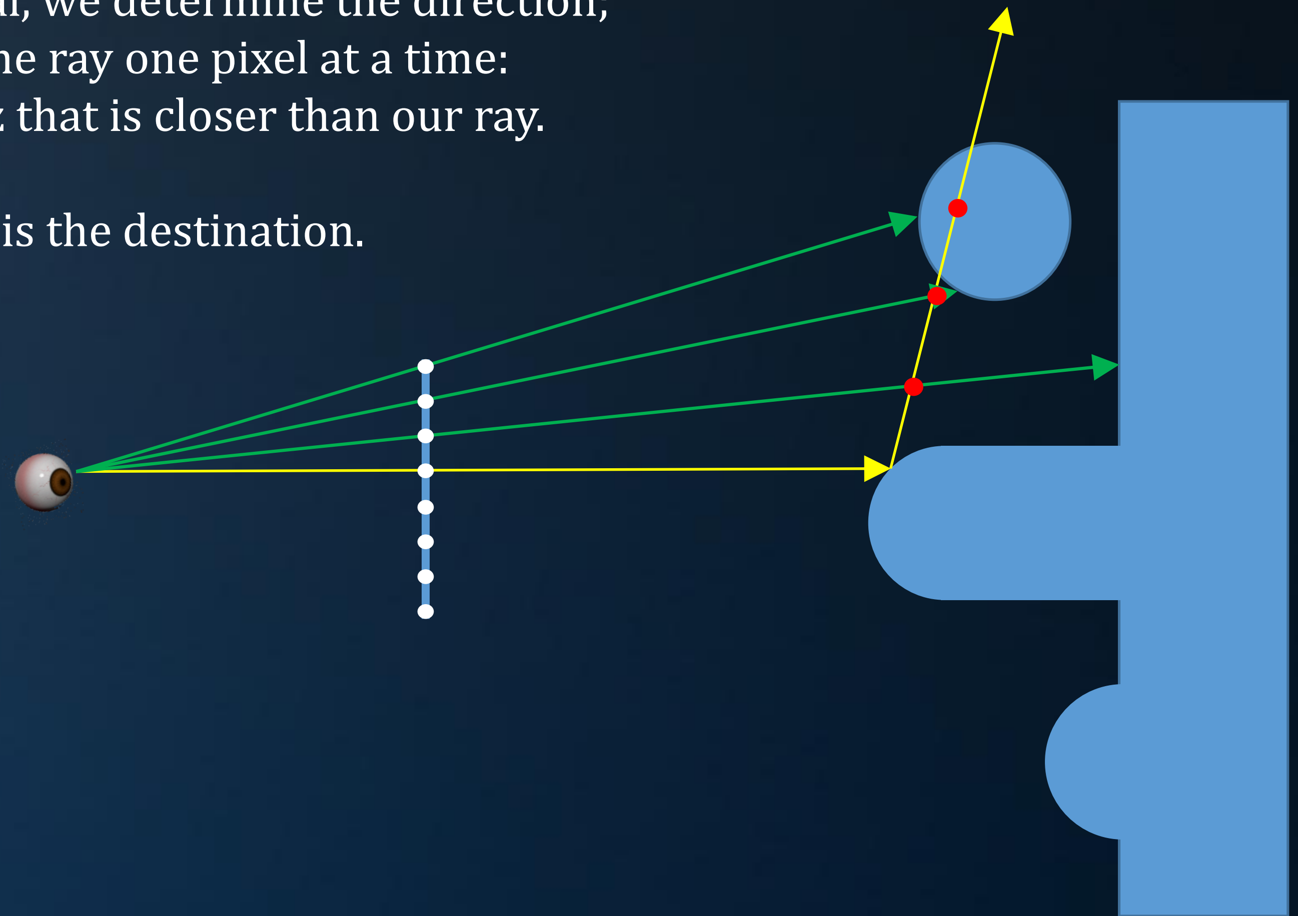
- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

Reflections

Screen Space Reflections

1. Based on depth, we determine the origin of the ray;
2. Based on normal, we determine the direction;
3. We step along the ray one pixel at a time:
4. Until we find a z that is closer than our ray.

The previous point is the destination.



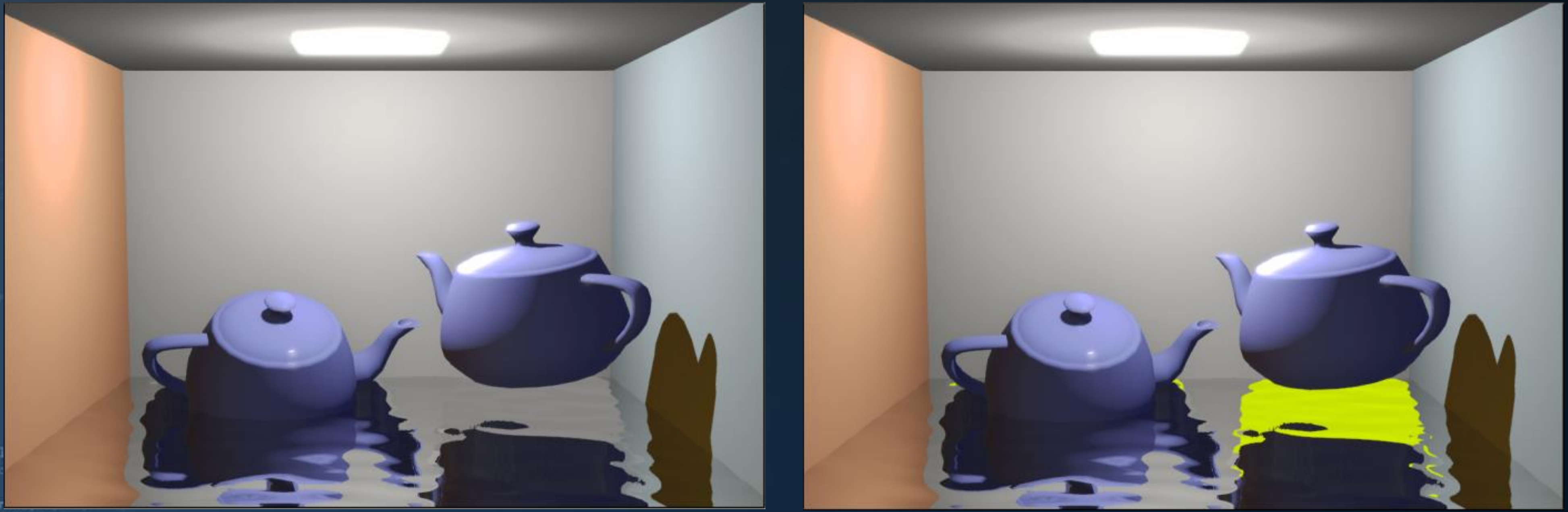
Reflections

Screen Space Reflections

From: http://www.kode80.com/blog/2015/03/11/screen-space-reflections-in-unity-5

Reflections

Screen Space Reflections



“Efficient GPU Screen-Space Ray Tracing”, McGuire & Mara, 2014

Battlefield 4

Killzone Shadowfall

Today’s Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

Famous Last Words

Post Processing Pipeline

In: rendered image, linear color space

- Ambient occlusion
- Screen space reflections
- Tone mapping
- HDR bloom / glare
- Depth of field
- Film grain / vignetting / chromatic aberration
- Color grading
- Gamma correction

Out: post-processed image, gamma corrected

Famous Last Words

Experimenting

A modified P3 template is now available.

New:

```
class RenderTarget
```

Usage:

```
target = new RenderTarget( screen.width, screen.height );
target.Bind();
// rendering will now happen to this target
target.Unbind();
```

Now, the texture identified by target.GetTextureID() contains your rendered image.

Famous Last Words

Experimenting

A modified P3 template is now available.

New:

```
class ScreenQuad
```

Usage:

```
quad = new ScreenQuad();
quad.Render( postprocShader, target.GetTextureID() );
```

This renders a full-screen quad using any texture (here: the render target texture), using the supplied shader. Note: no transform is used.

Famous Last Words

Example shader:

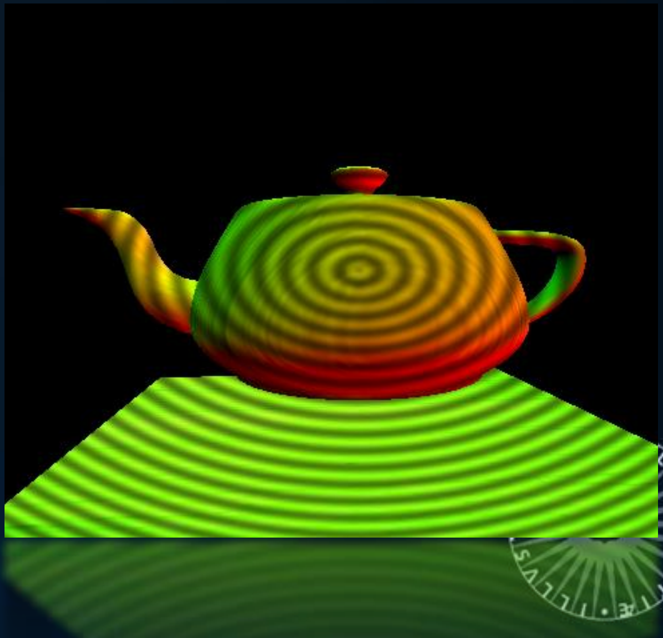
```
#version 330

// shader input
in vec2 P;                          // fragment position in screen space
in vec2 uv;                         // interpolated texture coordinates
uniform sampler2D pixels;           // input texture (1st pass render target)

// shader output
out vec3 outputColor;

void main()
{
    // retrieve input pixel
    outputColor = texture( pixels, uv ).rgb;
    // apply dummy postprocessing effect
    float dx = P.x - 0.5, dy = P.y - 0.5;
    float distance = sqrt( dx * dx + dy * dy );
    outputColor *= sin( distance * 200.0f ) * 0.25f + 0.75f;
}

// EOF
```



Today’s Agenda:

- The Postprocessing Pipeline
  - Vignetting, Chromatic Aberration
  - Film Grain
  - HDR effects
  - Color Grading
  - Depth of Field
- Screen Space Algorithms
  - Ambient Occlusion
  - Screen Space Reflections

INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 12: “Post-processing”

END of “Post-processing”

next lecture: “Stochastic Methods”