tic: ⊾ (depth < 144

: = inside / l it = nt / nc, dd os2t = 1.0f 0, N ); 3)

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

E <sup>=</sup> diffuse = true;

--:fl + refr)) && (depth < HAAD]

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Puncture st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

# 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





#### st3 factor = diffuse \* INVPI: st weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

#### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* tic: ⊾ (depth < 100

at a = nt - nc, b = nt - stat  $Tr = 1 - (R0 + (1 - Tr))R = (D^{-1} nnt - N^{-1})$ 

= diffuse; = true:

• •f] + refr)) 88 (death o NAA

>, N ); -efl \* E \* diffu = true;

#### AXDEPTH)

survive = SurvivalProbability( different estimation - doing it properly if; radiance = SampleLight( &rand, 1, 8, 8, 8) e.x + radiance.y + radiance.z) > 0) 88

v = true; at brdfPdf = EvaluateDiffuse( L, N, ) \* Pau st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

indom walk - done properly, closely follo vive)

; ht3 brdf = SampleDiffuse( diffuse, N, r1, r2, pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:





it = nt / nc, ddo 552t = 1.0f - nnt -9, N ); 3)

st a = nt - nc, b + nt + st Tr = 1 - (80 + (1 - 1 Tr) R = (0 \* nnt - 11 -

= diffuse; = true:

= true;

-:fl + refr)) && (depth is MANDITT

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

#### AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Prost3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf);

andom walk - done properly, closely follow. /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:





## Introduction

fice € (depth < 182

= inside / 1 it = nt / nc, dde os2t = 1.0f - nnt -O, N ); B)

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

= diffuse; = true;

-: efl + refr)) && (depth k HANDIII

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

AXDEPTH)

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

andom walk - done properly, closely following /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, F1, F2, R, F3, pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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.



), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff adiance = SampleLight( &rand, I, LL e.x + radiance.y + radiance.z) > 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ) at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR rvive; pdf; 1 = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Purpose: simulating camera / sensor behavior

Bright lights:

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







## Camera Effects

fice € (depth < 10.5

= inside / 1 it = nt / nc, dde os2t = 1.0f - nnt -O, N ); B)

st a = nt - nc, b - nt st Tr = 1 - (R0 + (1 Tr) R = (0 \* nnt - N

= diffuse; = true;

efl + refr)) && (depth k HANDIIII

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

#### AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pours) st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, R, D) pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Purpose: simulating camera / sensor behavior

### Camera imperfections:

- Vignetting
- Chromatic aberration
- Noise / grain









tice ≰ (depth < 10.5

= inside / : it = nt / nc, dde ss2t = 1.8f - nnt 3, N ); 3)

E <sup>=</sup> diffuse = true;

-:fl + refr)) && (depth & MACONNIN

D, N ); =efl \* E \* diffuse; = true;

AXDEPTH)

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

w = true; ot brdfPdf = EvaluateDiffuse( L, N, ) \* Pu st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); ot cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

; t33 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, hpr urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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.



tice ≰ (depth < 10.5

: = inside / 1 it = nt / nc, dda os2t = 1.0f = nnt D, N ); ∂)

at a = nt - nc, b + nt + + at Tr = 1 - (R0 + (1 - 10 Tr) R = (D \* nnt - N \*

= diffuse = true;

: :fl + refr)) && (depth & HARDITIN

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

#### AXDEPTH)

survive = SurvivalProbability( different estimation - doing it properly if; radiance = SampleLight( %rand, 1, %) e.x + radiance.y + radiance.z) > 0) %%

v = true;

at brdfPdf = EvaluateDiffuse( L, N.) \* Pel st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, po pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Lens Flares

### "Physically-Based Real-Time Lens Flare Rendering", Hullin et al., 2011

			) (] ¦ ))				$\mathbb{W}$			)]		
Canon EF 70	0–200mm f/2.8L	Nikon 80-	200mm f/2.8	Itoh 100–1	45mm f/3.5	Angenieux Biotar	100mm f/1.1	Kreitzer Te	le 390mm f/	/5.6	Brendel Tessar 10	00mm f/2.8
0.5/7.5 fps		1.2/23 fps		3.1/39 fps		1.8/58 fps		7.7/110 fps			18/228 fps	
0 7/9 5 fps		3 2/30 fps	···· · · · · · · · · · · · · · · · · ·	8 6/47 fps		6 4/84 fps		29/110 fps		1	21/189 fps	





### Lens Flares

tic: ⊾ (depth < N

= inside / : it = nt / nc, do ss2t = 1.0f - ... 5, N ); 3)

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

= diffuse = true;

-:fl + refr)) && (depth & MADII

D, N ); -efl \* E \* diffus: = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( %rand, I, M) e.x + radiance.y + radiance.z) > 0) %

v = true; tbrdfPdf = EvaluateDiffuse( L, N ) = Pauro st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, R, staturvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:



### From: <u>www.alienscribbleinteractive.com/Tutorials/lens\_flare\_tutorial.html</u>



tica ⊾ (depth < NAS

= inside / L it = nt / nc, dde ss2t = 1.0f - nnt 5, N ); 3)

at a = nt - nc, b - nt - n at Tr = 1 - (80 + 1 Tr) R = (D \* nnt - N

= diffuse = true;

-: :fl + refr)) && (depth k HADDE

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

#### AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight( @rand I = 1) e.x + radiance.y + radiance.r) = 0.000

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pource st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 0000

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, Dod prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

### Vignetting

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





D, N ); -efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( different estimation - doing it properly) if; radiance = SampleLight( %rand, I, &t e.x + radiance.y + radiance.z) > 0)

v = true;

st brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse \* INVPI;
st weight = Mis2( directPdf, brdfPdf
st cosThetaOut = dot( N, L );
E \* ((weight \* cosThetaOut) / direct

andom walk - done properly, closely /ive)

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







fice ⊾(depth < 1935

= = inside / L it = nt / nc, dde ss2t = 1.0f - nnt 5, N ); 3)

st a = nt - nc, b - nt st Tr = 1 - (R0 + (1 fr) R = (D \* nnt - N

= diffuse = true;

: efl + refr)) && (depth < HANDIIII

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it proper) if; radiance = SampleLight( &rand, I .x + radiance.y + radiance.z) 0

v = true; t brdfPdf = EvaluateDiffuse( L, N ) Process st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, so pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## 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.



tic: K (depth ⊂ NASS

= inside / 1 it = nt / nc, dde ss2t = 1.0f = ont 5, N ); 3)

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

= diffuse; = true;

: :fl + refr)) && (depth < HANDIIII

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

#### AXDEPTH)

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) \* Pause st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* India

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, pr pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### **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.





11:05

2=

1000

-inint

0

Frank

- -

+100

0

88

-

86

10

-----

9

0

-



INFOGR – Lecture 12 – "Advanced Shading"

## Camera Effects

tica ≰ (depth < 1000

= inside / 1 it = nt / nc, dde ss2t = 1.0f = nnt ), N ); 3)

st a = nt - nc, b - nt ) st Tr = 1 - (80 + (1 - 1 fr) R = (D \* nnt - N \* )

E <sup>=</sup> diffuse = true;

-: :fl + refr)) && (depth is HANDIIII

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

AXDEPTH)

v = true; t brdfPdf = EvaluateDiffuse( L, N ) \* Purple st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* Pulple

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### **Chromatic Aberration**

Calculating chromatic aberration:

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



tic: ⊾ (depth ⊂ 100

= inside / : it = nt / nc, dde ss2t = 1.0f - nnt 5, N ); 3)

at a = nt - nc, b + nt + + at Tr = 1 - (R0 + (1 - 10 Tr) R = (D \* nnt - N \*

= diffuse = true;

-: :fl + refr)) && (depth < HAND)

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

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( %rand, I, M) e.x + radiance.y + radiance.z) > 0) %

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pour base st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 0000

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, Doffurvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Noise / Grain

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







tic: ⊾ (depth < 100

= inside / L it = nt / nc, dde ss2t = 1.0f = nnt 5, N ); 3)

st a = nt - nc, b - nt st Tr = 1 - (80 + (1 fr) R = (D \* nnt - N

= diffuse; = true;

efl + refr)) && (depth k HANDIIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I) e.x + radiance.y + radiance.r) = 0.000

v = true; t brdfPdf = EvaluateDiffuse( L, N ) \* Pour st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* Pour E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, brd pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

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





#### st3 factor = diffuse \* INVPI: st weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

#### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

tic: K (depth < 10.55

= inside / 1 it = nt / nc, ddo ss2t = 1.0f = ont 5, N ); 3)

at a = nt - nc, b - nt - at Tr = 1 - (R0 + 11 - 11 Tr) R = (D \* nnt - N

= diffuse; = true;

-:fl + refr)) && (depth < MAXDIIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it property ff; radiance = SampleLight( %rand, I & A e.x + radiance.y + radiance.z) > 0)

v = true; st brdfPdf = EvaluateDiffuse( L, N ) = st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L );

E \* ((weight \* cosThetaOut) / directPdf) \* (masheddow walk - done properly, closely following)

vive)

; t33 brdf = SampleDiffuse( diffuse, N, F1, F2, SR, Soff urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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





tic: ⊾ (depth < 100⊂

: = inside / 1 it = nt / nc, dde ss2t = 1.0f = nnt 1 5, N ); 3)

st a = nt - nc, b - nt - ncst Tr = 1 - (80 + 1) Tr ) R = (0 \* nnt - 1)

= diffuse; = true;

-:fl + refr)) && (depth < HADIIII

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

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; radiance = SampleLight( &rand, I, M. e.x + radiance.y + radiance.z) > 0) M.

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pun st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follow: /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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







Le: (depth < NASS

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nnt ), N ); 3)

st a = nt - nc, b - nt st Tr = 1 - (80 + (1 Tr) R = (0 \* nnt - N

= diffuse; = true;

-: efl + refr)) && (depth is MANDIIII

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property ff; radiance = SampleLight( %rand, I e.x + radiance.y + radiance.z) = 0.000

v = true;

it brdfPdf = EvaluateDiffuse( L, N.) Power st3 factor = diffuse \* INVPI; ot weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, hoff pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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

fice € (depth ⊂ 1930

: = inside / l it = nt / nc, dde os2t = l.0f - nnt -D, N ); B)

at a = nt - nc, b = nt - ncat Tr = 1 - (R0 + 11 - 10)Tr ) R = (D \* nnt - 10)

= diffuse; = true;

≕ efl + refr)) 88 (depth k HAADIII

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property ff; radiance = SampleLight( %rand, I e.x + radiance.y + radiance.z) = 0)

v = true;

st brdfPdf = EvaluateDiffuse( L, N ) \* Paulot st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* Paulot

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, bof urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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





#### st3 factor = diffuse \* INVPI: st weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

#### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

## **Color Grading**

tica ⊾ (depth (c)100

= inside / 1 it = nt / nc, dde ss2t = 1.0f = ont / 5, N ); 3)

st a = nt - nc, b = nt - ncst Tr = 1 - (R0 + (1 - 1))Tr ) R = (D \* nnt - N - 1)

= diffuse; = true;

efl + refr)) && (depth & MANDIIII

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N, ) \* Purchas st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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**

tica ≰ (depth ≤ 100⊂

: = inside / l it = nt / nc, ddo os2t = 1.0f - nn: 0, N ); 0)

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

= diffuse; = true;

-: :fl + refr)) && (depth < NADII

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I. &... e.x + radiance.y + radiance.z) > 0) %

v = true;

at brdfPdf = EvaluateDiffuse( L, N) Pauro st3 factor = diffuse = INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

indom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, D) pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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

# Niether

Warframe





## 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





#### st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

tica k (depth k 1933

= = inside / 1 it = nt / nc, dde os2t = 1.0f = nnt ), N ); 3)

at a = nt - nc, b - nt - --at Tr = 1 - (80 + (1) Tr) R = (D \* nnt - N

= diffuse; = true;

⊂ efl + refr)) && (depth < HAODUT

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( different estimation - doing it properly, if; radiance = SampleLight( %rand, I, %L, SI) e.x + radiance.y + radiance.z) > 0) %%

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pourse st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 0000

andom walk - done properly, closely following : /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, R, l); pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Concept



Monitors respond in a non-linear fashion to input.



tic: ⊾ (depth < 100

= inside / : it = nt / nc, dde ss2t = 1.8f - nn: 5, N ); 8)

st a = nt - hc, b = nt - hc, st Tr = 1 - (R0 + (1 - 1))(r) R = (0 \* nnt - 1)

= diffuse = true;

-: :fl + refr)) && (depth & HADIII

D, N ); -efl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, e.x + radiance.y + radiance.z) > 0) #

v = true; at brdfPdf = EvaluateDiffuse( L, N) \* Point st3 factor = diffuse \* INVPI; bt weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

sndom walk - done properly, closely following vive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, sr urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Concept

Monitors respond in a non-linear fashion to input: Displayed intensity  $I = a^{\gamma}$ 

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

Let's see what  $\gamma$  is on the beamer.  $\bigcirc$ 

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





11c) 6 (depth < 112)

: = inside / 1 it = nt / nc, dde os2t = 1.0f - nnt ' D, N ); D)

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

= diffuse = true;

: :fl + refr)) && (depth k HANDIII

D, N ); =efl \* E \* diffuse; = true;

AXDEPTH)

v = true; t brdfPdf = EvaluateDiffuse( L, N.) Promote st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* \*\*\*

sndom walk - done properly, closely following vive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, bp4 pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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  $\gamma = 2$ ,  $a' = \sqrt{a}$ ), so that  $I = a'^{\gamma} = (a^{\frac{1}{\gamma}})^{\gamma} = a$ .



![](_page_44_Picture_18.jpeg)

tica k (depth < 10.5

: = inside / 1 it = nt / nc, dde os2t = 1.0f = ont ), N ); 3)

st a = nt - nc, b - mt st Tr = 1 - (80 + (1 Tr) R = (D \* nnt - N

= diffuse; = true;

efl + refr)) && (depth & NADIIII

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, &., ) e.x + radiance.y + radiance.z) > 0) ##

v = true; st brdfPdf = EvaluateDiffuse( L, N ) \* Pro st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following /ive)

, t3 brdf = SampleDiffuse( diffuse, N, r1, r2, 18, 1999 µrvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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  $a^{\gamma}$  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:

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

![](_page_45_Picture_19.jpeg)

![](_page_45_Picture_20.jpeg)

tic: k (depth < NUS

= inside / l it = nt / nc, ddc os2t = 1.0f - nnt -D, N ); B)

st a = nt - nc, b - m) st Tr = 1 - (R0 + (1 - 1) Tr) R = (D \* nnt - N

= diffuse; = true;

-: :fl + refr)) && (depth < HADIII

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( %rand, I, M.) e.x + radiance.y + radiance.z) > 0) MM

v = true; t brdfPdf = EvaluateDiffuse( L, N.) Provident st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 1

andom walk - done properly, closely following : /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, sr urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

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:  $a' = a^{\overline{\gamma}}$
- 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...

![](_page_46_Picture_20.jpeg)

![](_page_46_Picture_21.jpeg)

## 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

![](_page_47_Picture_10.jpeg)

![](_page_47_Picture_11.jpeg)

#### st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

### INFOGR – Lecture 12 – "Advanced Shading"

# 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'. st weight = Mis2( directPdf, brdfPd

at cosThetaOut = dot( N, L ); E ((weight \* cosThetaOut) / directPdf)

st Tr = 1

), N );

= true;

AXDEPTH)

v = true

if

-efl \* E \* diffuse;

survive = SurvivalProbabil.

adiance = SampleLight( &rand radiance.v + radiance

st brdfPdf = EvaluateDiffuse(

st3 factor = diffuse \* INVPI;

andom walk - done properly, closely follo /ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, AR, rvive; pdf; i = E \* brdf \* (dot( N, R ) / pdf); sion = true

![](_page_48_Picture_11.jpeg)

INFOGR – Lecture 12 – "Advanced Shading"

# Depth of Field

tic: ≰ (depth ⊂ 1925)

= inside / 1 it = nt / nc, ddo ss2t = 1.0f - nnt " 5, N ); 3)

st a = nt - hc, b = nt - hc, st Tr = 1 - (R0 + (1 - R))(r) R = (0 + nnt - R - 1)

= diffuse; = true;

efl + refr)) && (depth k HANDIIII

D, N ); -efl \* E \* diffuse; = true;

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N.) \* Punct st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf); at cosThetaOut = dot( N, L); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following : /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, N, st urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

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'.

![](_page_49_Picture_17.jpeg)

INFOGR – Lecture 12 – "Advanced Shading"

# Depth of Field

tic: ≰ (depth ⊂ 1925)

= inside / 1 it = nt / nc, ddo ss2t = 1.0f - nnt ' 5, N ); 3)

st a = nt - hc, b = nt - hc, st Tr = 1 - (R0 + (1 - R))(r) R = (0 + nnt - R - 1)

= diffuse; = true;

efl + refr)) && (depth k HANDIIII

D, N ); =efl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, I) e.x + radiance.y + radiance.z) > 0) #10

v = true; at brdfPdf = EvaluateDiffuse( L, N, ) \* Pu st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely follow. /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, loc prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

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.

![](_page_50_Picture_18.jpeg)

# Depth of Field

at a = m

), N );

AXDEPTH)

v = true:

/ive)

efl + refr)) && (dep

efl \* E \* diffuse;

survive = SurvivalProbability

radiance = SampleLight( &rand
e.x + radiance.y + radiance.z

st brdfPdf = EvaluateDiffuse( L, N )
st3 factor = diffuse \* INVPI;
st weight = Mis2( directPdf, brdfPdf
st cosThetaOut = dot( N, L );

E ((weight \* cosThetaOut) / directPdf

andom walk - done properly, closely fol

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.

![](_page_51_Picture_10.jpeg)

st3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, U urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 52

![](_page_52_Picture_0.jpeg)

## 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

![](_page_53_Picture_10.jpeg)

![](_page_53_Picture_11.jpeg)

#### st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

![](_page_54_Picture_0.jpeg)

![](_page_55_Picture_0.jpeg)

### WAVE 1 TAKE DOWN ALL HOSTILES HOSTILES REMAINING 5

1071 MHZ

[SPACE] CLIMB

10M

STICKY NOISEMAKER

![](_page_56_Picture_3.jpeg)

![](_page_57_Picture_0.jpeg)

tica ⊾ (depth < 10.5

= = inside / 1 it = nt / nc, dde ss2t = 1.0f - not 1 5, N ); 3)

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

= diffuse = true;

-:fl + refr)) && (depth & NADIII

D, N ); -efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property ff; radiance = SampleLight( &rand, I e.x + radiance.y + radiance.r) > 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Pours st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following a /ive)

, t33 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, so prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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.

![](_page_58_Picture_16.jpeg)

tica ≰ (depth < 10.5

: = inside / 1 it = nt / nc, dde os2t = 1.0f = nnt 5, N ); 3)

st a = nt - nc, b = nt so st Tr = 1 - (R0 + (1 - 1))Tr ) R = (D \* nnt - N - 1)

E \* diffuse; = true;

efl + refr)) && (depth < HANDIIII

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly, if; radiance = SampleLight( &rand, I, L, L) e.x + radiance.y + radiance.z) > 0) #10

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

andom walk - done properly, closely following : /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, bod urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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.

![](_page_59_Picture_19.jpeg)

tic: ⊾ (depth < 100

= inside / :
it = nt / nc, dde
ss2t = 1.0f = nnt
), N );

st  $a = nt - nc_1 b - nt$ st Tr = 1 - (R0 + (1))Tr ) R = (0 \* nnt - N)

= diffuse = true;

-:fl + refr)) && (depth & MADIII

D, N ); ~efl \* E \* diffuse; = true;

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; adiance = SampleLight( %rand, I, %) e.x + radiance.y + radiance.z) > 0) %%

w = true; at brdfPdf = EvaluateDiffuse( L, N, ) \* Pun 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)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, set pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### Concept

or

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

![](_page_60_Figure_15.jpeg)

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

![](_page_60_Picture_17.jpeg)

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

![](_page_60_Picture_19.jpeg)

), N ); efl \* E \* diffuse = true;

AXDEPTH)

urvive = Survival adiance = SampleLight

v = true at3 factor = diffuse st weight = Mis2( directPdf, brdfPd) at cosThetaOut = dot( N, L ) E \* ((weight \* cosThetaOut) / directPd

andom walk - done properly, closely foll /ive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, D rvive; pdf; i = E \* brdf \* (dot( N, R ) / pdf);

Screen Space Ambient Occlusion

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

![](_page_61_Picture_15.jpeg)

- 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*.

![](_page_61_Picture_20.jpeg)

![](_page_62_Picture_0.jpeg)

tice ⊾ (depth < 1000

= inside / L it = nt / nc, dde ss2t = 1.0f - nnt 5, N ); 3)

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

= diffuse = true;

: :fl + refr)) && (depth is HANDIIII

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, I) e.x + radiance.y + radiance.z) > 0) %%

v = true; t brdfPdf = EvaluateDiffuse( L, N.) Promote st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* \*\*\*

andom walk - done properly, closely following -/ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### 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.

![](_page_63_Picture_17.jpeg)

## 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

![](_page_64_Picture_10.jpeg)

![](_page_64_Picture_11.jpeg)

#### st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

# Reflections

tic: k (depth < 100

= inside / : it = nt / nc, dde ss2t = 1.8f - nnt 3, N ); 3)

st  $a = nt - hc_{2} b + nt - st$ st Tr = 1 - (R0 + (1 - 1))Tr ) R = (0 + nnt - 1)

E \* diffuse; = true;

-: efl + refr)) && (depth is HARDIIII

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it property ff; radiance = SampleLight( %rand, I e.x + radiance.y + radiance.z) > 0) %

v = true; t brdfPdf = EvaluateDiffuse( L, N.) = Pour st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* 000

andom walk - done properly, closely following a /ive)

; t3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, Dpf pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

### 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.

![](_page_65_Picture_19.jpeg)

### Reflections

### Screen Space Reflections

![](_page_66_Picture_3.jpeg)

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

andom walk - done properly, closely following /ive)

; t3 Brdf = SampleDiffuse( diffuse, N, r1, r2, RR, Rod urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

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

![](_page_66_Picture_8.jpeg)

## Reflections

Screen Space Reflections

iles 6 (depth : Places

= = inside / 1 nt = nt / nc, dds os2t = 1.0f = nnt 0, N ); 3)

st a = nt - nc, b - : st Tr = 1 - (R0 + (1 Tr) R = (D \* nnt - N

E = diffuse = true;

-:**fl + refr**)) && (depth

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

AXDEPTH)

survive = SurvivalProbabil estimation - doing it pro ff; radiance = SampleLight( %r e.x + radiance.y + radiance.z) > 0)

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

andom walk - done properly, closely following /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, sourvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

### "Efficient GPU Screen-Space Ray Tracing", McGuire & Mara, 2014

![](_page_67_Picture_15.jpeg)

![](_page_67_Picture_16.jpeg)

![](_page_68_Picture_0.jpeg)

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

![](_page_70_Picture_10.jpeg)

![](_page_70_Picture_11.jpeg)

#### st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

### v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

## Famous Last Words

tic: k (depth < 100⊂

= inside / 1 it = nt / nc, dda os2t = 1.0f = nnt D, N ); B)

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

= diffuse; = true;

efl + refr)) && (depth is HANDISTI

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

AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; radiance = SampleLight( &rand, I do the e.x + radiance.y + radiance.z) > 0) #100

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

andom walk - done properly, closely following : /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, local pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

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

![](_page_71_Picture_24.jpeg)
# Famous Last Words

tic: ⊾ (depth < 100

= = inside / 1 it = nt / nc, dde ss2t = 1.0f = nnt ), N ); 3)

st a = nt - nc, b = nt - ncst Tr = 1 - (R0 + (1 - 0) Tr) R = (D \* nnt - N - 0)

E \* diffuse; = true;

efl + refr)) && (depth & HANDIIII

D, N ); ~efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; radiance = SampleLight( &rand, I, II, e.x + radiance.y + radiance.r) > 0) Mile

v = true; st brdfPdf = EvaluateDiffuse( L, N.) \* Pour st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following: /ive)

; ot3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, local prvive; pdf; a = E \* brdf \* (dot( N, R ) / pdf); sion = true;

# 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

tic: ⊾ (depth ∈ 100

= = inside / 1 it = nt / nc, dde ss2t = 1.0f = nnt -5, N ); 3)

st a = nt - nc, b = nt - ncst Tr = 1 - (R0 + (1 - 0) Tr) R = (D \* nnt - N - 0)

E = diffuse; = true;

-:fl + refr)) && (depth is HANDIII

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse( L, N) \* Prost3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \*

andom walk - done properly, closely following /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, UR, Upd prvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

# 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

tic: k (depth < NAS

= inside / 1 it = nt / nc, dde ss2t = 1.0f = ont 5, N ); 3)

st a = nt - nc, b + nt st Tr = 1 - (80 + (1 Tr) R = (0 \* nnt - N

= diffuse; = true;

: :fl + refr)) && (depth k HADDI

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

#### AXDEPTH)

survive = SurvivalProbability( difference estimation - doing it properly if; adiance = SampleLight( &rand, I, I, e.x + radiance.y + radiance.z) > 0) #8

v = true; t brdfPdf = EvaluateDiffuse( L, N ) \* Pur st3 factor = diffuse \* INVPI; st weight = Mis2( directPdf, brdfPdf ); st cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf)

andom walk - done properly, closely following: /ive)

; st3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, M // EOF urvive; pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true:

## Example shader:

#version 330

// shader input
in vec2 P;
in vec2 uv;
uniform sampler2D pixels;

// 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;

// fragment position in screen space
// interpolated texture coordinates
// input texture (1st pass render target)

# 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





### st3 factor = diffuse \* INVPI: at weight = Mis2( directPdf, brdfPdf ) andom walk - done properly, closely follow vive)

at3 brdf = SampleDiffuse( diffuse, N, r1, r2, LR irvive; pdf; = E \* brdf \* (dot( N, R ) / pdf); sion = true:

), N ); efl \* E \* diffuse; = true;

#### AXDEPTH)

survive = SurvivalProbability( diff. if; adiance = SampleLight( &rand, I, .... e.x + radiance.y + radiance.z) > 0)

## v = true; at brdfPdf = EvaluateDiffuse( L, N )

at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* tic: ⊾ (depth < 144

: = inside / L it = nt / nc, dde os2t = 1.0f 0, N ); 3)

st a = nt - nc, b - nt + st Tr = 1 - (R0 + (1 fr) R = (D \* nnt - N \*

= diffuse = true;

≕ efl + refr)) && (depth k NADIIII

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

AXDEPTH)

survive = SurvivalProbability difference estimation - doing it properly if; radiance = SampleLight( %rand, I, I) e.x + radiance.y + radiance.r) > 0)

v = true; at brdfPdf = EvaluateDiffuse( L, N ) \* Paurola st3 factor = diffuse \* INVPI; at weight = Mis2( directPdf, brdfPdf ); at cosThetaOut = dot( N, L ); E \* ((weight \* cosThetaOut) / directPdf) \* ObjectPdf

andom walk - done properly, closely following a /ive)

; pt3 brdf = SampleDiffuse( diffuse, N, r1, r2, NR, NS, pdf; n = E \* brdf \* (dot( N, R ) / pdf); sion = true;

# INFOGR – Computer Graphics

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

# END of "Post-processing"

next lecture: "Stochastic Methods"

