tica (depth < NAX

: = inside / : ht = nt / nc, dr bs2t = 1.0f 5, N); 3)

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

= diffuse = true;

efl + refr)) && (depth x HACOL

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N,) Pare 1 st3 factor = diffuse * INVPI; at weight = Mis2(directPdf, brdfPdf); at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * (000)

andom walk - done properly, closely following /ive)

st3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Not urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;

INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

Lecture 14: "Post-processing"

Welcome!



Today's Agenda:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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





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 - NAA

>, N); ref1 * E * diff(= true;

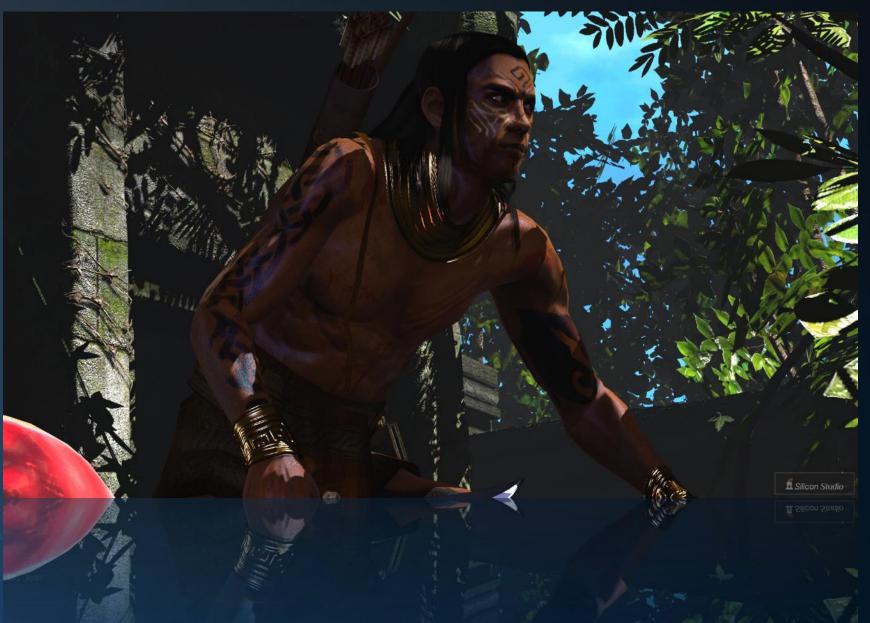
AXDEPTH)

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

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:





ref1 * E * diffuse; = true;

AXDEPTH)

e.x + radiance.y + radiance.z) > 0) #8

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

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





Introduction

nice K (depth < 1923

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

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

= diffuse; = true;

≕ efl + refr)) 88 (depth k HANDIII

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

AXDEPTH)

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

v = true; st brdfPdf = EvaluateDiffuse(L, N) Probt3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

indom walk - done properly, closely following
/ive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, bp3 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 open space
- Anti-aliasing

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

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



6

INFOGR – Lecture 14 – "Post-processing"

Camera Effects

tice k (depth < 10.5

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

at a = nt - nc, b - nt at Tr = 1 - (R0 + Tr) R = (D * nnt - 4

= diffuse; = true;

efl + refr)) && (depth is HANDIII)

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Puur 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, R, boo pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Purpose: simulating camera / sensor behavior

Bright lights:

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





INFOGR – Lecture 14 – "Post-processing"

Camera Effects

fice € (depth < 10.5

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

st a = nt - nc, b - nt st Tr = 1 - (80 + (1 - 1 Fr) R = (0 * nnt - 8

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, M) 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)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, R, b) 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 = 1.0f - nnt), N); 3)

= diffuse; = true;

= efl + refr)) && (depth k HANDETTI

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

AXDEPTH)

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)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, bod prvive; 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 **i (depth** < 10.5

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

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

= diffuse = true;

efl + refr)) && (depth & MADITIE

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

AXDEPTH)

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

v = true;

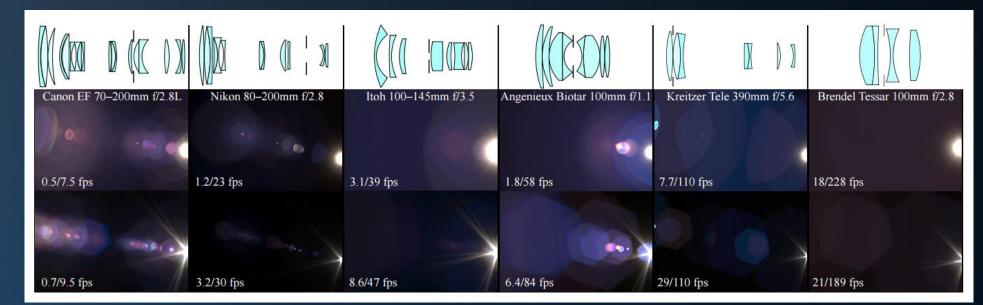
st brdfPdf = EvaluateDiffuse(L, N) * Pu st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf)

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

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, bp3 pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Lens Flares

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







Lens Flares

tic: ⊾(depth < R

= inside / 1 nt = nt / nc, de os2t = 1.0f - ... o, N); ð)

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

= diffuse
= true;

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

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; at brdfPdf = EvaluateDiffuse(L, N) * Pour 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, 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 < NA⊂

= inside / 1 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

E ⁼ 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) * Pourse 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)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, ppd 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); refl * E * diffuse; = true;

AXDEPTH)

survive = SurvivalProbability(different estimation - doing it properly if; radiance = SampleLight(%rand, I, & 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;







tice ⊾ (depth < 1920

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

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

= diffuse = true;

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

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

AXDEPTH)

survive = SurvivalProbability(different estimation - doing it property 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.



INFOGR – Lecture 14 – "Post-processing"

Camera Effects

tica € (depth < 10.5

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

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

= diffuse; = true;

: efl + refr)) && (depth & HADDIII)

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, bpd prvive; 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

201

10

9

0

-



INFOGR – Lecture 14 – "Post-processing"

Camera Effects

tica ⊾ (depth < 10.5

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

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

E ⁼ diffuse = true;

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

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Pour 1 st3 factor = diffuse * INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) * 0000

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

; t33 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, hpd urvive; 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 / 1 it = nt / nc, ddo ss2t = 1.0f - nnt 3, N); 3)

at a = nt - nc, b + nt + + at Tr = 1 - (R0 + (1 - 10 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, M) =x + radiance.y + radiance.r) > 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) * 0000

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

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, RR, Dpd 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 - nf st Tr = 1 - (80 + (1 fr) R = (0 * 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)

andom walk - done properly, closely following -/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:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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





tic: (depth (1920)

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

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

= diffuse; = true;

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

D, N); refl * 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; at brdfPdf = EvaluateDiffuse(L, N) * Pour 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, NR, so 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 < 1920)

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

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

= diffuse; = true;

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

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

AXDEPTH)

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

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

; st3 brdf = SampleDiffuse(diffuse, N, r1, r2, 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







Le: (depth < NASS

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

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

v = true;

it brdfPdf = EvaluateDiffuse(L, N.) Pours it3 factor = diffuse * INVPI; it weight = Mis2(directPdf, brdfPdf); it 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 ht = nt / nc, dde os2t = 1.0f - nnt D, N); B)

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

= diffuse; = true;

• •fl + cefc)) 88 /death / Havi

), 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) * Paulos 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 : /ive)

; st3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, SS 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:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

- 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

11c) 4 (depth - 114)

= = 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 = (0 * nnt - 1)

= diffuse; = true;

efl + refr)) && (depth k HANDIIII

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

AXDEPTH)

survive = SurvivalProbability different estimation - doing it property if; radiance = SampleLight(&rand I .x + radiance.y + radiance.r) 00 %

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

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

, t3 brdf = SampleDiffuse(diffuse, N, r1, r2, R, second prvive; 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)*

*: <u>https://priceonomics.com/why-every-movie-looks-sort-of-orange-and-blue</u>



Color Grading

tica ≰ (depth ≤ 100⊂

: = 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;

-: :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; st brdfPdf = EvaluateDiffuse(L 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, Doff urvive; 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:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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





tica k (depth k 1933

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

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

= diffuse; = true;

⊂ efl + refr)) && (depth < HANDETT

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

AXDEPTH)

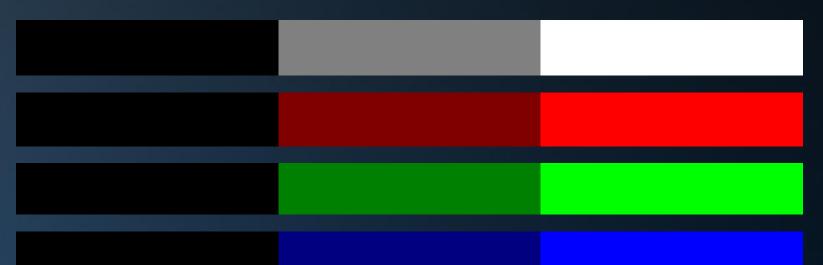
survive = SurvivalProbability(difference estimation - doing it properly if; radiance = SampleLight(%rand, I e.x + radiance.y + radiance.z) > 0) %

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) * 000

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

; t3 brdf = SampleDiffuse(diffuse, N, r1, r2, RR, Nor urvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

Concept



Monitors respond in a non-linear fashion to input.



tic: ⊾ (depth < 100

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

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

= 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, ... e.x + radiance.y + radiance.z) > 0) %

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)

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

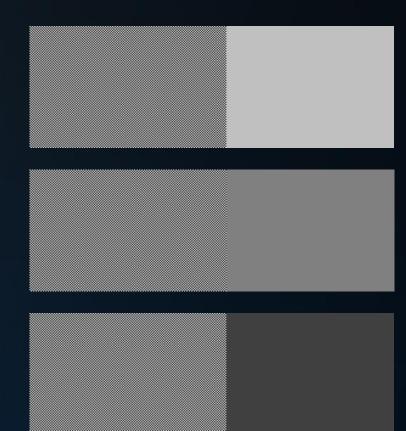
Concept

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

Example for
$$\gamma = 2$$
: $a = \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\} \rightarrow I = \{0, \frac{1}{16}, \frac{1}{4}, \frac{9}{16}, 1\}$

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

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





11c) 4 (depth - 114)

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

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

= diffuse = true;

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

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) * ***

andom walk - done properly, closely following vive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR, LCH urvive; 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 $I = a'^{\gamma} = (a^{\frac{1}{\gamma}})^{\gamma} = a$.





tica ≰ (depth < 10.5

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

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

= diffuse; = true;

efl + refr)) && (depth & HADDEFF

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; tbrdfPdf = 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 $\gamma \approx 2$

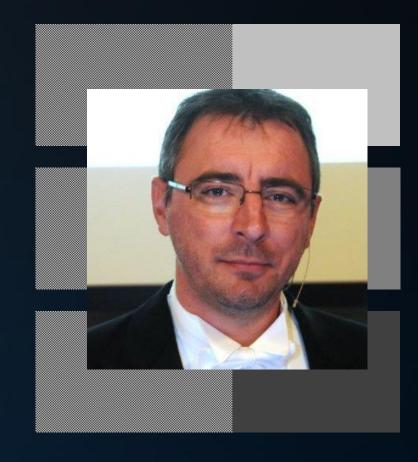
Apart from 'gamma correcting' our output, we also need to pay attention to our input.

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^{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:

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





tic: k (depth <)∪...

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

at a = nt - nc, b - nt - at Tr = 1 - (R0 + -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) * 10

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

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, Spd 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...





Today's Agenda:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

- 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'. 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

at 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

Depth of Field

tice ≰ (depth < 10.5

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

st $a = nt - hc_{2} b + nt - hc_{3}$ st Tr = 1 - (R0 + (1 - R))(r) R = (0 + nnt - R - 1)

= 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 = .x + radiance.y + radiance.r) = 0.000

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)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, brd 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'.



Depth of Field

tic: k (depth < NAS

= 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 - nc)

= diffuse; = true;

efl + refr)) && (depth & 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) #8

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, NR, lost 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.



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.



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

53



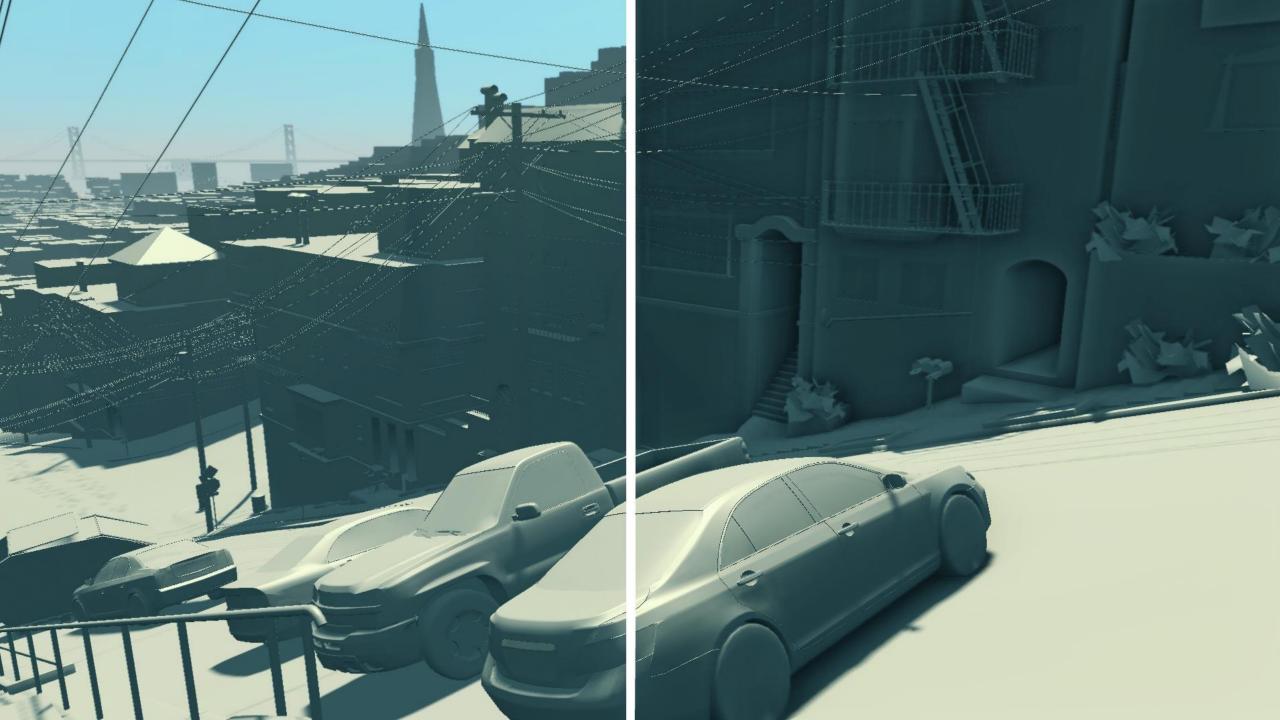
Today's Agenda:

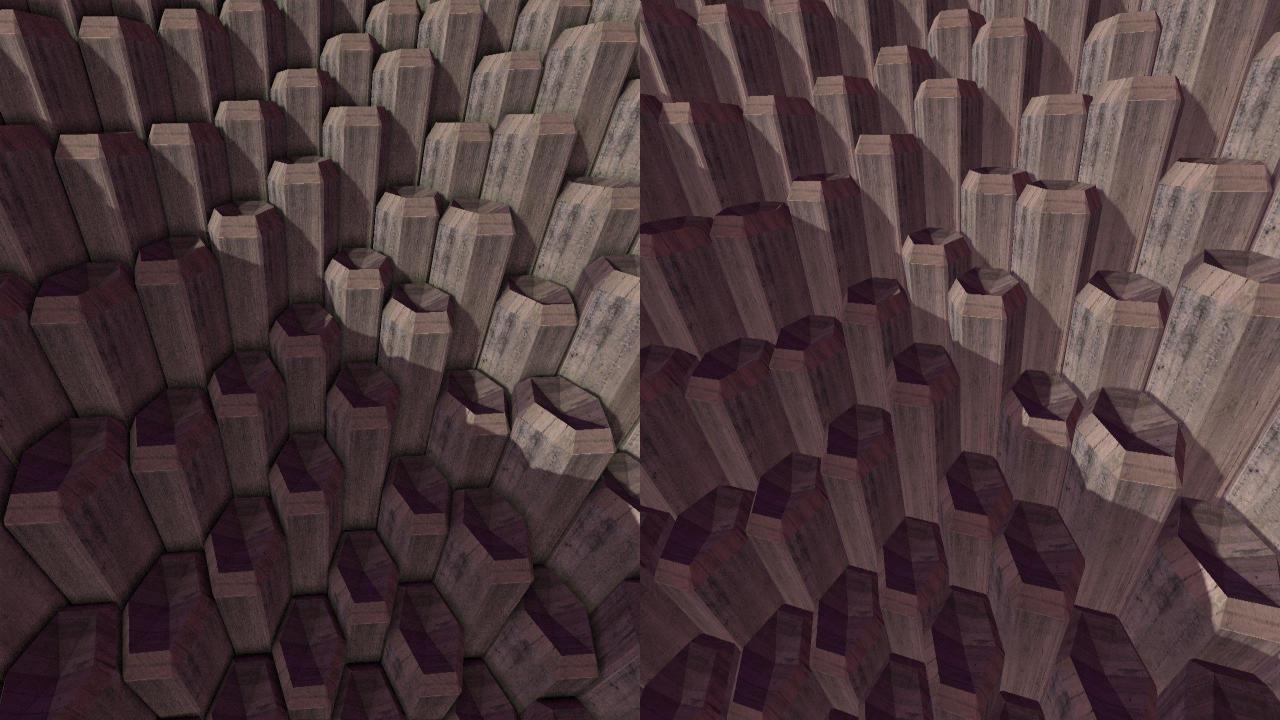
-), N); efl * E * diffuse; = true;
- AXDEPTH)
- survive = SurvivalProbability(diff. if; adiance = SampleLight(&rand, I, e.x + radiance.y + radiance.z) > 0)
- v = true; at brdfPdf = EvaluateDiffuse(L, N) st3 factor = diffuse * INVPI: st weight = Mis2(directPdf, brdfPdf) at cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) *
- andom walk done properly, closely follow vive)
- at3 brdf = SampleDiffuse(diffuse, N, r1, r2, LR irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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









WAVE 1 TAKE DOWN ALL HOSTILES HOSTILES REMAINING 5

1071 MHZ

[SPACE] CLIMB

10M

STICKY NOISEMAKER





tice ⊾ (depth < 1000

= = 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 & NADIII

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Ptury at3 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, RR, lost urvive; 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.



tica **↓ (depth** < 10.5

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

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 k HADIIII

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

AXDEPTH)

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

v = true; at brdfPdf = EvaluateDiffuse(L, N.) * Promise 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, 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.



tic: ⊾ (depth < 155

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

st $a = nt - nc_{1} b - nt - at Tr = 1 - (R0 + (1 - 1))$ r) R = (0 + nt - 1)

= 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, M) e.x + radiance.y + radiance.r) > 0) %

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

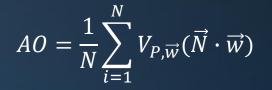
andom walk - done properly, closely following vive)

; pt3 brdf = SampleDiffuse(diffuse, N, r1, r2, NR, brd 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.



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



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



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

AXDEPTH)

urvive = Survival adiance = SampleLight

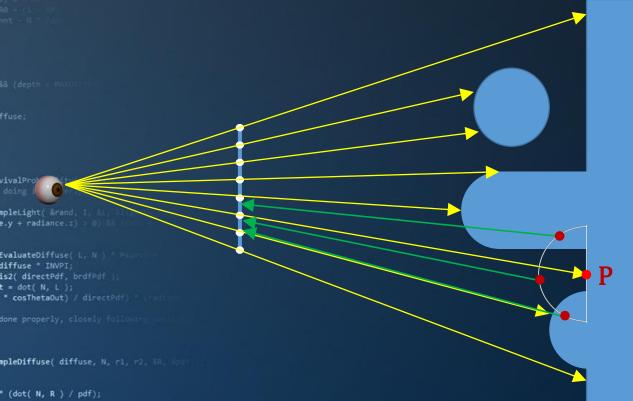
v = true at3 factor = diffuse st weight = Mis2(directPdf, brdfPd) at cosThetaOut = dot(N, L) E * ((weight * cosThetaOut) / directPd

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.



- 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*.





tica ≰ (depth < 10.5

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

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

= diffuse = true;

: :fl + refr)) && (depth & HARDI);

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.) = Pours) st3 factor = diffuse = INVPI; st weight = Mis2(directPdf, brdfPdf); st cosThetaOut = dot(N, L); E * ((weight * cosThetaOut) / directPdf) = 0000

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.



Today's Agenda:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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





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 k HAADIIII

D, N); refl * 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 -/ive)

; t3 brdf = SampleDiffuse(diffuse, N, F1, F2, NR, Np); 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.



Reflections

Screen Space Reflections



w = true; at brdfPdf = EvaluateDiffuse(L, N) at weight = Mis2(directPdf, brdfPdf)) 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, Fl, F2, SR, Spect prvive; pdf; n = E * brdf * (dot(N, R) / pdf); Sion = true:



Reflections

Screen Space Reflections

tice (depth o NACE)

= = inside / 1 nt = nt / nc, ddd 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)

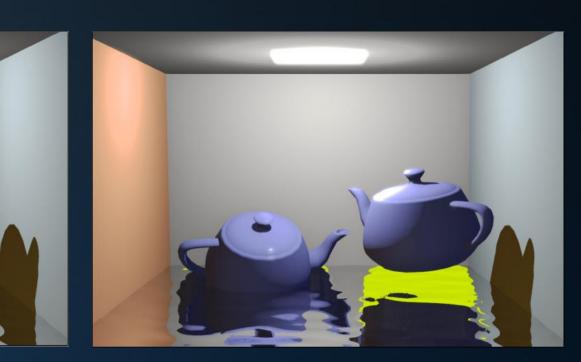
survive = SurvivalProbabil estimation - doing it pro ff; radiance = SampleLight(%r e.x + radiance.y + radiance.z) > 0)

w = true; at brdfPdf = EvaluateDiffuse(L, N) (bt3 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, st pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

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







Killzone Shadowfall

Today's Agenda:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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





tici k (depth < 10.55

: = inside / | it = nt / nc, dde os2t = 1.0f = nn; D, N); B)

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

= diffuse; = true;

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

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

AXDEPTH)

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

v = true; st brdfPdf = EvaluateDiffuse(L, N) * Puers 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, F1, F2, UR, prvive; 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



tic: ⊾ (depth < 155

= = 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)

= diffuse; = true;

efl + refr)) && (depth is HANDIC)

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

w = 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, lock pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;

Experimenting

Use the post-processing functionality in the P3 template.

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.



tic: ⊾ (depth < 100

= = inside / 1 it = nt / nc, dde ss2t = 1.0f - nnt 7 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 MANDIE

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) * Prom 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, Upd prvive; pdf; n = E * brdf * (dot(N, R) / pdf); sion = true;

Experimenting

Use the post-processing functionality in the P3 template.

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.



efl + refr)) && (depth

), N); refl * E * diffuse;

AXDEPTH)

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

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

andom walk - done properly, closely follo vive)

// EOF at3 brdf = SampleDiffuse(diffuse, N, r1, r2, UR rvive; pdf; i = 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:

-), 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) 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 irvive; pdf; = E * brdf * (dot(N, R) / pdf); sion = true:

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





tice (depth c NA

z = inside / L it = nt / nc, dd os2t = 1.0f o, N); 8)

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

E ⁼ diffuse = true;

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

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

AXDEPTH)

v = true; at brdfPdf = EvaluateDiffuse(L, N) * Purple 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, UR, st pdf; n = E * brdf * (dot(N, R) / pdf); sion = true:

INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

END OF lecture 14: "Post-processing"

Next lecture: "Grand Recap"

