

# Uncharted 2: HDR Lighting

---

John Hable  
Naughty Dog



# Agenda

---

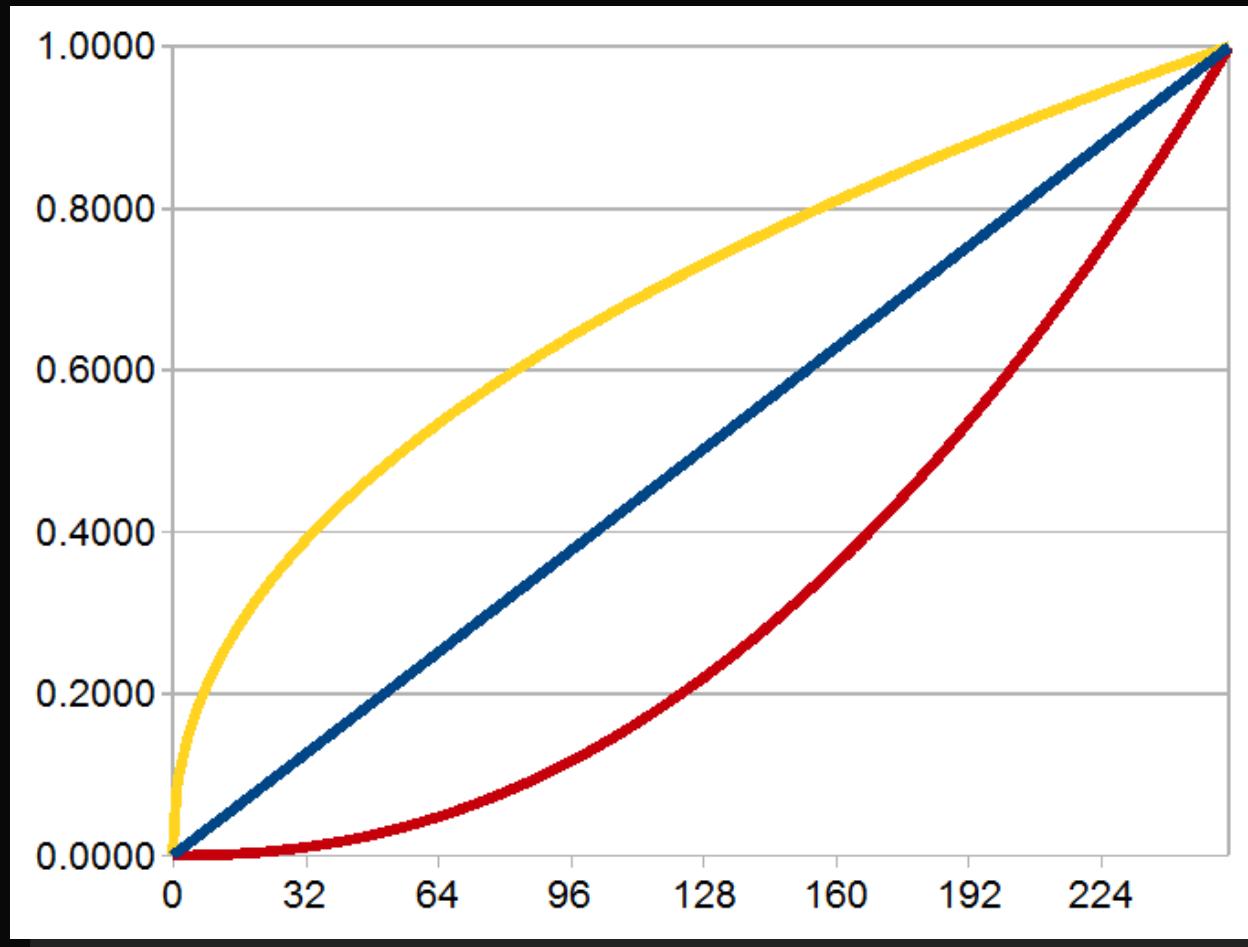
1. Gamma/Linear-Space Lighting
2. Filmic Tonemapping
3. SSAO
4. Rendering Architecture/How it all fits together.

# Skeletons in the Closet

---

- Used to work for EA Black Box (Vancouver)
  - Ucap Project
  - I.e. Tiger Woods's Face
- Also used to work for EA Los Angeles
  - LMNO (Spielberg Project)
  - Not PQRS (Boom Blox)
- Now at Naughty Dog
  - Uncharted 2

# Part 1: Gamma!



# Gamma

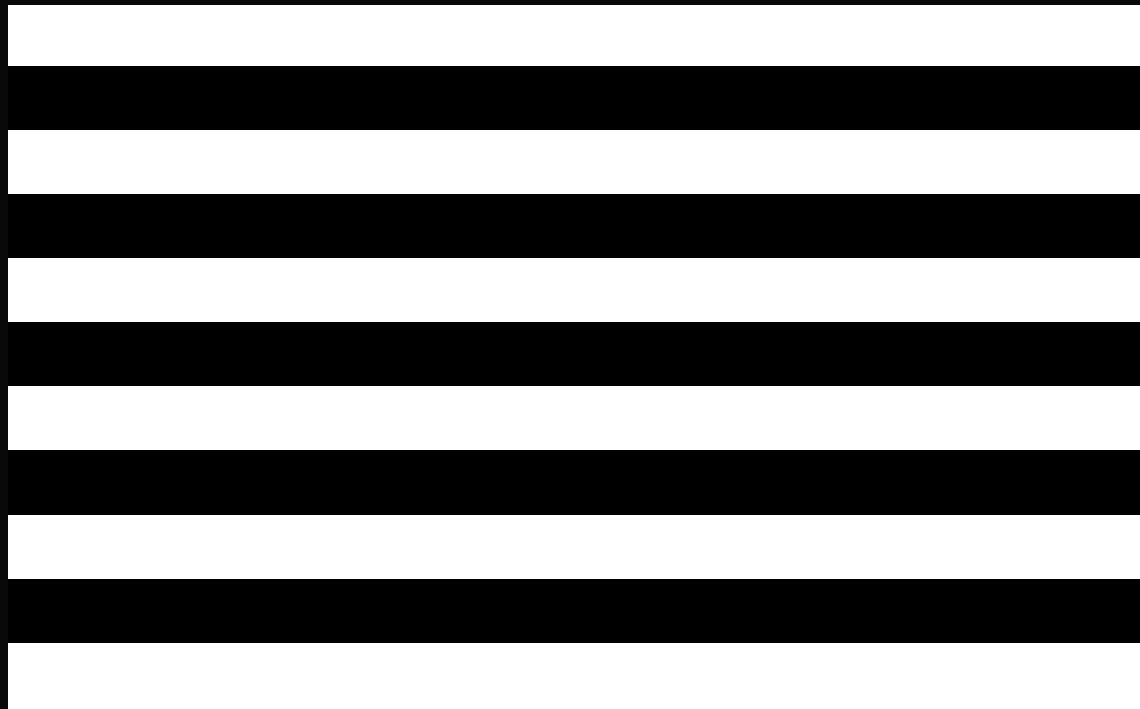
---

- Mostly solved in Film Industry after lots of kicking and screaming
- Never heard about it in college
- George Borshukov taught it to me back at EA
  - Matrix Sequels
- Issue is getting more traction now
  - Eugene d'Eon's chapter in GPU Gems 3

# Demo

---

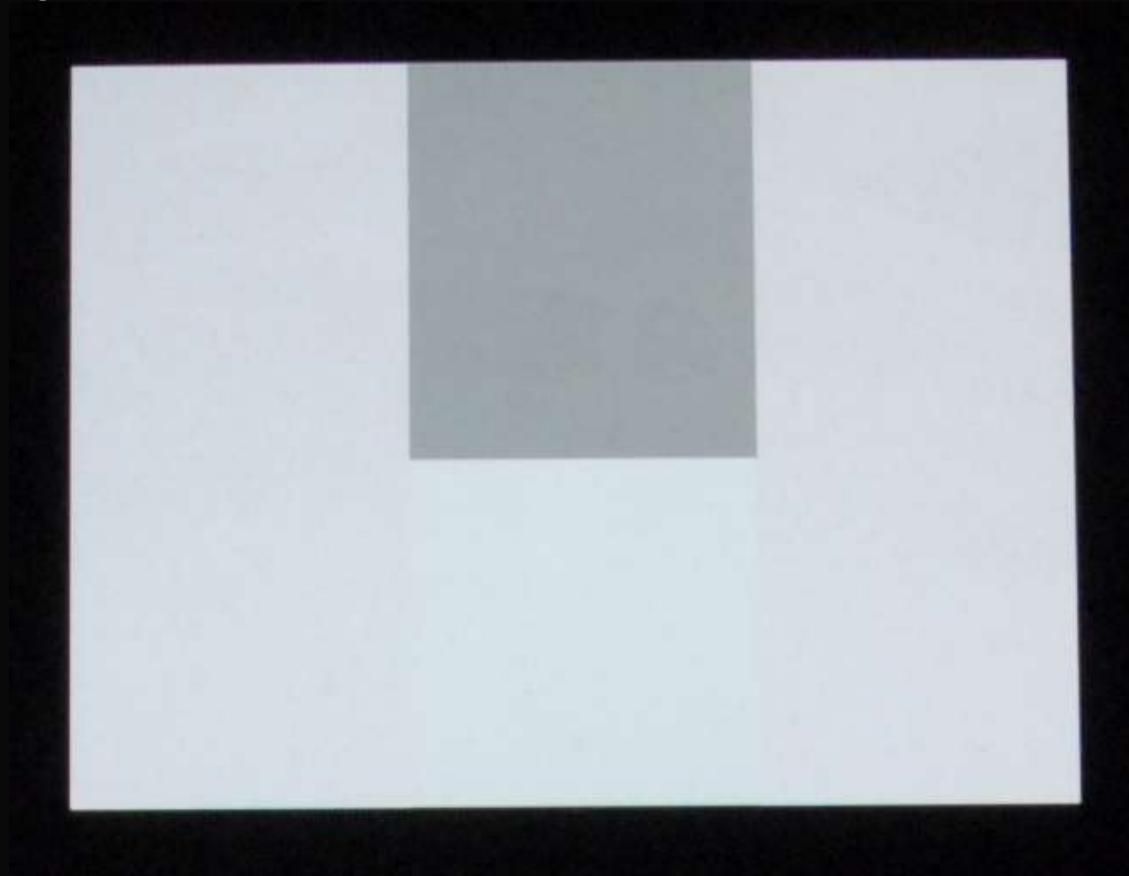
- What is halfway between white and black
- Suppose we show alternating light/dark lines.
- If we squint, we get half the linear intensity.



# Demo

---

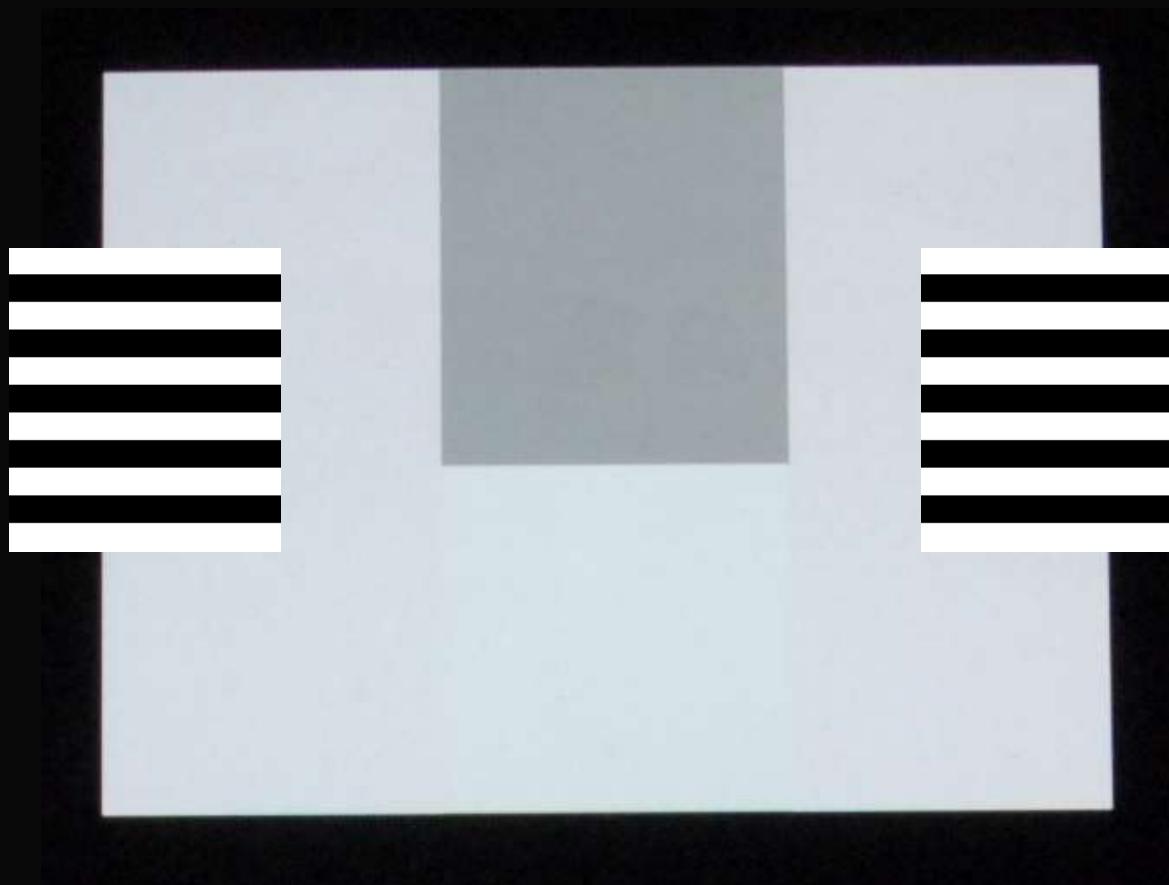
- Test image with 4 rectangles.
- Real image of a computer screen shot from my camera.



# Demo

---

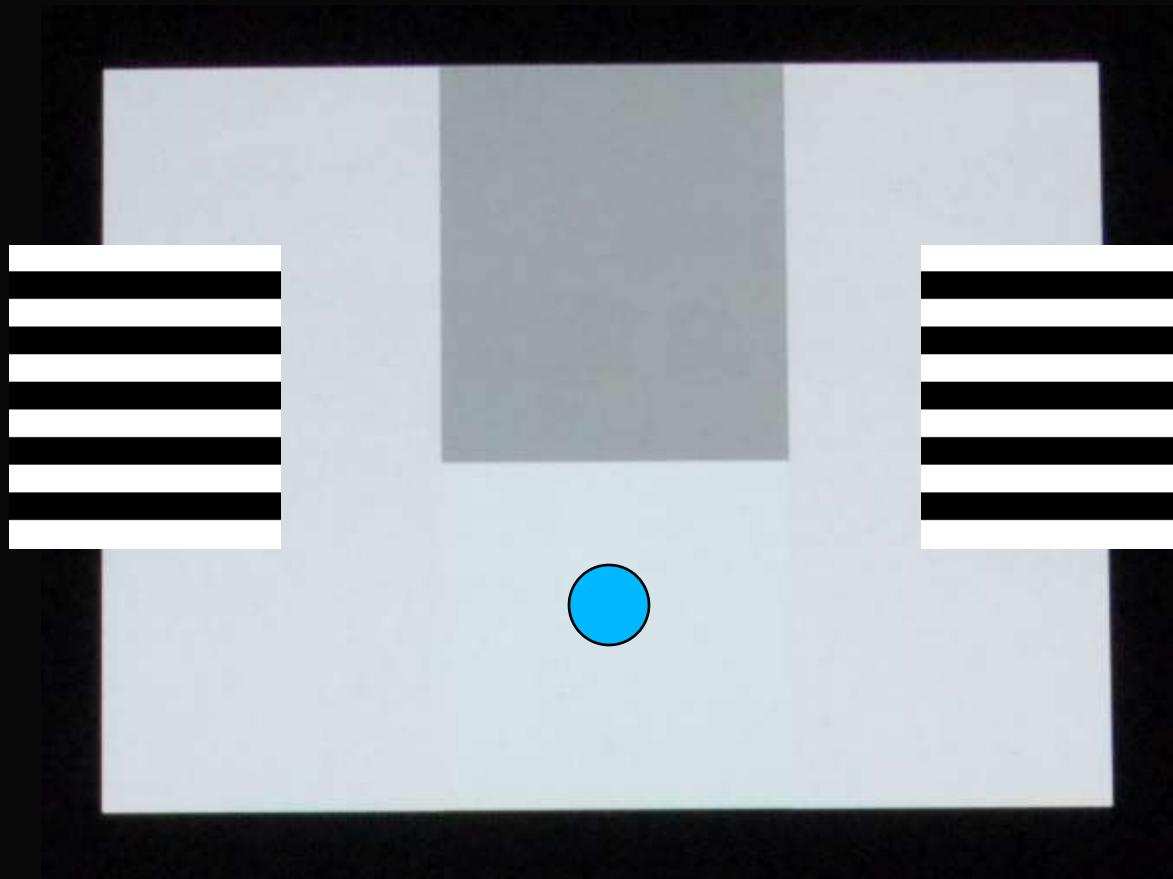
- Alternating lines of black and white (0 and 255).



# Demo

---

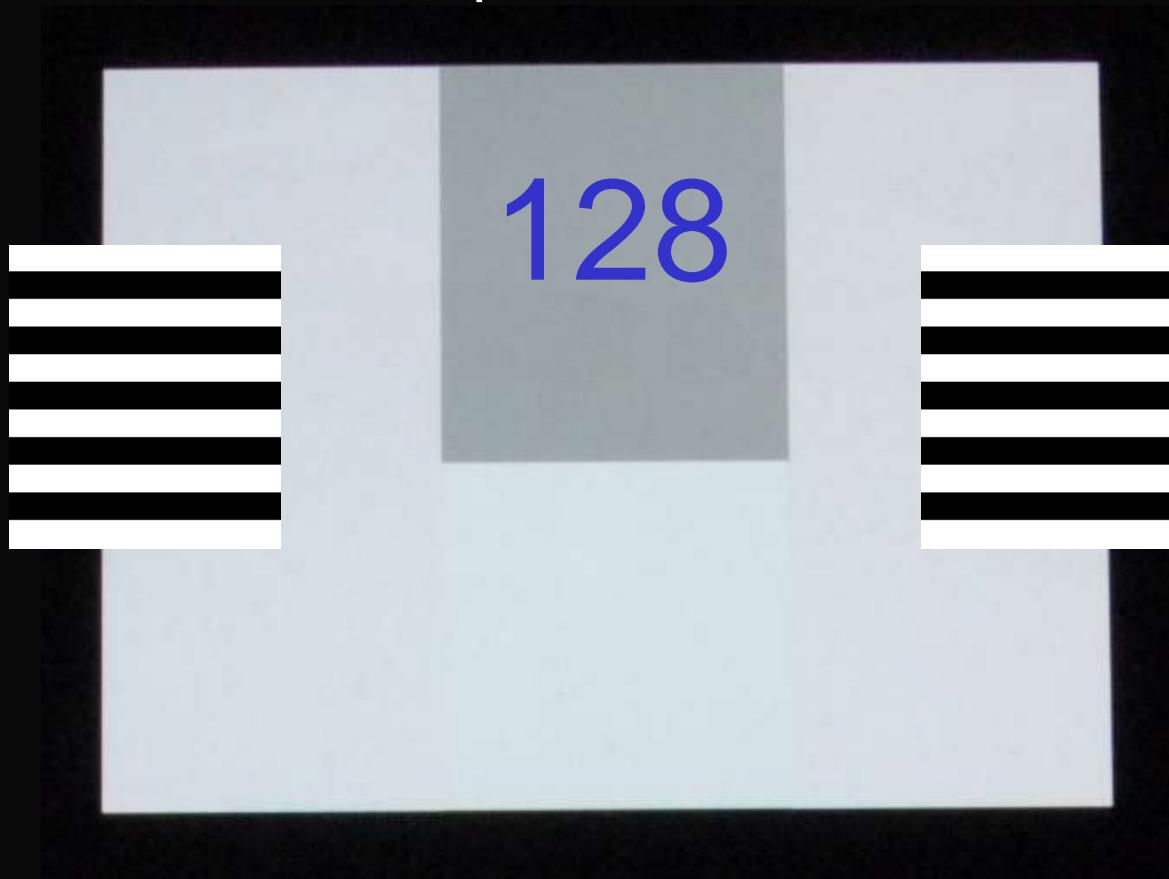
- So if the left and right are alternating lines of 0 and 255, then what color is the rectangle with the blue dot?



# Demo

---

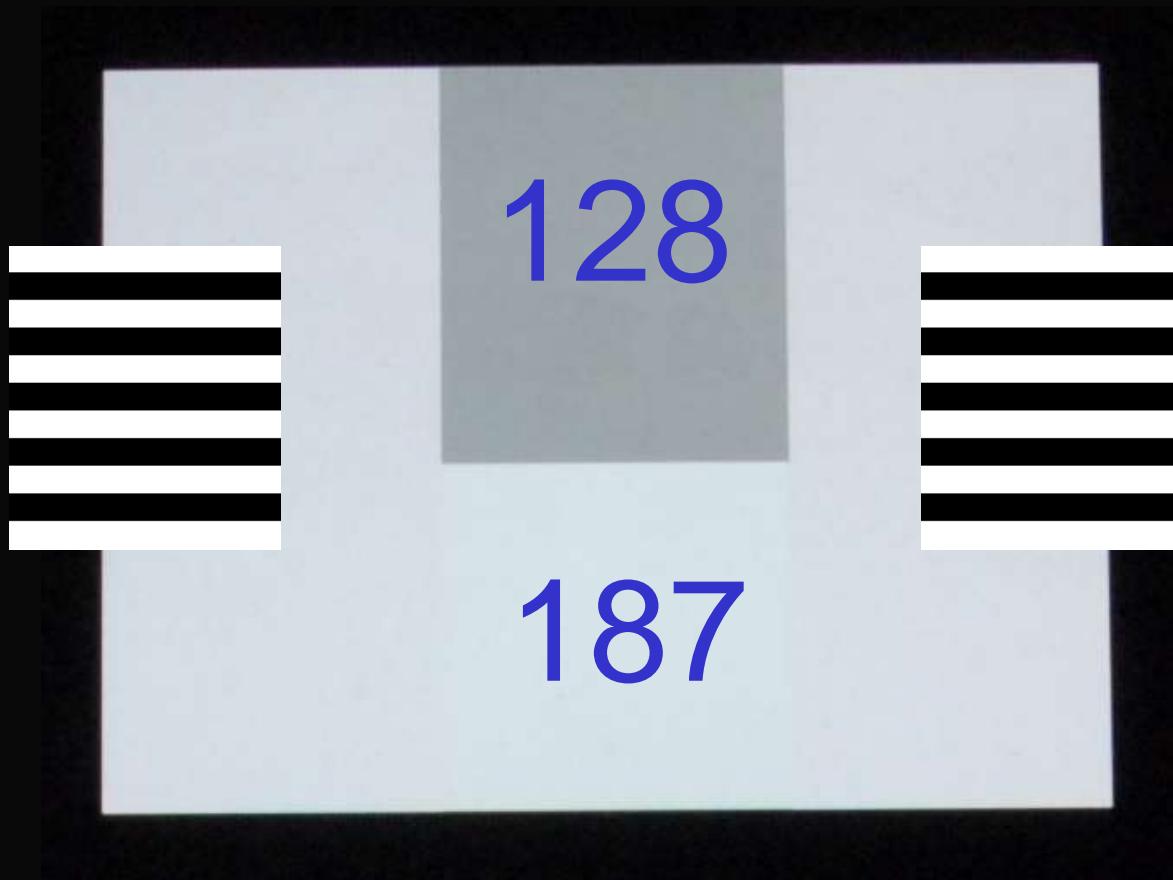
- Common wrong answer: 127/128.
- That's the box on the top.



# Demo

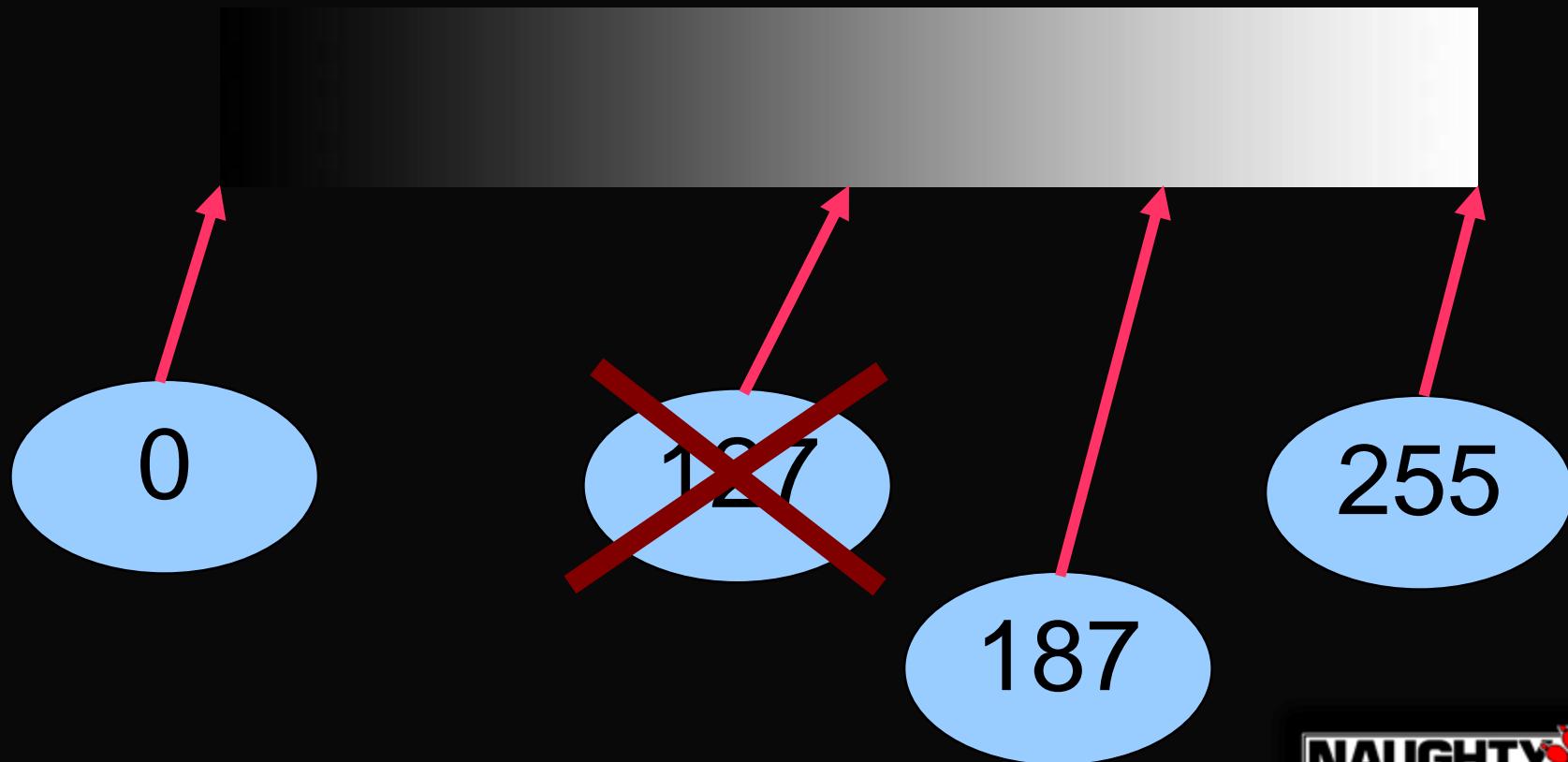
---

- Correct answer is 187.
- WTF?



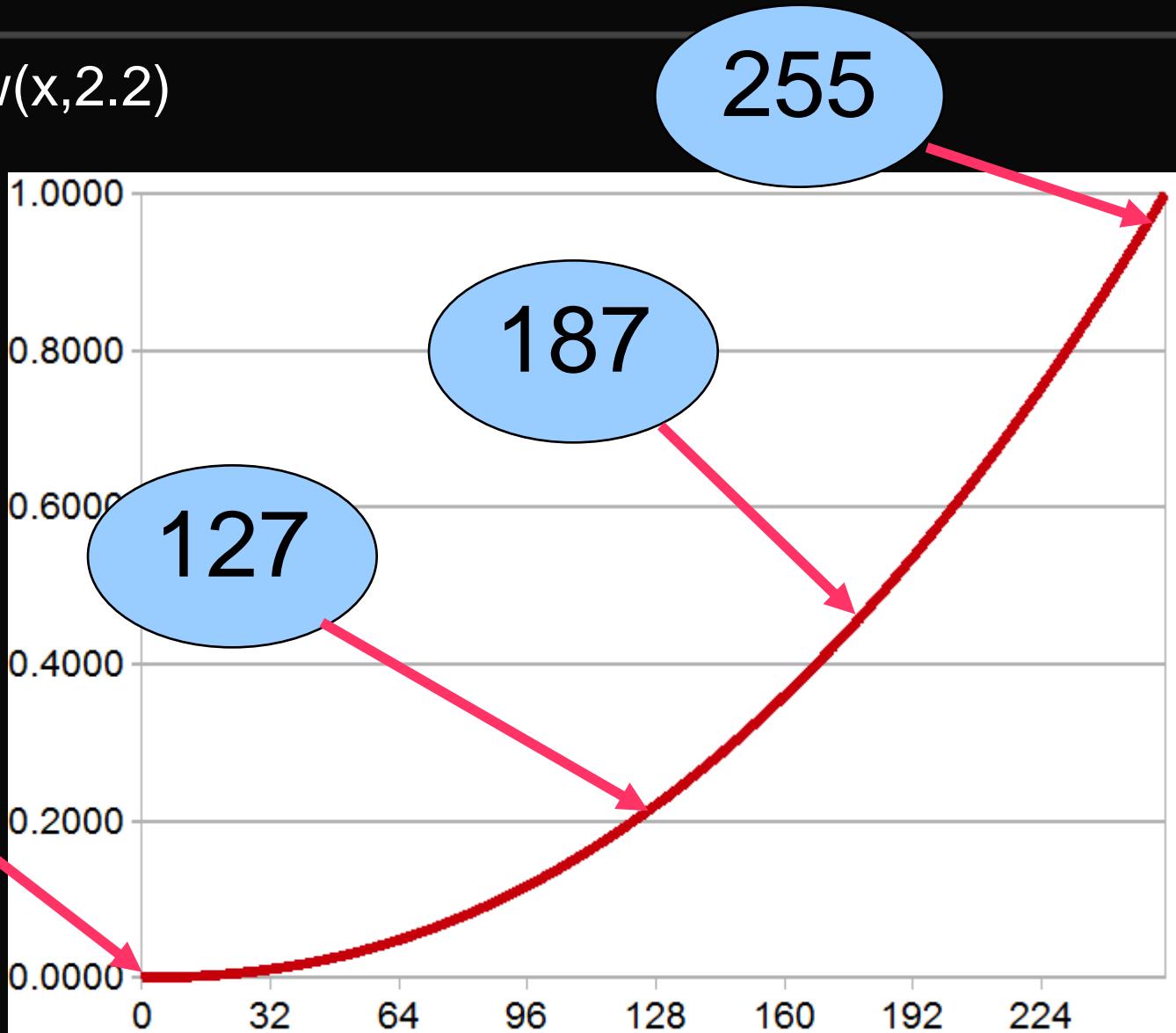
# Color Ramp

- You have seen this before.



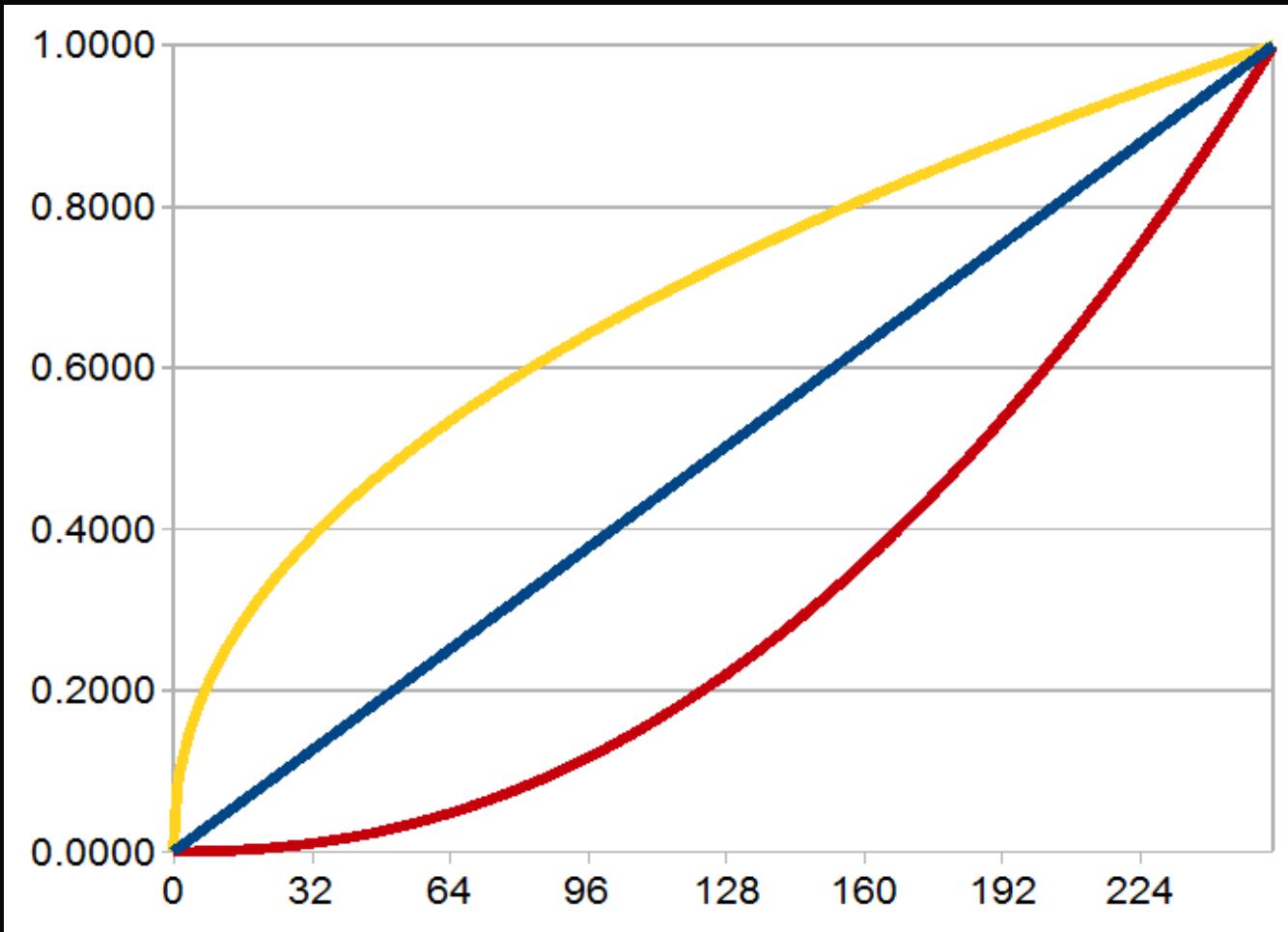
# Gamma Lines

- $F(x) = \text{pow}(x, 2.2)$



# What is Gamma

- Gamma of 0.45, 1, 2.2
- Our monitors are the red one



# We all love mink...

---

- $F(x) = x$



# We all love mink...

---

- $F(x) = \text{pow}(x, 0.45)$       note:  $.45 \approx 1/2.2$



# We all love mink...

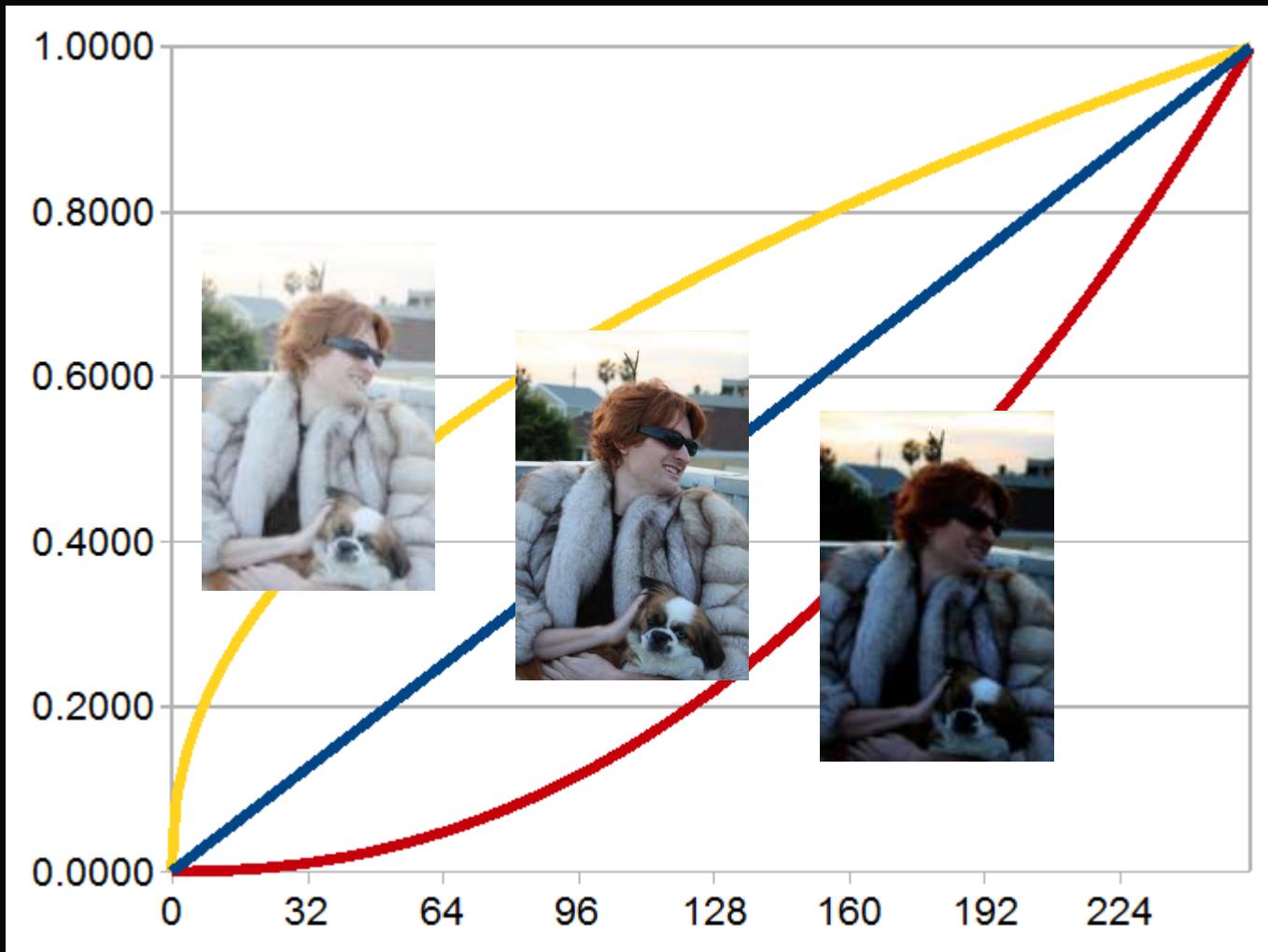
---

- $F(x) = \text{pow}(x, 2.2)$



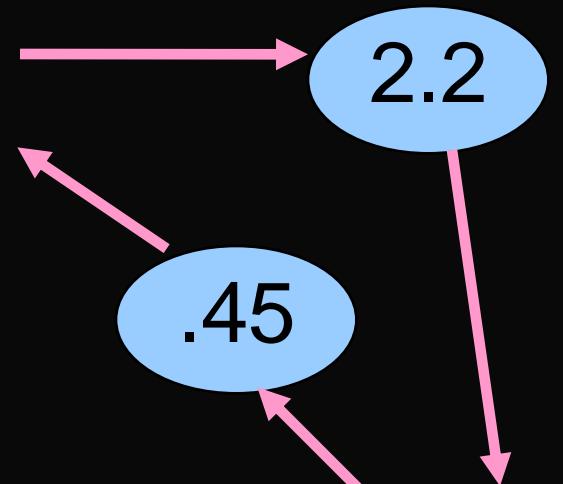
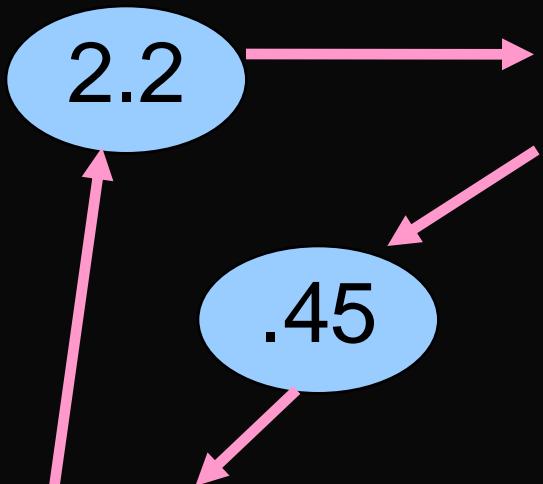
# What is Gamma

- Get to know these curves...



# What is Gamma

- Back and forth between curves.



# How bad is it?

---

- Your monitor has a gamma of about 2.2
- So if you output the left image to the framebuffer, it will actually look like the one right.



2.2



# Let's build a Camera...

- A linear camera.

Actual Light

1.0



Hard Drive

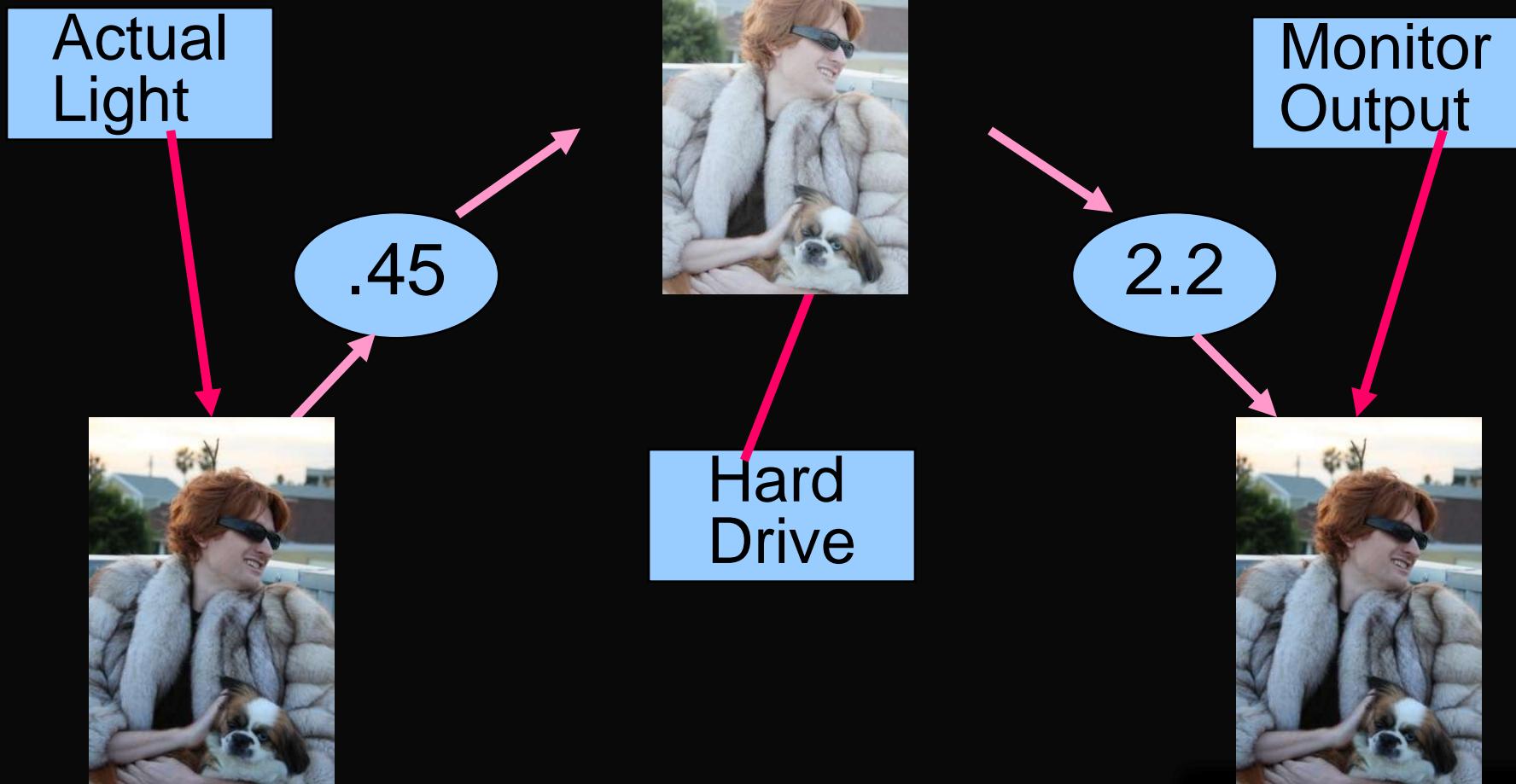
Monitor Output

2.2



# Let's build a Camera...

- Any consumer camera.



# How to fix this?

---

- Your screen displays a gamma of 2.2
- This is stored on your hard drive.
  - You never see this image, but it's on your hard drive.

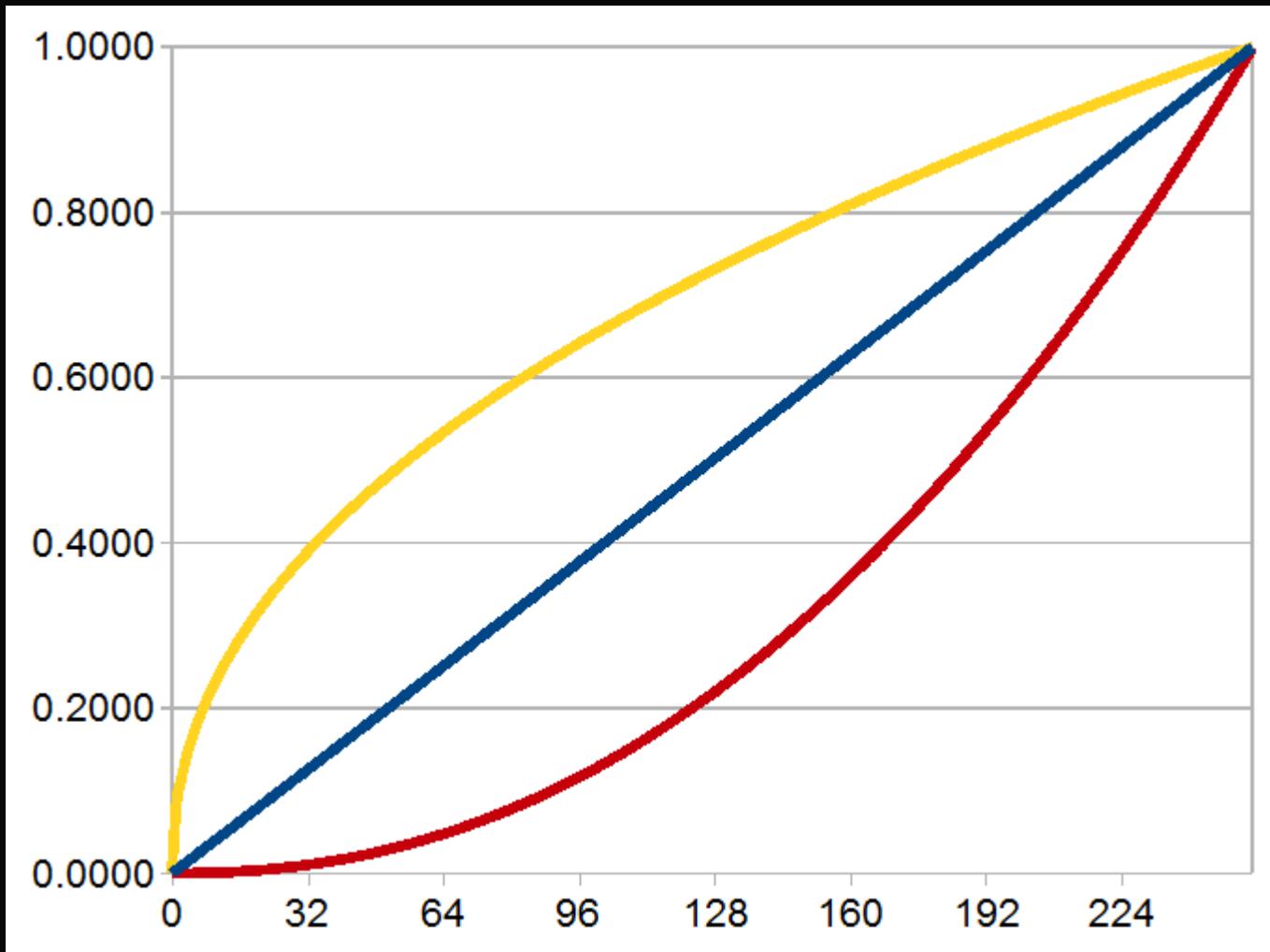


2.2



# What is Gamma

- Curves again.

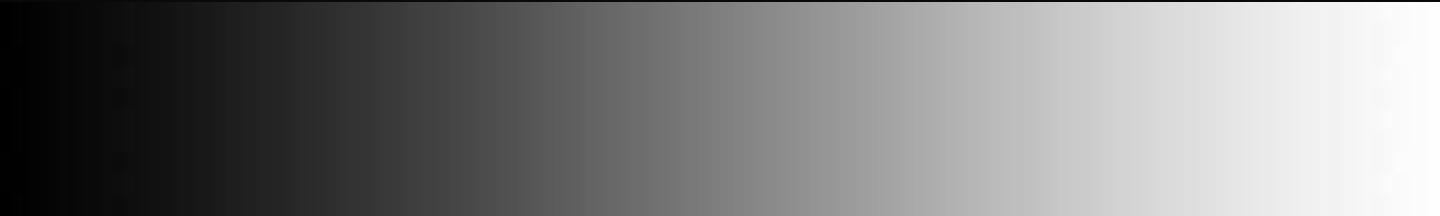


# What is Gamma

---

Q) Why not just store as linear?

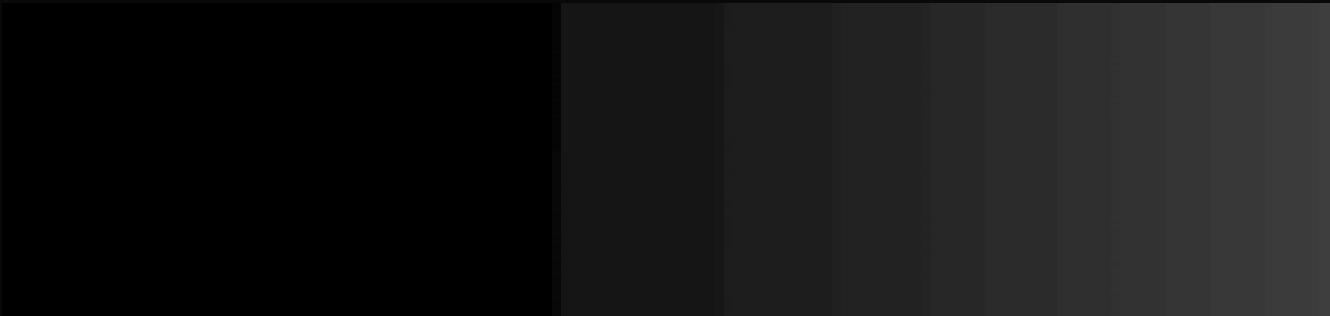
A) Our eyes perceive more data in the blacks.



# How bad is it?

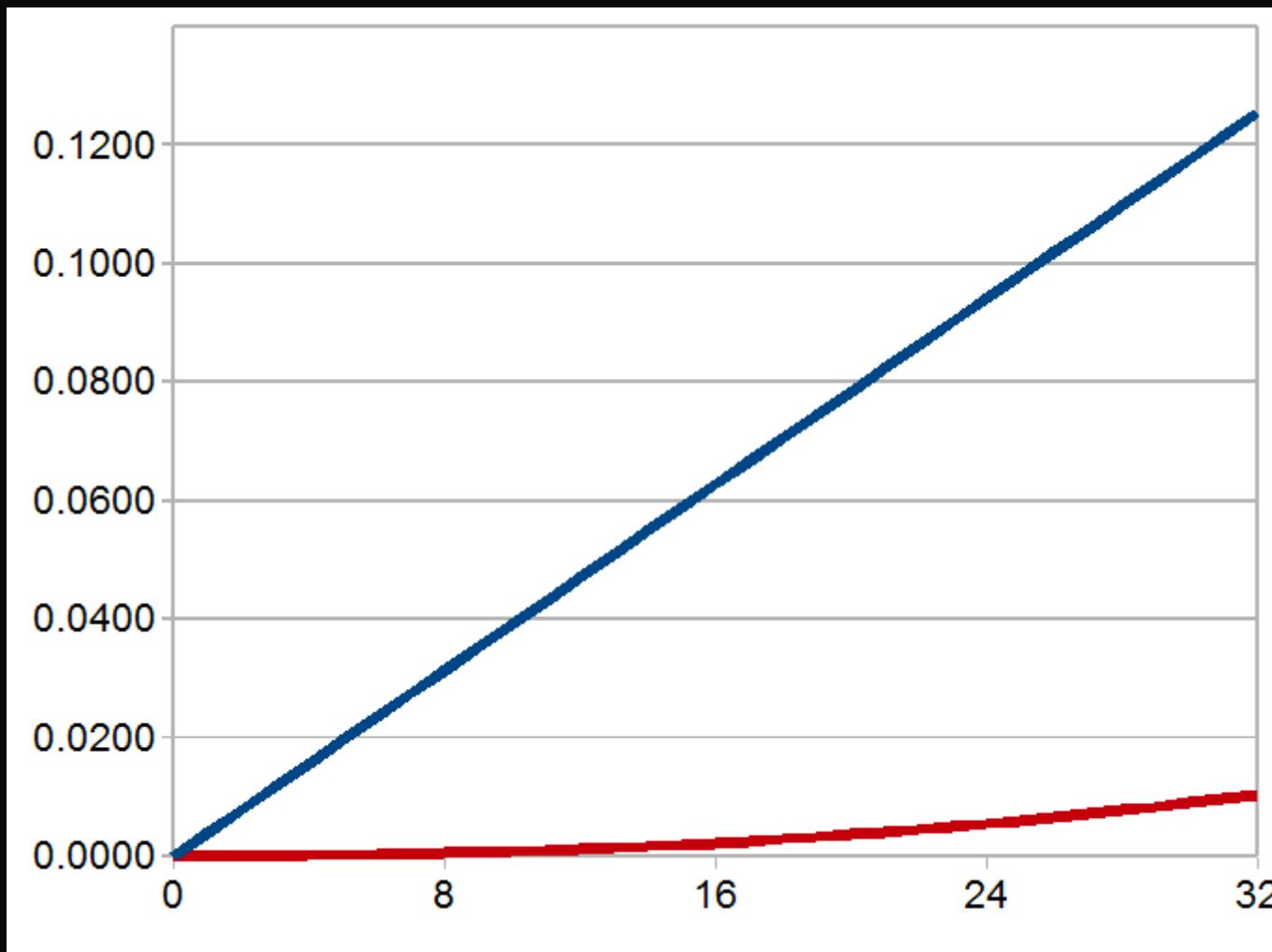
---

- Suppose we look at the 0-63 range.
- What if displays were linear?



# Gamma

- Gamma of 2.2 gives us more dark colors.



# How bad is it?

---

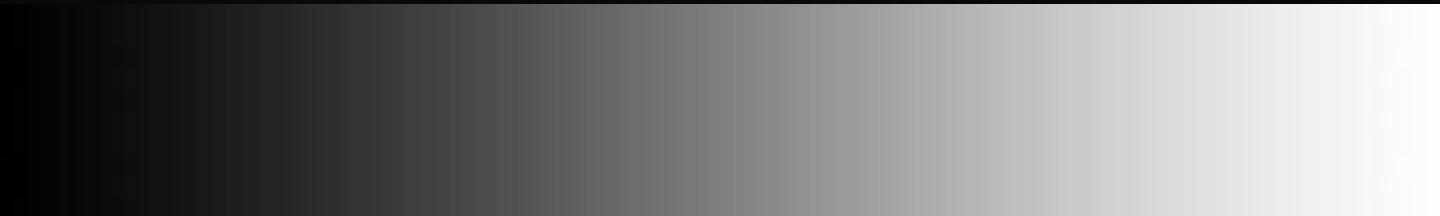
- If displays were linear:
  - On a scale from 0-255
  - 0 would equal 0
  - 1 would equal our current value of 20
  - 2 would equal our current value of 28
  - 3 would equal our current value of 33
- Darkest Values:



# Ramp Again

---

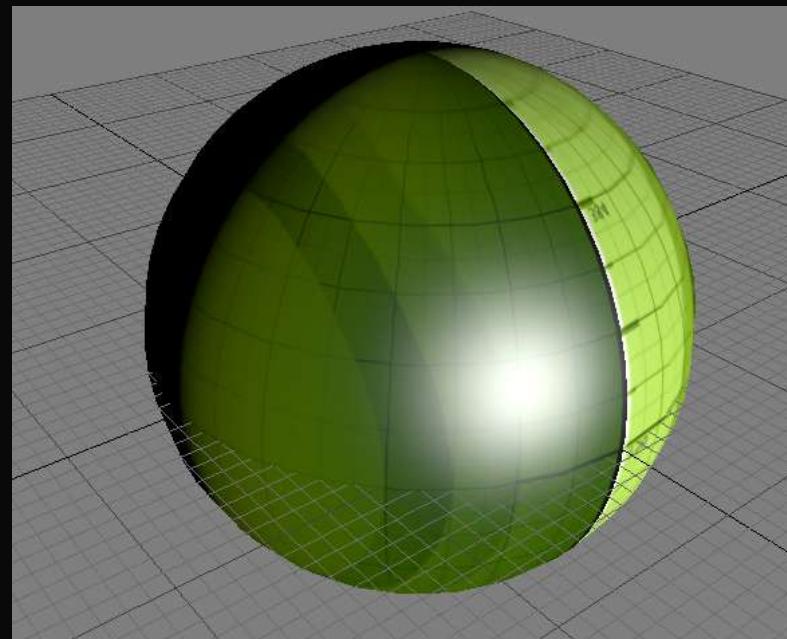
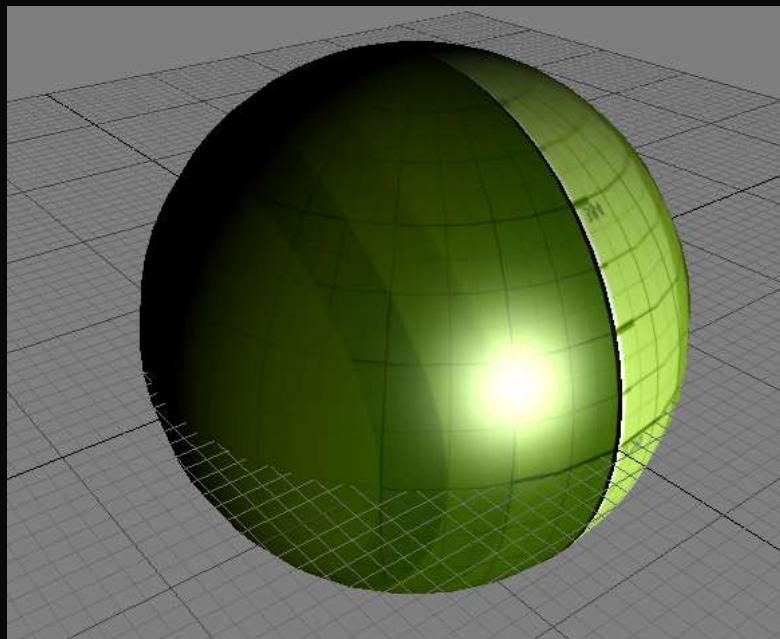
- See any banding in the whites?



# 3D Graphics

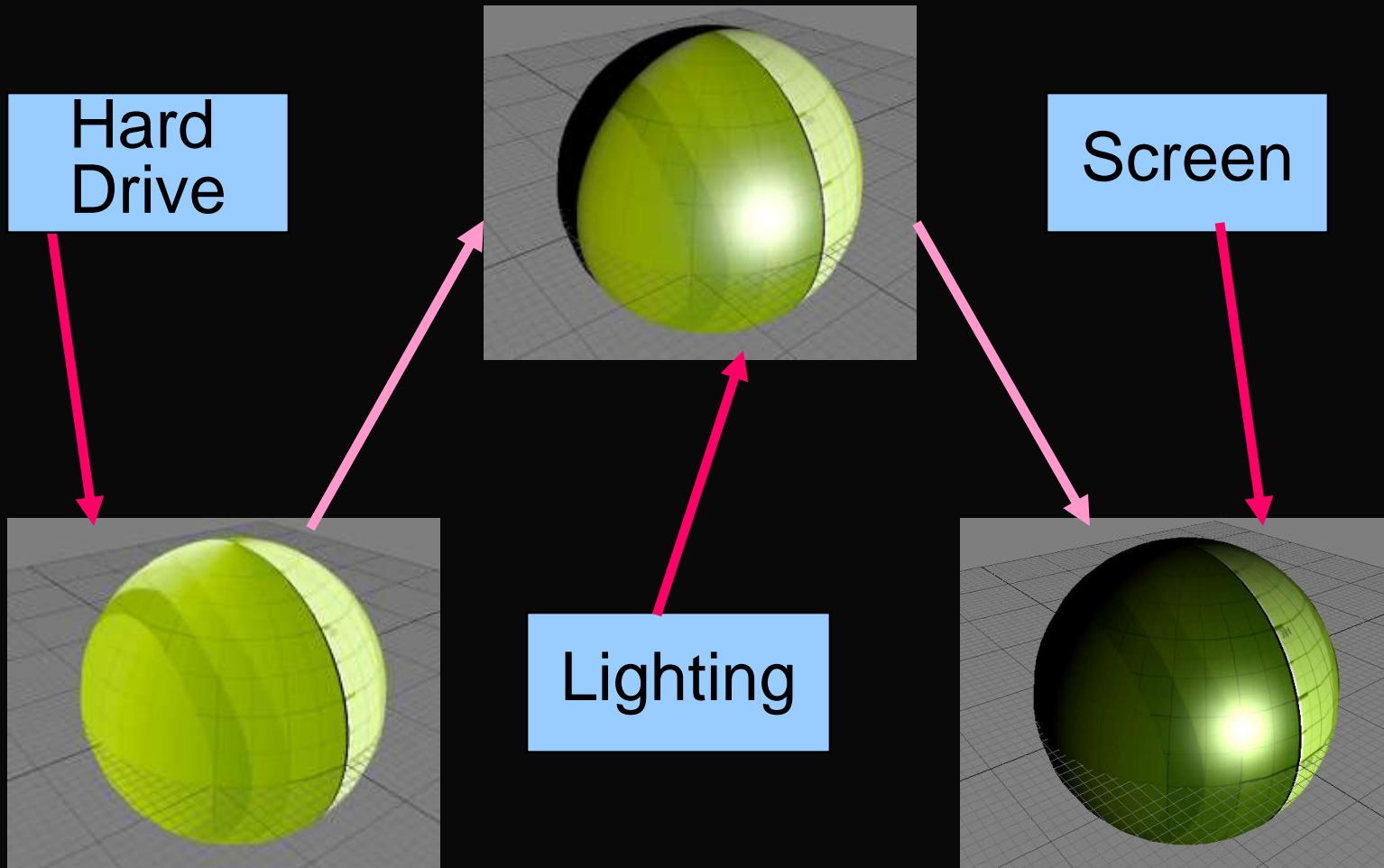
---

- If your game is not gamma correct, it looks like the image on the left..
- Note: we're talking about “correct”, not “good”



# 3D Graphics

- Here's what is going on.



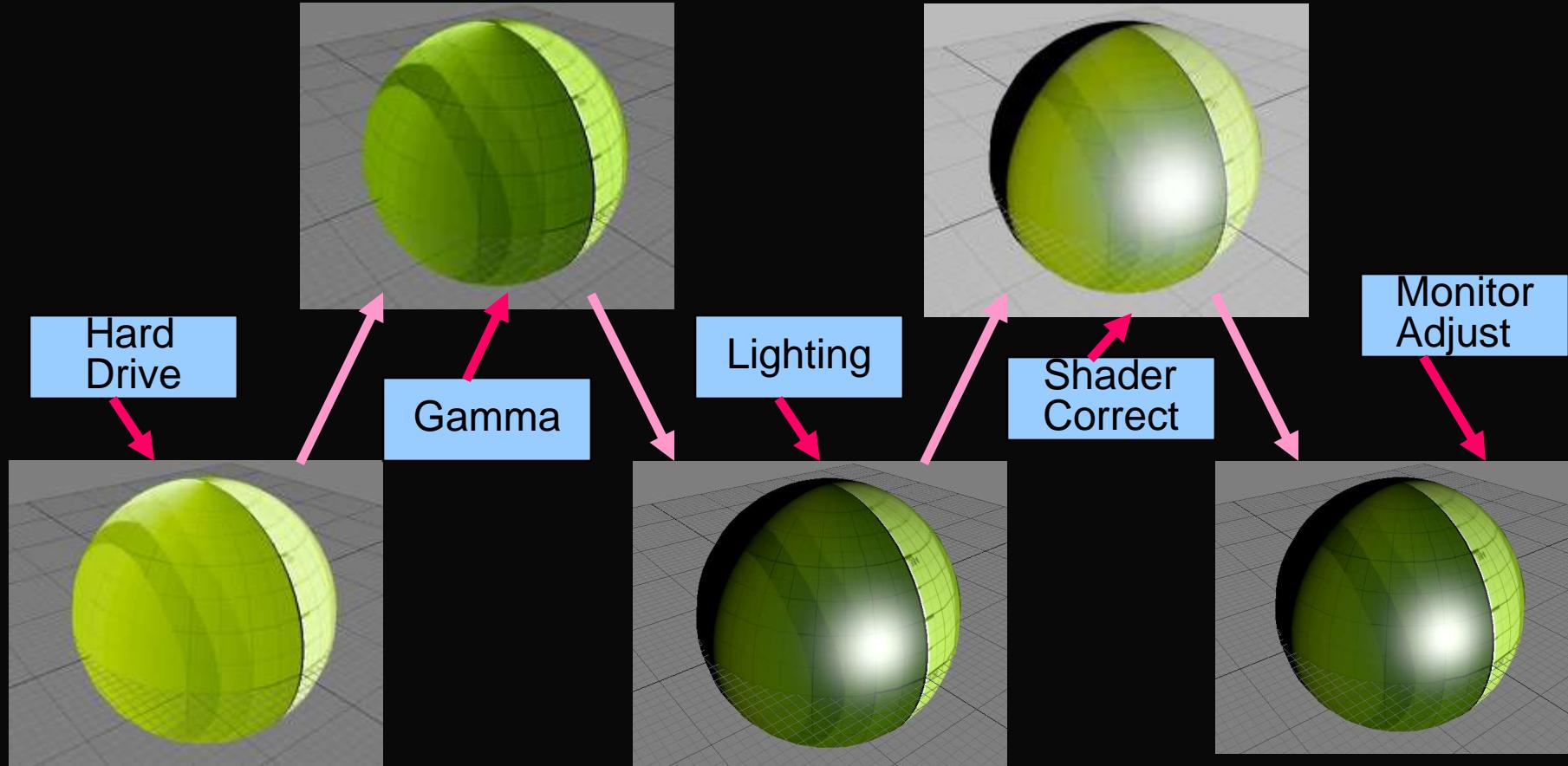
# 3D Graphics

---

- Account for gamma when reading textures
  - $\text{color} = \text{pow}(\text{tex2D( Sampler, Uv )}, 2.2)$
- Do your lighting calculations
- Account for gamma on color output
  - $\text{finalcolor} = \text{pow}(\text{color}, 1/2.2)$

# 3D Graphics

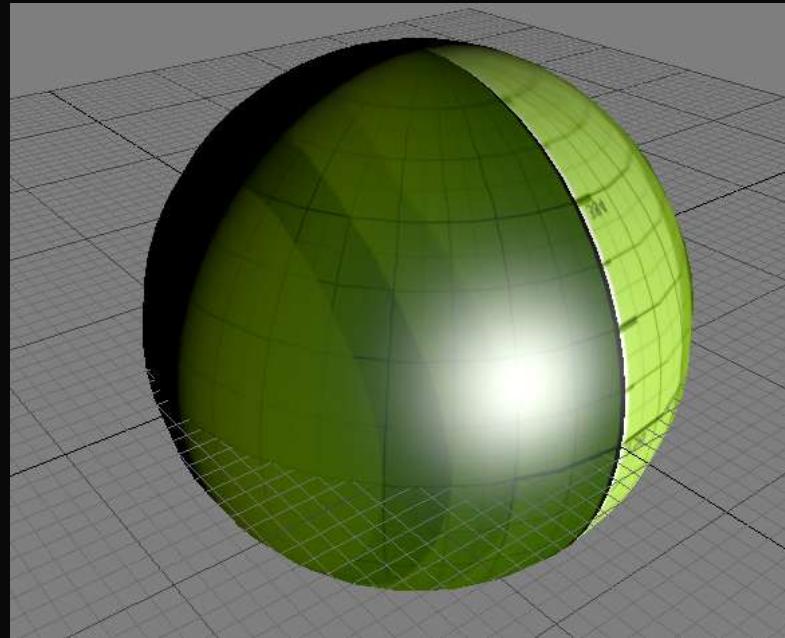
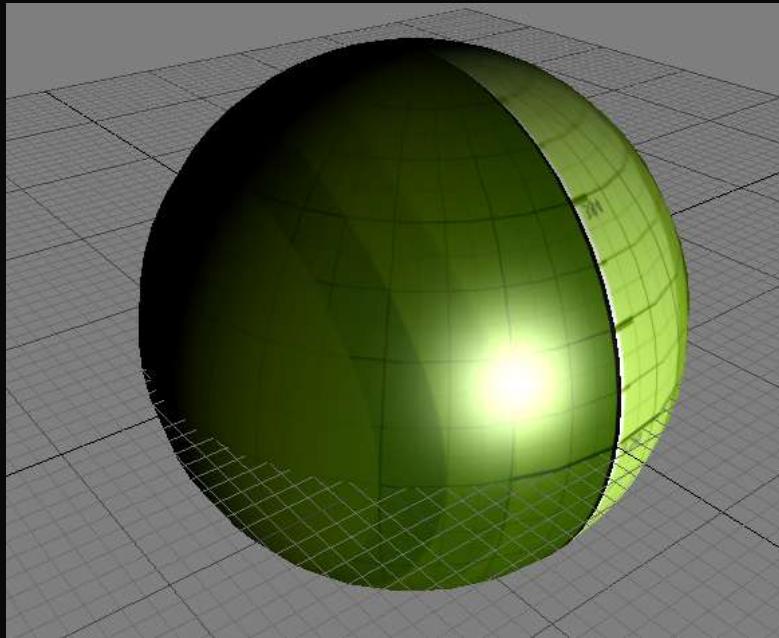
- Here's what is going on.



# 3D Graphics

---

- Comparison again...



# 3D Graphics

---

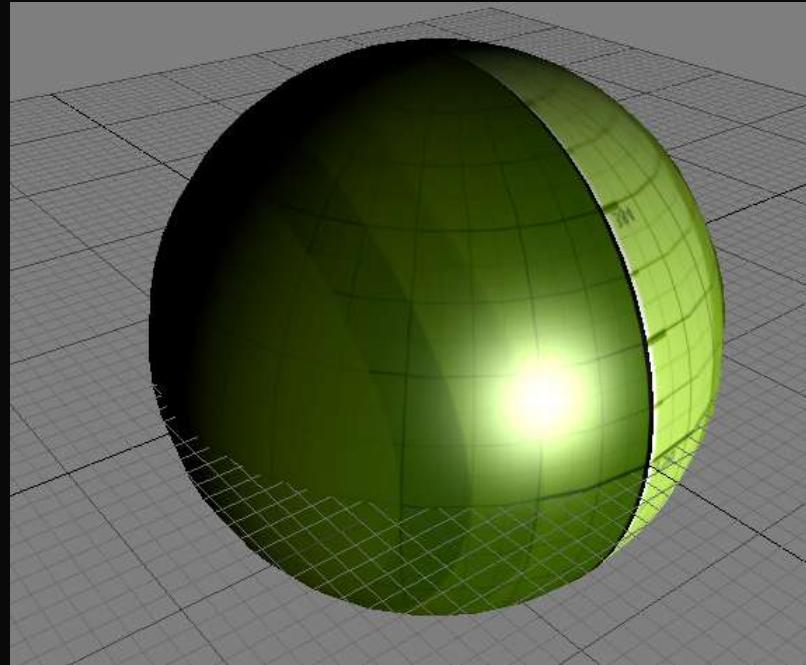
- The Wrong Shader?

```
Spec = CalSpec();
```

```
Diff = tex2D( Sampler, UV );
```

```
Color = Diff * max( 0, dot( N, L ) ) + Spec;
```

```
return Color;
```



# 3D Graphics

---

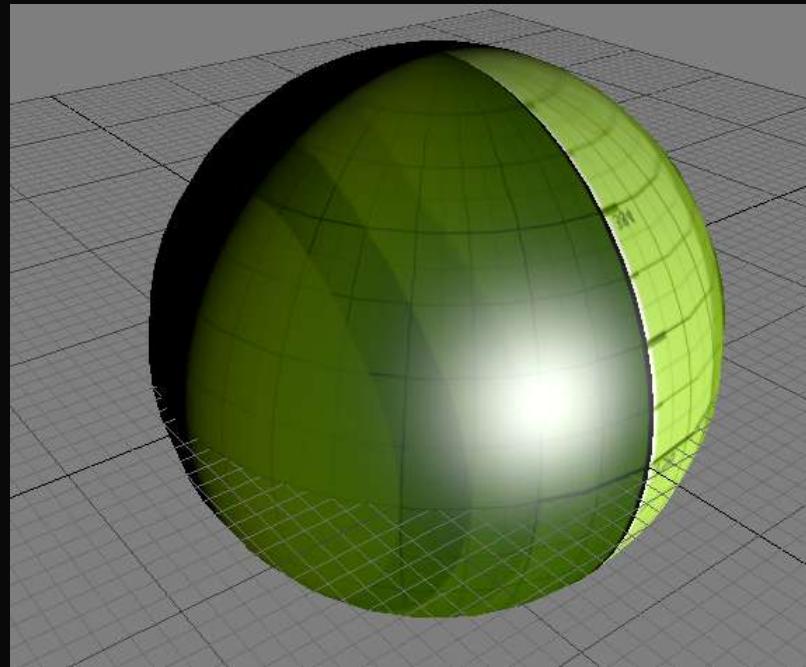
- The Right Shader?

```
Spec = CalSpec();
```

```
Diff = pow( tex2D( Sampler, UV ), 2.2 );
```

```
Color = Diff * max( 0, dot( N, L ) ) + Spec;
```

```
return pow( Color, 1/2.2 );
```



# 3D Graphics

---

- But, there is hardware to do this for us.
  - Hardware does sampling for free
  - For Texture read:
    - D3DSAMP\_SRGBTEXTURE
  - For RenderTarget write:
    - D3DRS\_SRGBWRITEENABLE

# 3D Graphics

---

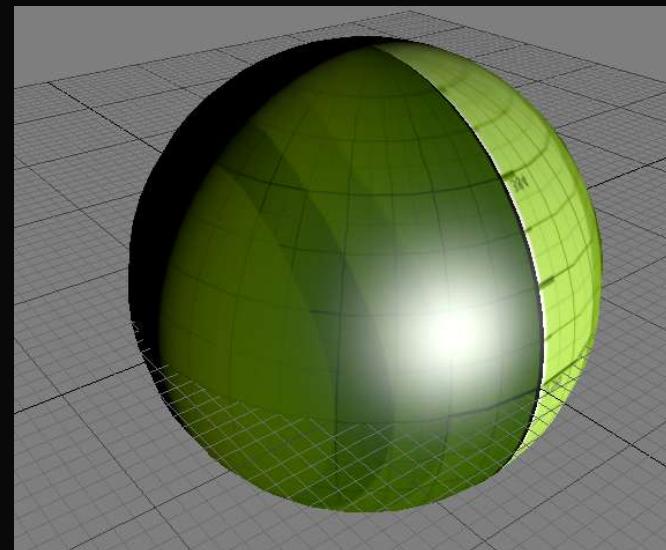
- With those states we can remove the pow functions.

```
Spec = CalSpec();
```

```
Diff = tex2D( Sampler, UV );
```

```
Color = Diff * max( 0, dot( N, L ) ) + Spec;
```

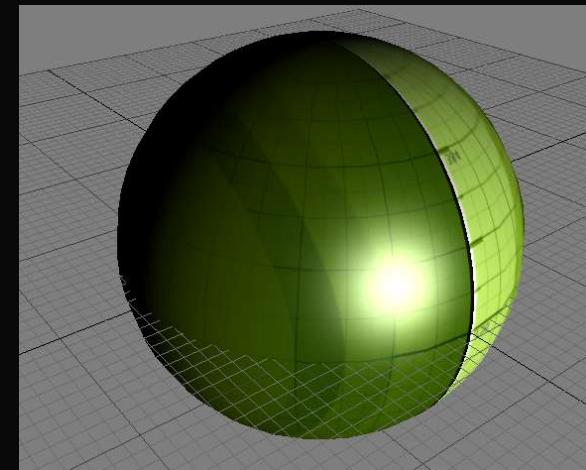
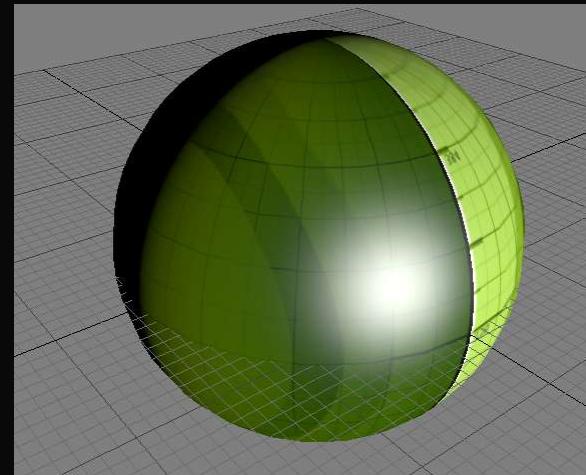
```
return Color;
```



# 3D Graphics

---

- What does the real world look like?
  - Notice the harsh line.



# 3D Graphics

---

- Another Example

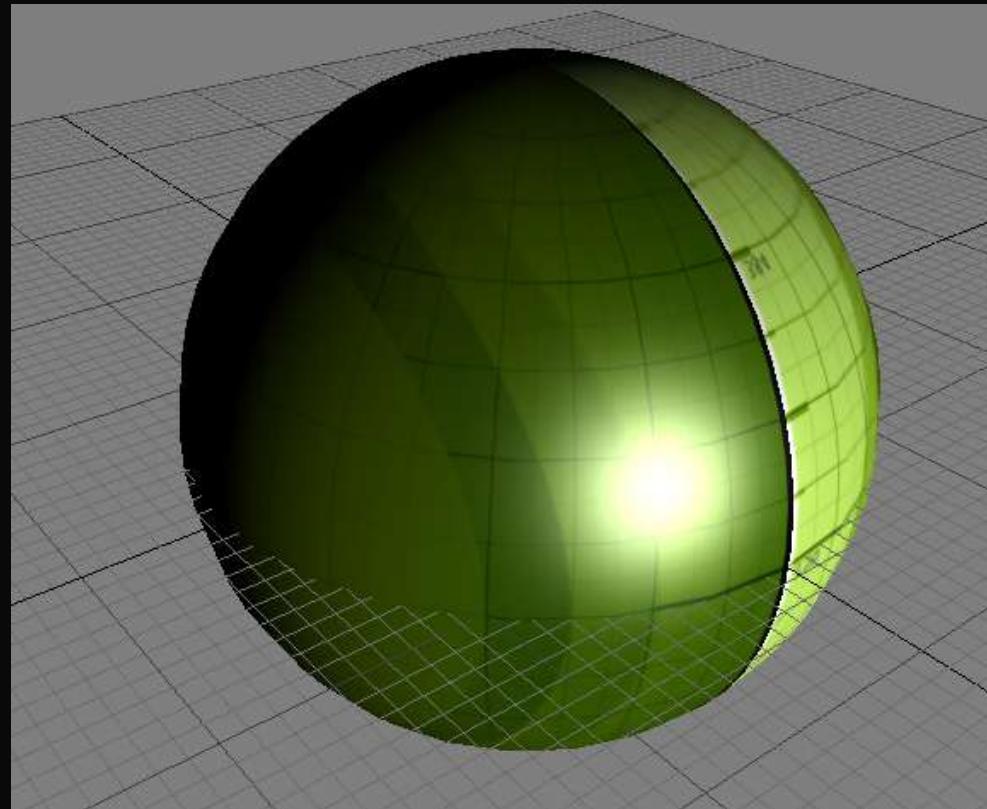


# FAQs

---

Q1) But I like the soft falloff!

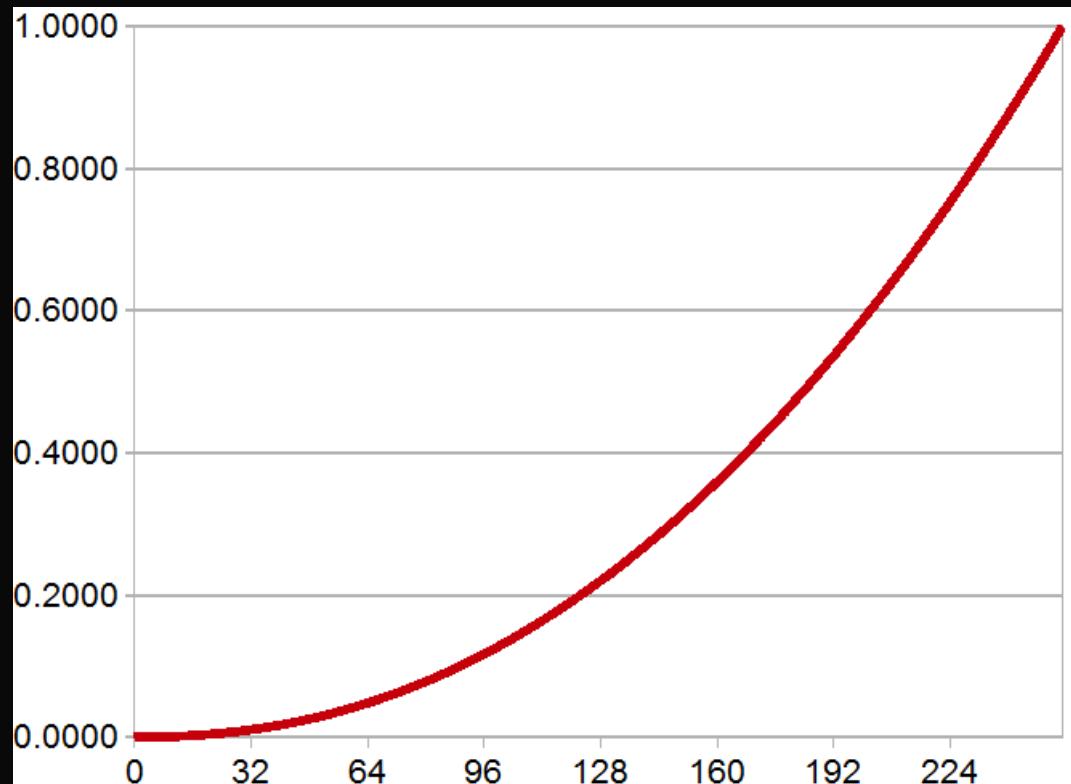
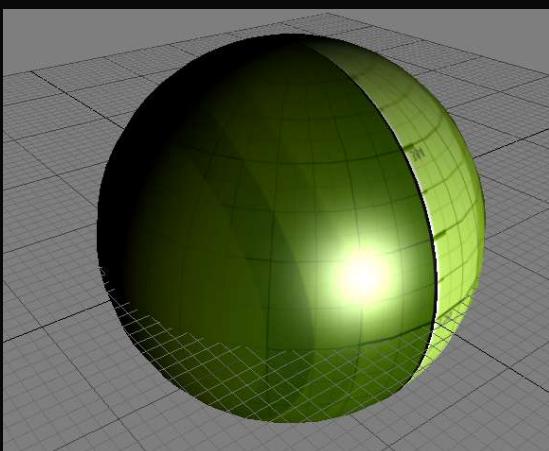
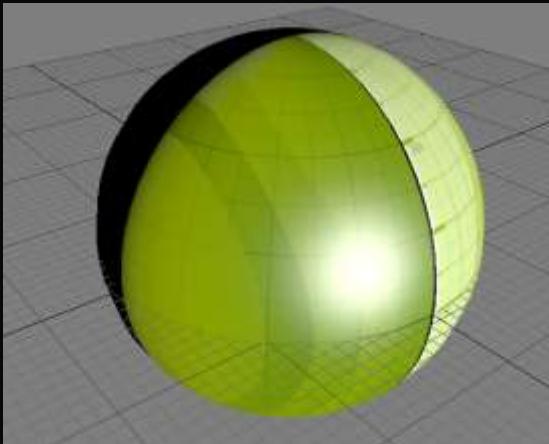
A1) Don't be so sure.



# FAQs

---

- It looks like the one on top before the monitor's 2.2



# Harsh Falloff

---

- Devastating.



# Harsh Falloff

---

- Devastating.



# Harsh Falloff

---

- Devastating.



# Which Maps?

---

- Which maps should be Linear-Space vs Gamma-Space?
  - Gamma-Space
    - Use sRGB hardware or  $\text{pow}(2.2)$  on read
    - $128 \approx 0.2$
  - Linear-Space
    - Don't use sRGB or  $\text{pow}(2.2)$  on read
    - $128 = 0.5$

# Which Maps?

---

- Diffuse Map
  - Definitely Gamma
- Normal Map
  - Definitely Linear

# Which Maps?

---

- Specular?
  - Uncharted 2 had them as Linear
  - Artists have trouble tweaking them to look right
  - Probably should be in Gamma

# Which Maps?

---

- Ambient Occlusion
  - Technically, it's a mathematical value like a normal map.
    - But artists tweak them a lot and bake extra lighting into them.
  - Uncharted 2 had them as Linear
  - Probably should be Gamma

# Exercise for the Reader

---

- Gamma  $2.2 \neq$  sRGB  $\neq$  Xenon PWL
- sRGB: PC and PS3 gamma
- Xenon PWL: Piecewise linear gamma

# Xenon Gotchas

---

- Xbox 360 Gamma Curve is wonky
  - Way too bright at the low end.
    - *HDR the Bungie Way*, Chris Tchou
    - *Post Processing in The Orange Box*, Alex Vlachos
  - Output curve is extra contrasty
    - Henry LaBounta was going to talk about it.
    - Try hooking up your Xenon to a waveform monitor and display a test pattern. Prepare to be mortified.
  - Both factors counteract each other

# Linear-Space Lighting: Conclusion

---

“Drake! You must *believe* in linear-space lighting!”



# Part 2: Filmic Tonemaping

---



# Filmic Tonemapping

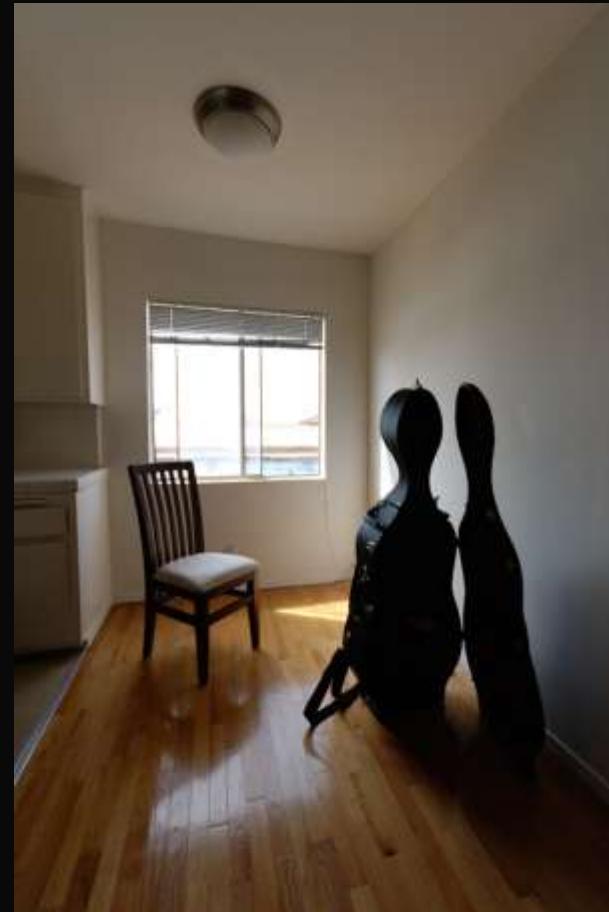
---

- Let's talk about the magic of the human eye.

# Filmic Tonemapping

---

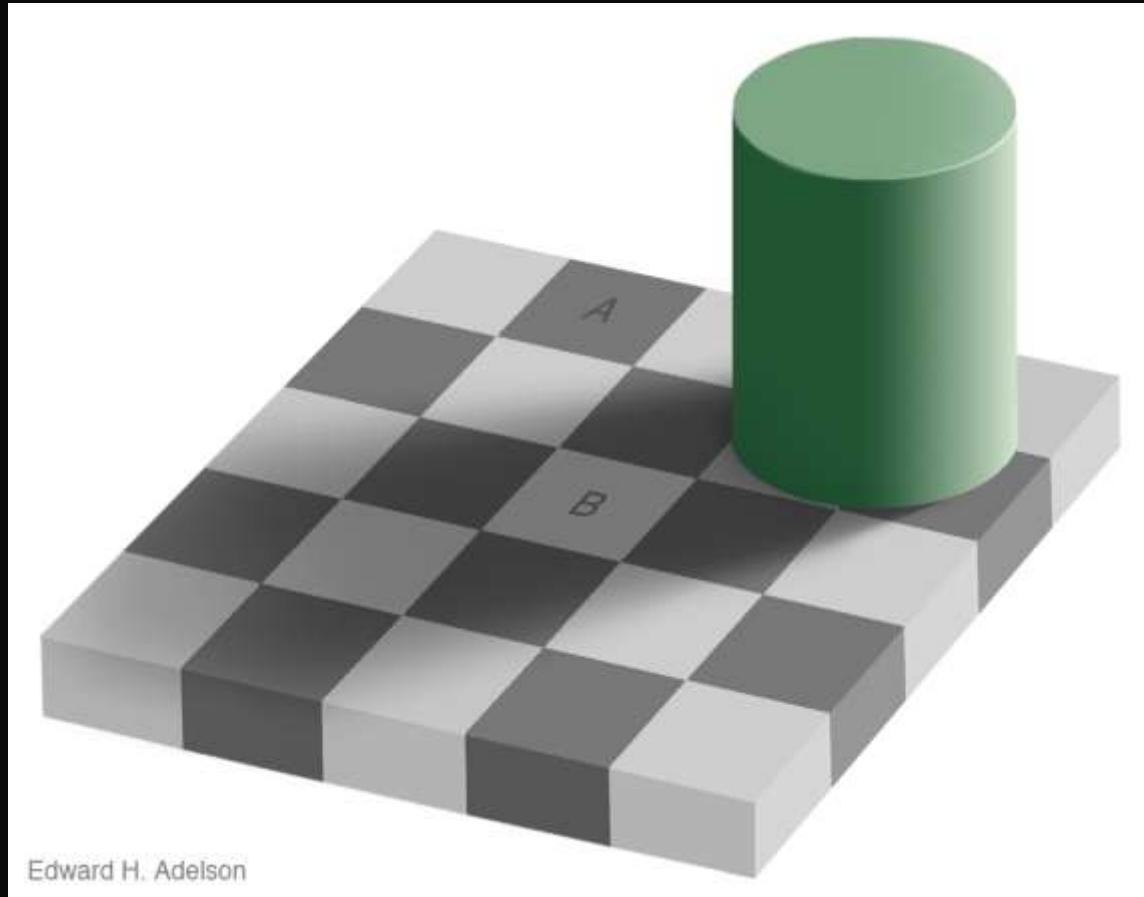
- Real example from my apartment.
- Full Auto settings.
- Outside blown out.
- Cello case looks empty.
- My eye somehow adjusts...



# Filmic Tonemapping

---

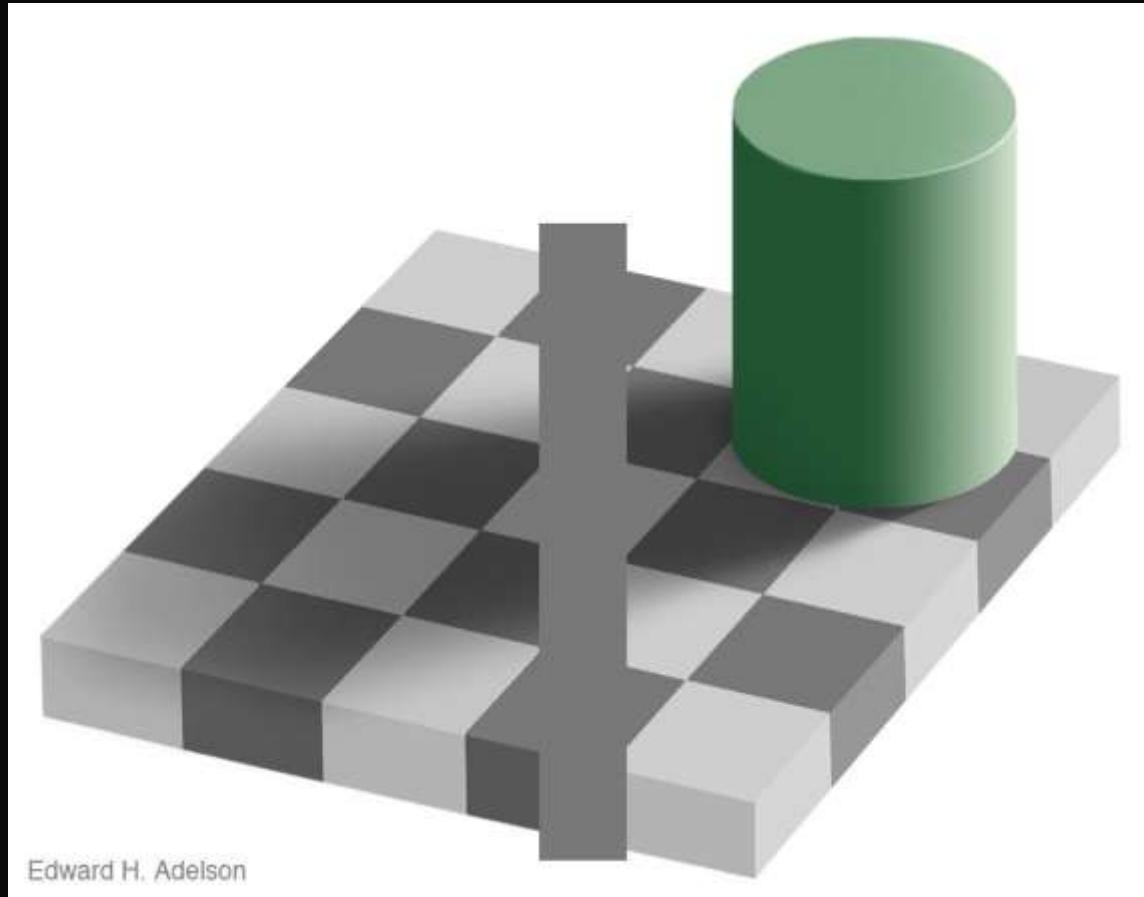
- Adaptiveness of human eye.



# Filmic Tonemapping

---

- Adaptiveness of human eye.

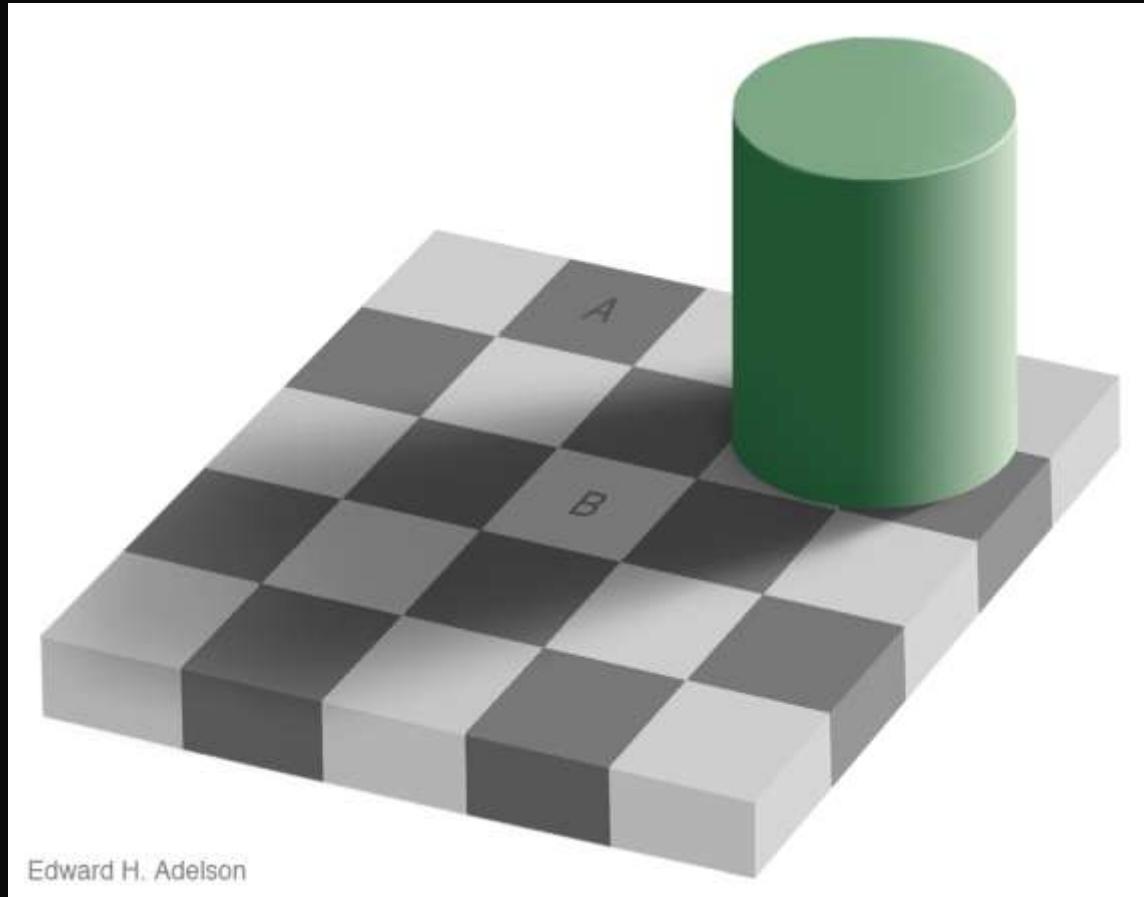


Edward H. Adelson

# Filmic Tonemapping

---

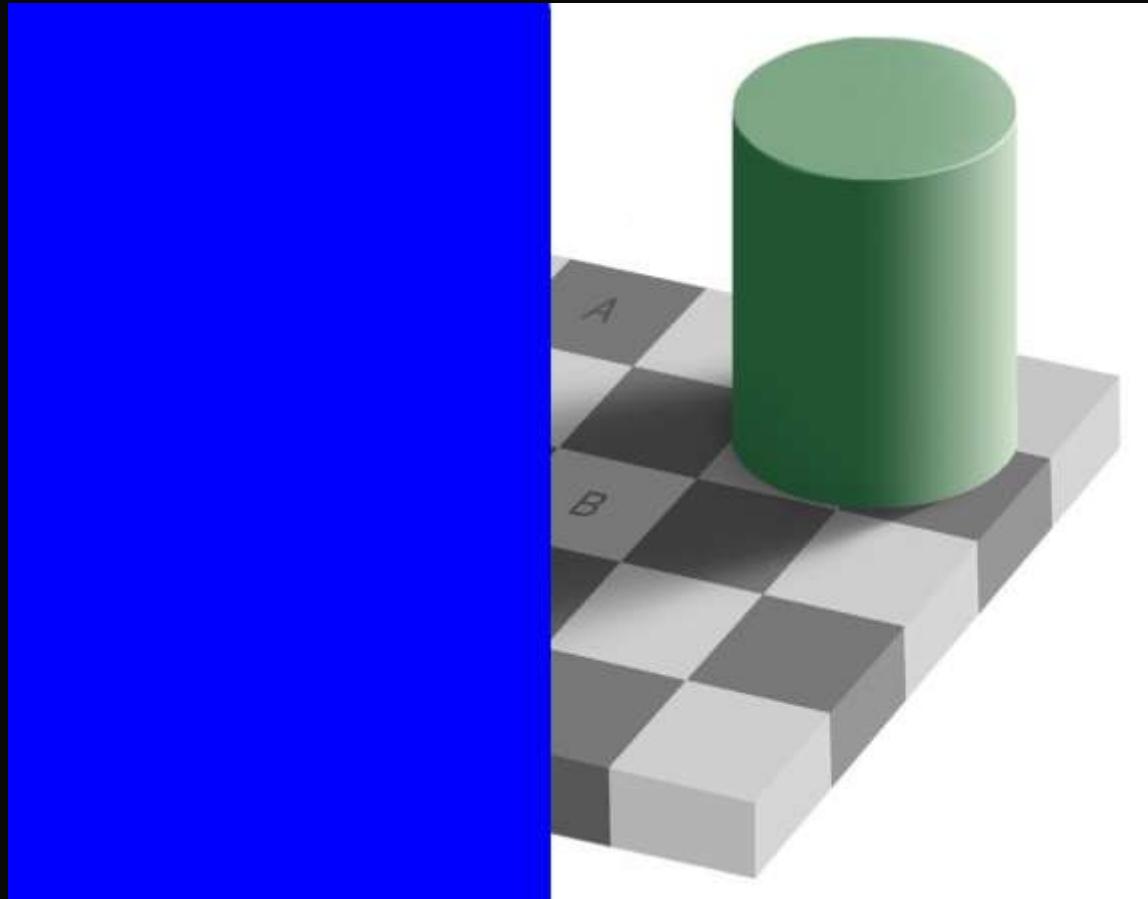
- Adaptiveness of human eye.



# Filmic Tonemapping

---

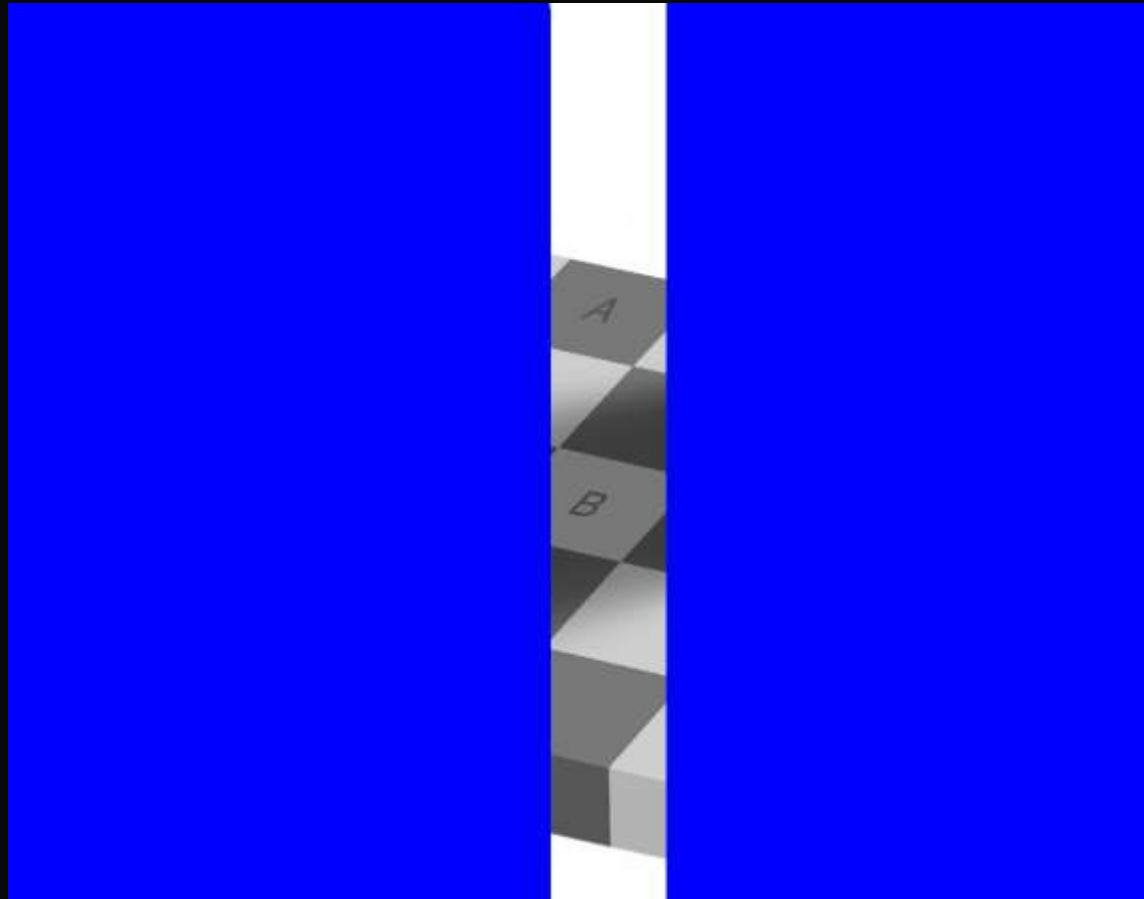
- Adaptiveness of human eye.



# Filmic Tonemapping

---

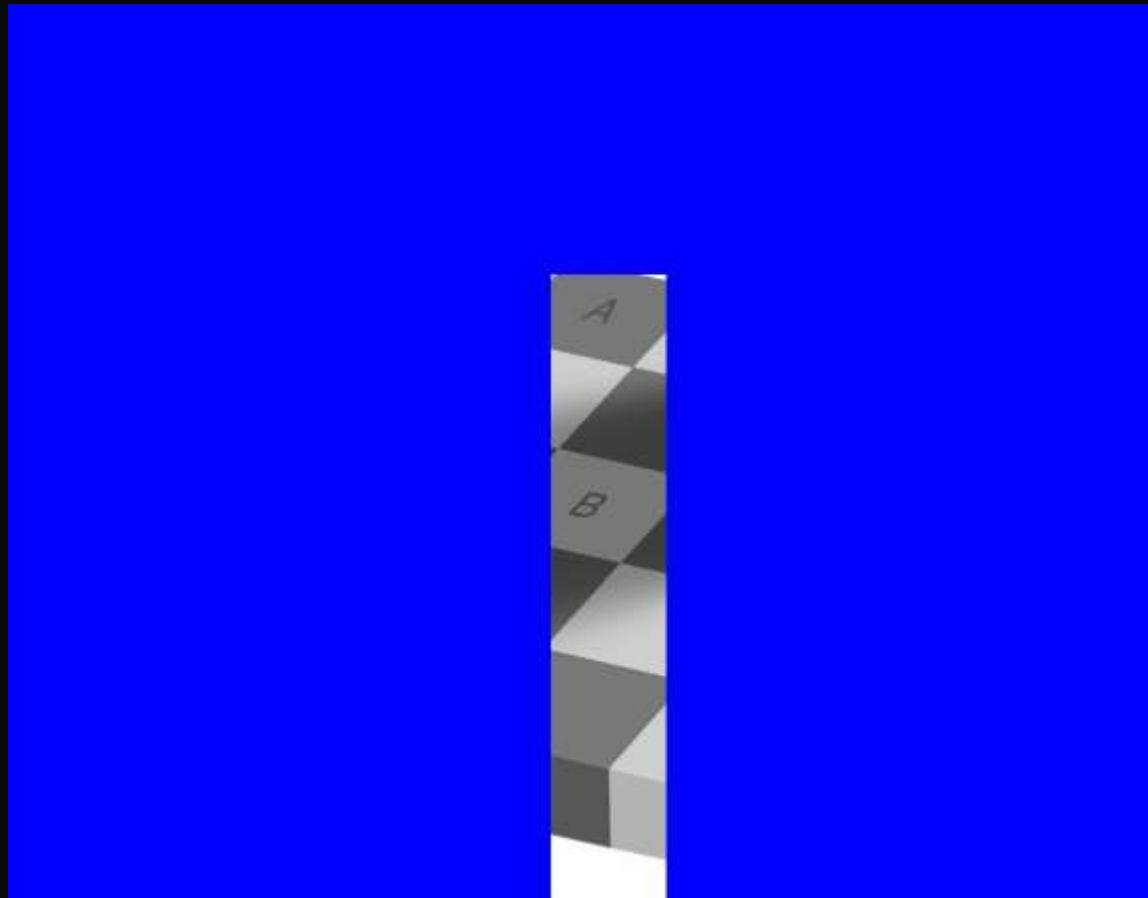
- Adaptiveness of human eye.



# Filmic Tonemapping

---

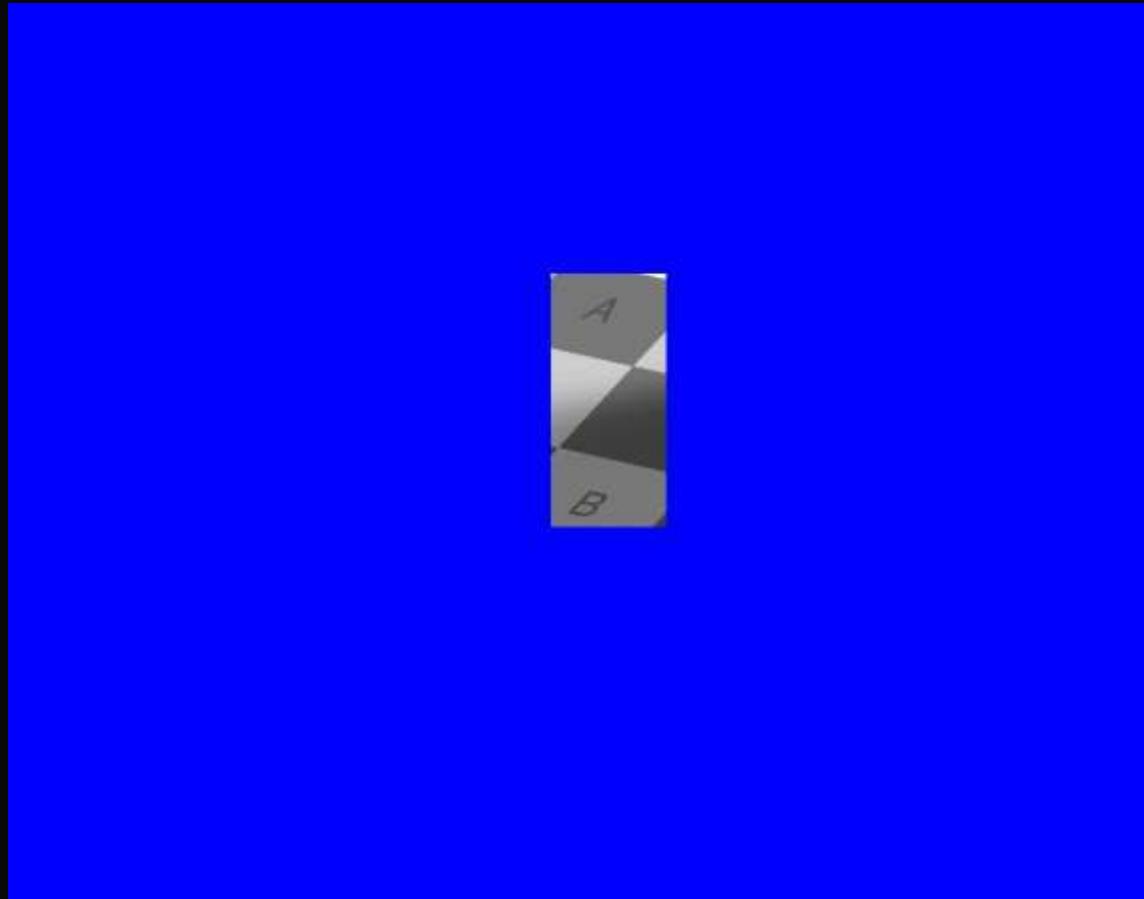
- Adaptiveness of human eye.



# Filmic Tonemapping

---

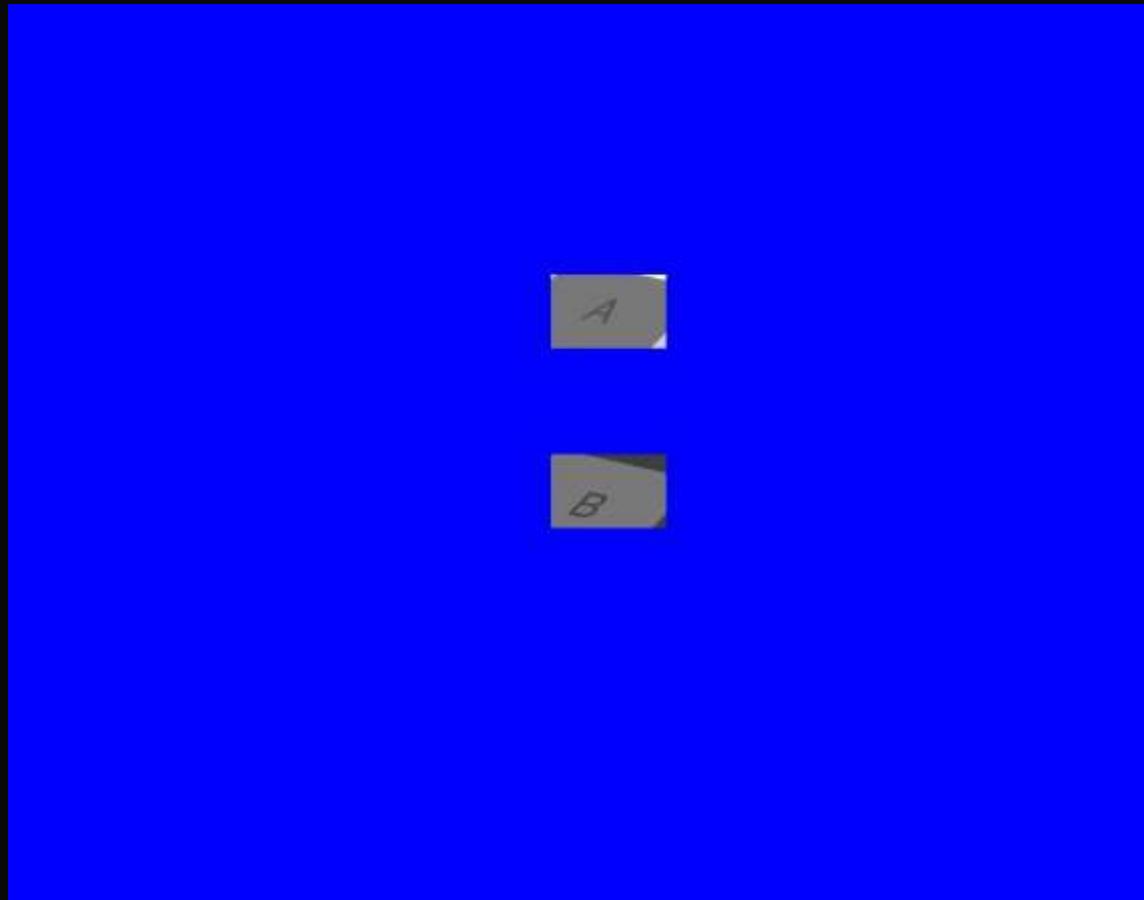
- Adaptiveness of human eye.



# Filmic Tonemapping

---

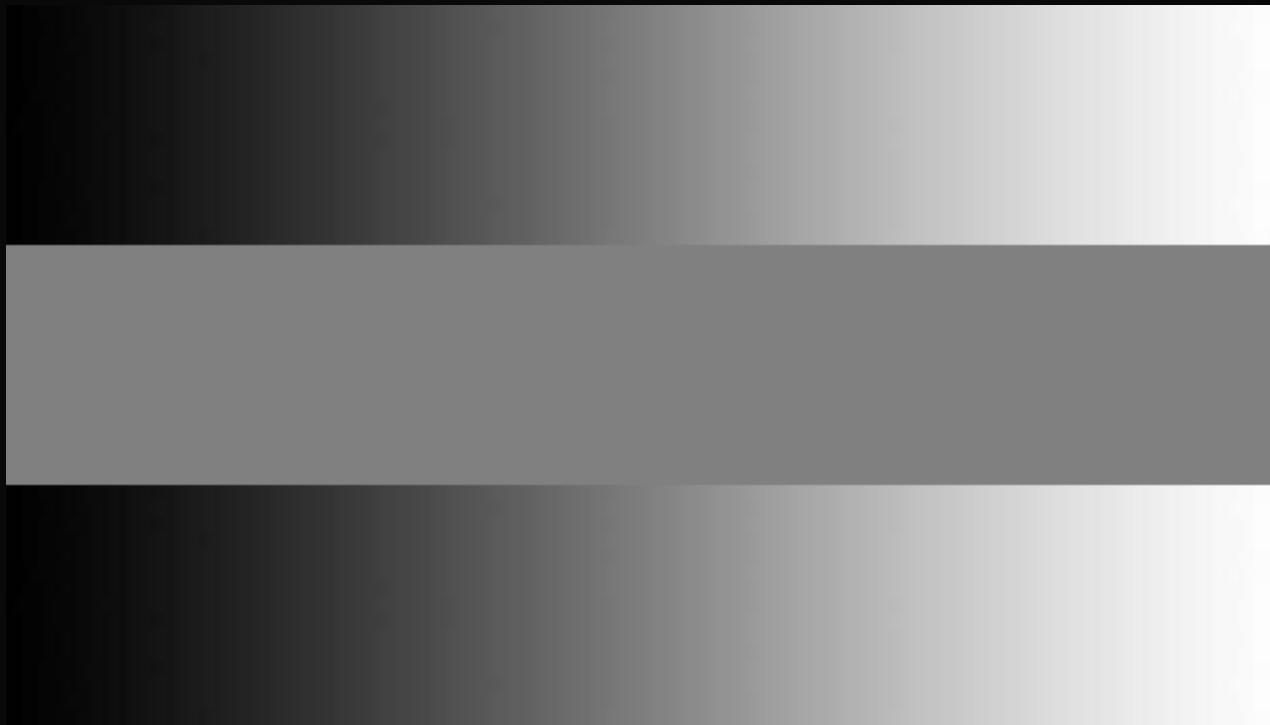
- Adaptiveness of human eye.



# Filmic Tonemapping

---

- One more



# Filmic Tonemapping

---

- One more



# Gamma

---

- Let's take an HDR image
  - Note: 1 F-Stop is a power of two
  - So 4 F-stops is  $2^4=16x$  difference in intensity

# Gamma

---

- Exposure: -4



# Gamma

---

- Exposure: -2



# Gamma

---

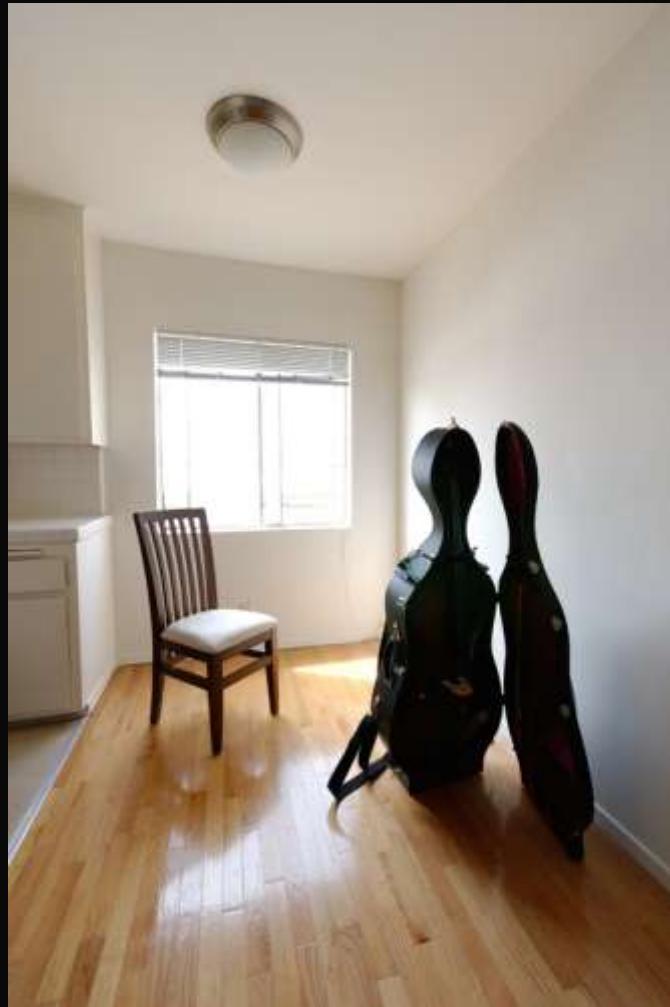
- Exposure: 0



# Gamma

---

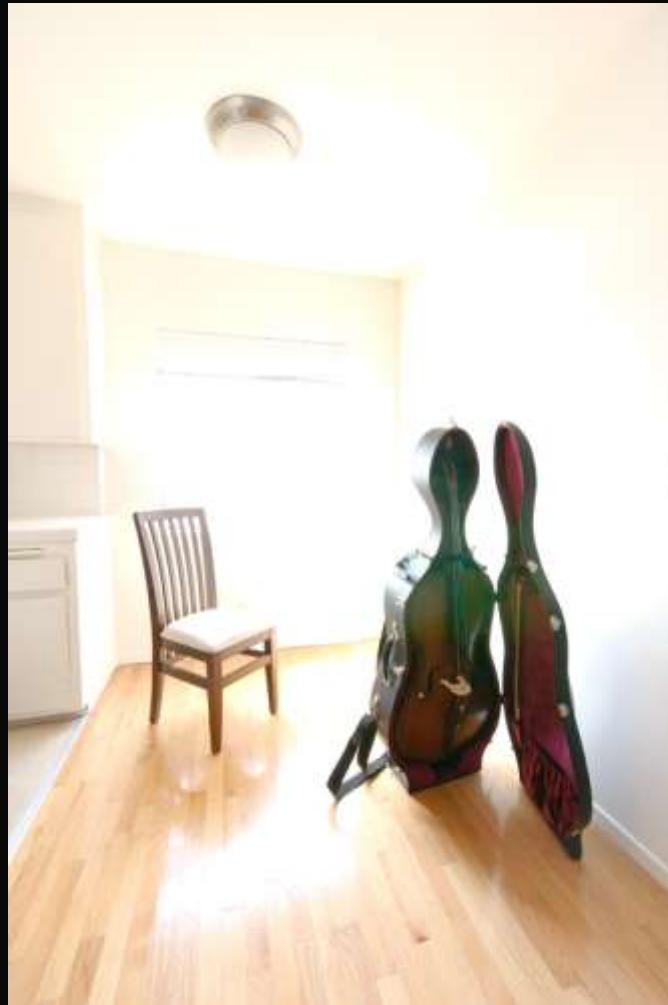
- Exposure: 2



# Gamma

---

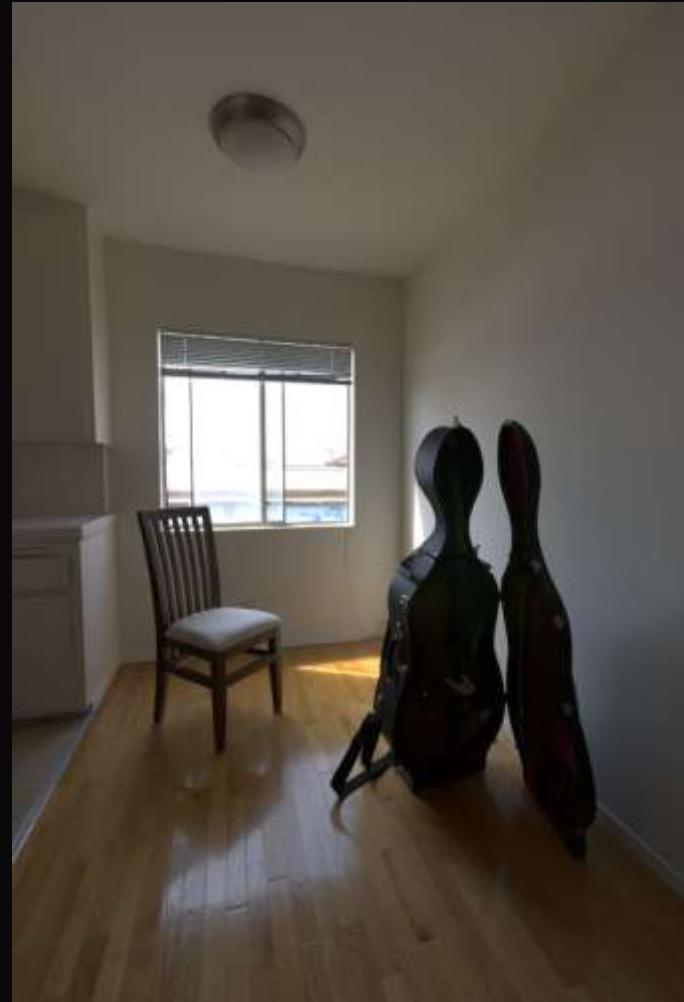
- Exposure: 4



# Now what?

---

- We actually need 8 stops
- Good News: HDR Image
- Bad News: Now what?
- Re-tweaking exposure won't help



# Gamma

---

- This is what Photography people call tonemapping.
  - Photomatix
  - This one is local
  - The one we'll use is global



# Point of Confusion

---

- Two problems to solve.
  - Simulate the Iris
    - Dynamic Range in different shots
    - Tunnel vs. Daylight
  - Simulate the Retina
    - Different Range within a shot
    - Inside looking outside

# Terminology

---

- Photography/Film
  - Within a Shot – HDR Tonemapping
  - Different Shots – Choosing the correct exposure?
- Video Games
  - Within a Shot – HDR Tonemapping
  - Different Shots – HDR Tonemapping?

# Terminology

---

- For this presentation
  - Within a Shot – HDR Tonemapping
  - Different Shots
    - Automatic Exposure Adjustment
    - Dynamic Iris

# Linear Curve

---

- Apartment of Habib Zargarpour
- Creative Director, Microsoft Game Studios
- Used to be Art Director on LMNO at EA



# Linear Curve

---

- OutColor = pow(x,1/2.2)



# Reinhard

---

- Most common tonemapping operator is Reinhard
- Simplest version is:
  - $F(x) = x/(x+1)$
  - Yes, I'm oversimplifying for time.



# Fade!

---

- Let's fade from linear to Reinhard.



# Fade!

---

- Let's fade from linear to Reinhard.



# Fade!

---

- Let's fade from linear to Reinhard.



# Fade!

---

- Let's fade from linear to Reinhard.



# Fade!

---

- Let's fade from linear to Reinhard.



# Fade!

---

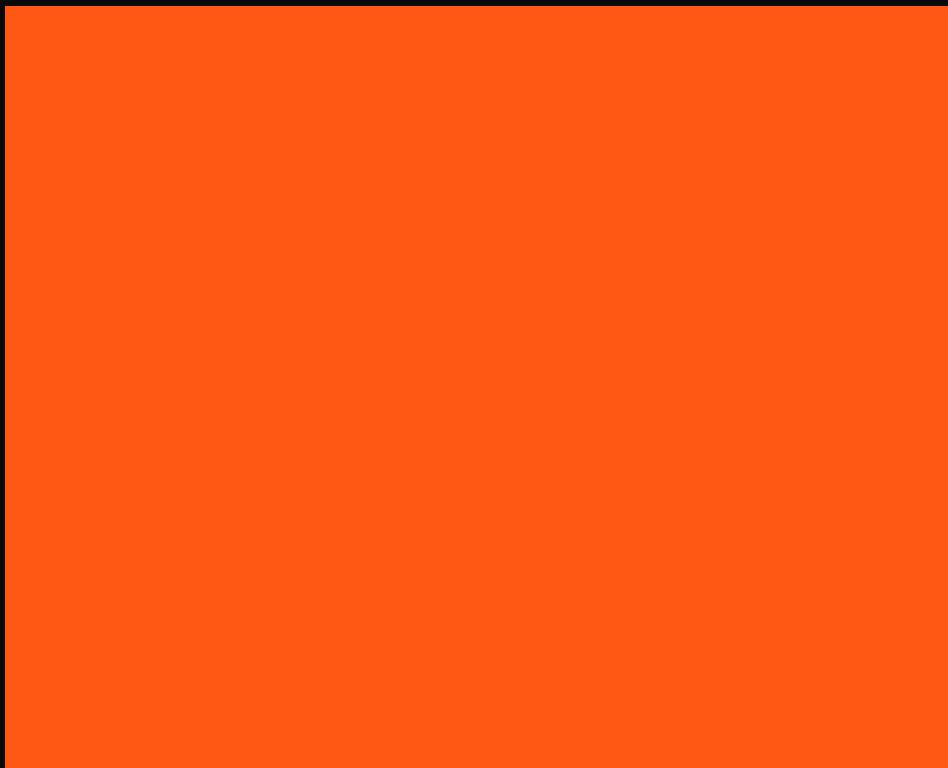
- Let's fade from linear to Reinhard.



# Why Does this happen

---

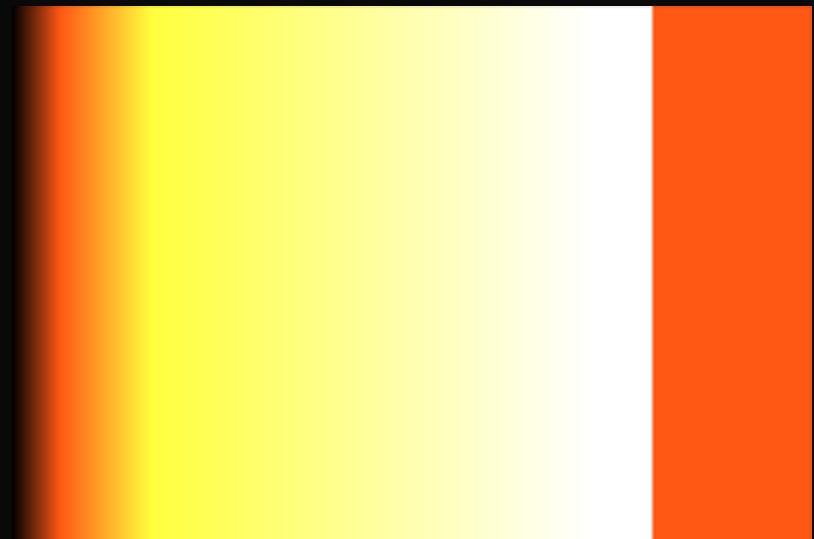
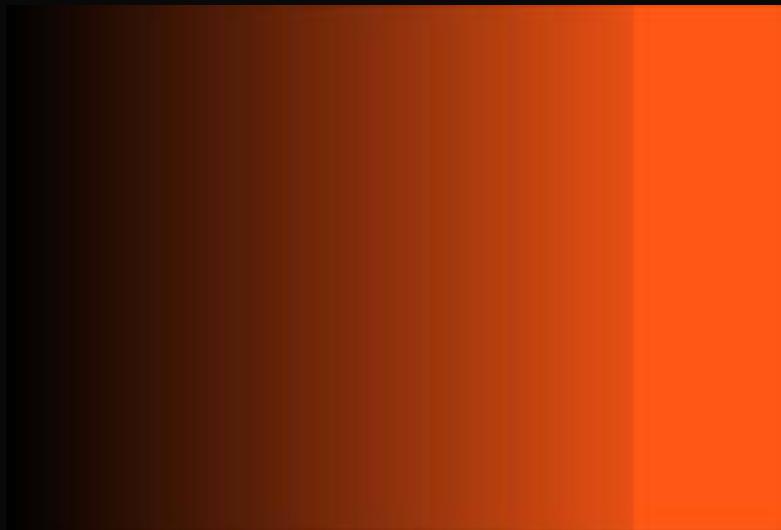
- Start with orange (255,88,21)



# Linear Curve

---

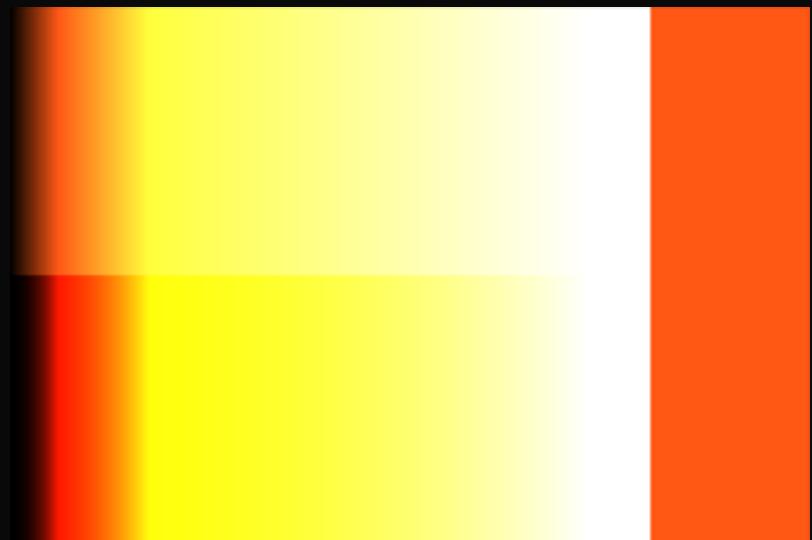
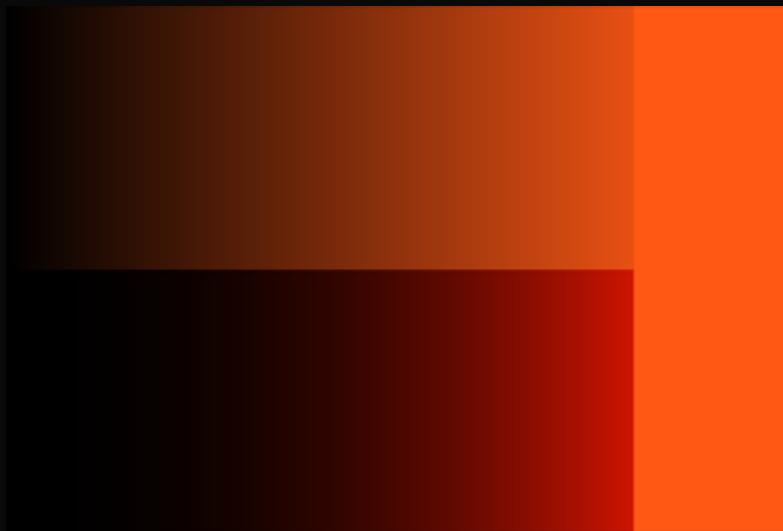
- Linear curve at the low end and high end



# Bad Gamma

---

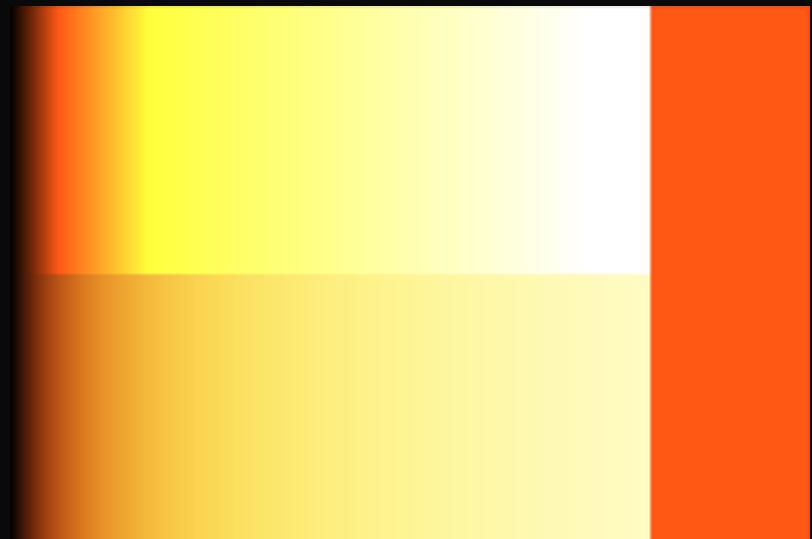
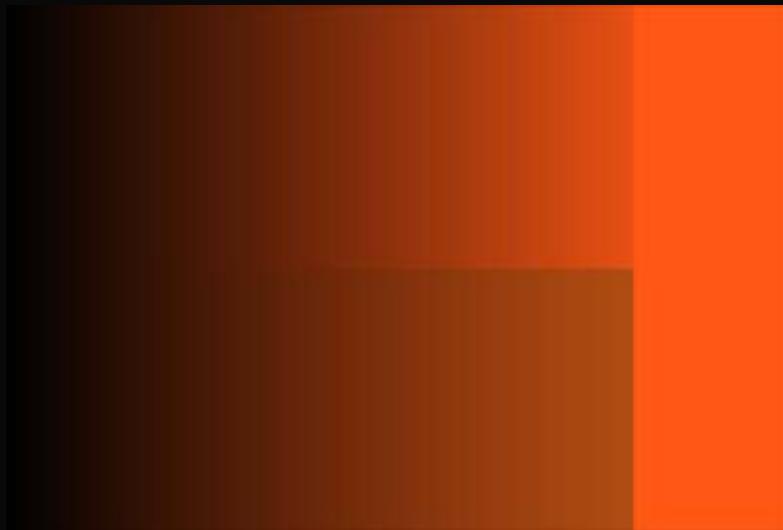
- Better blacks actually, but wrong color at the bottom and horrible hue-shifting.



# Reinhard

---

- Desaturated blacks, but nice top end.



# Color Changes

---

- Gee...if only we could get:
  - The crisper, more saturated blacks of improper gamma.
  - The nice soft highlights of Reinhard
  - And get the input colors to actually match like pure linear.

# Voila!

---

- Guess what? There is a solution!

# Voila!

---

- Solution by Haarm-Peter Duiker
- CG Supervisor at Digital Domain
- Used to be a CG Supervisor on LMNO at EA

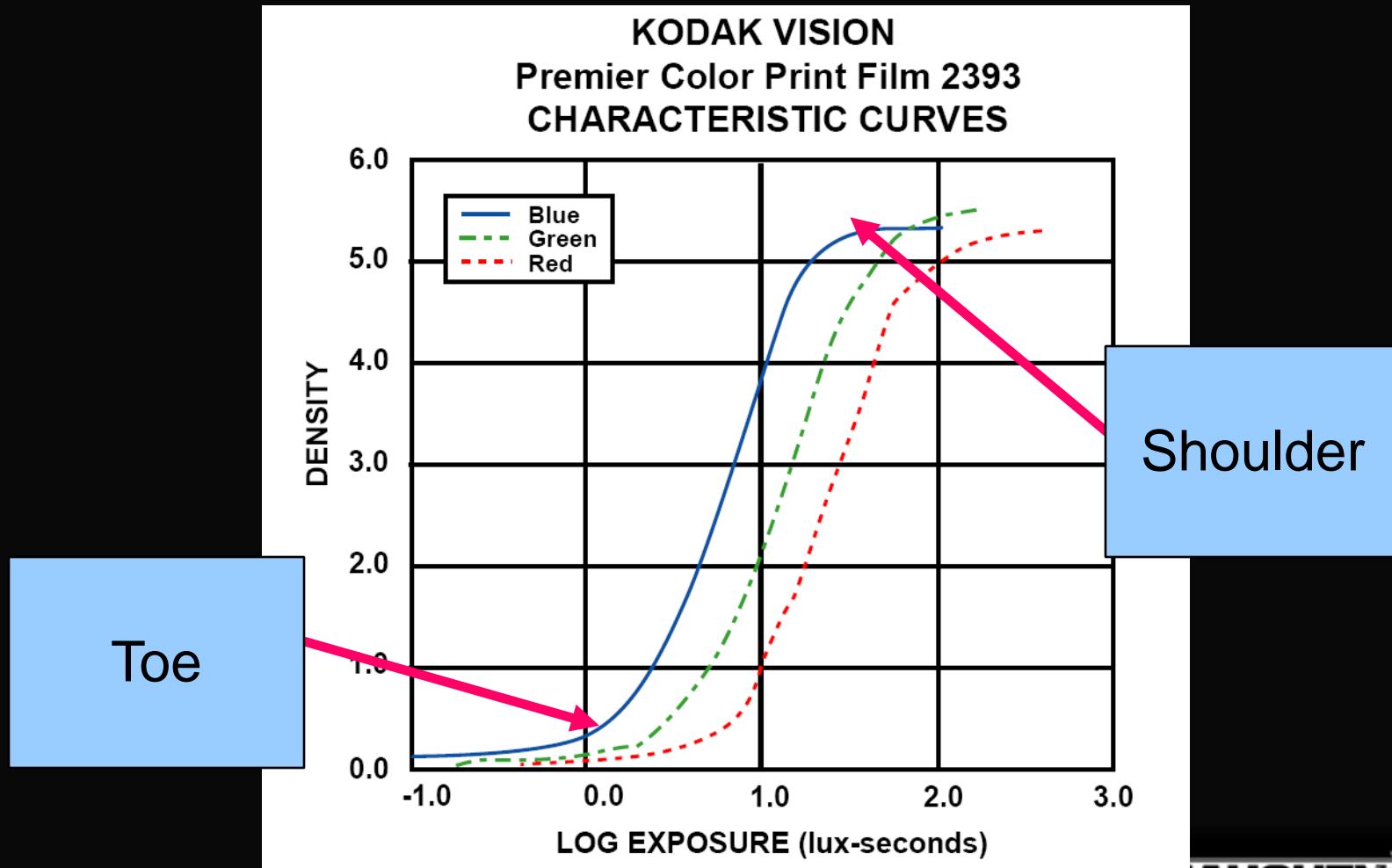
# Film

---

- Q) Why do they still use film in movies?  
A) The look.

# Film

- Kodak film examples.



# Film

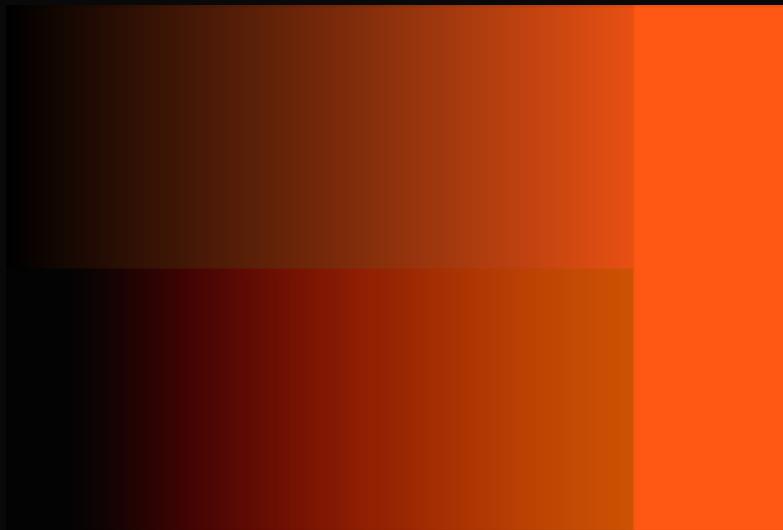
---

- Using a film curve solves tons of problems.
- Film vs. Digital purists.
- Who would've thought: Kodak knows how to make good film.

# Filmic Curve

---

- Crisp blacks, saturated dark end, and nice highlights.



# Filmic Tonemapping

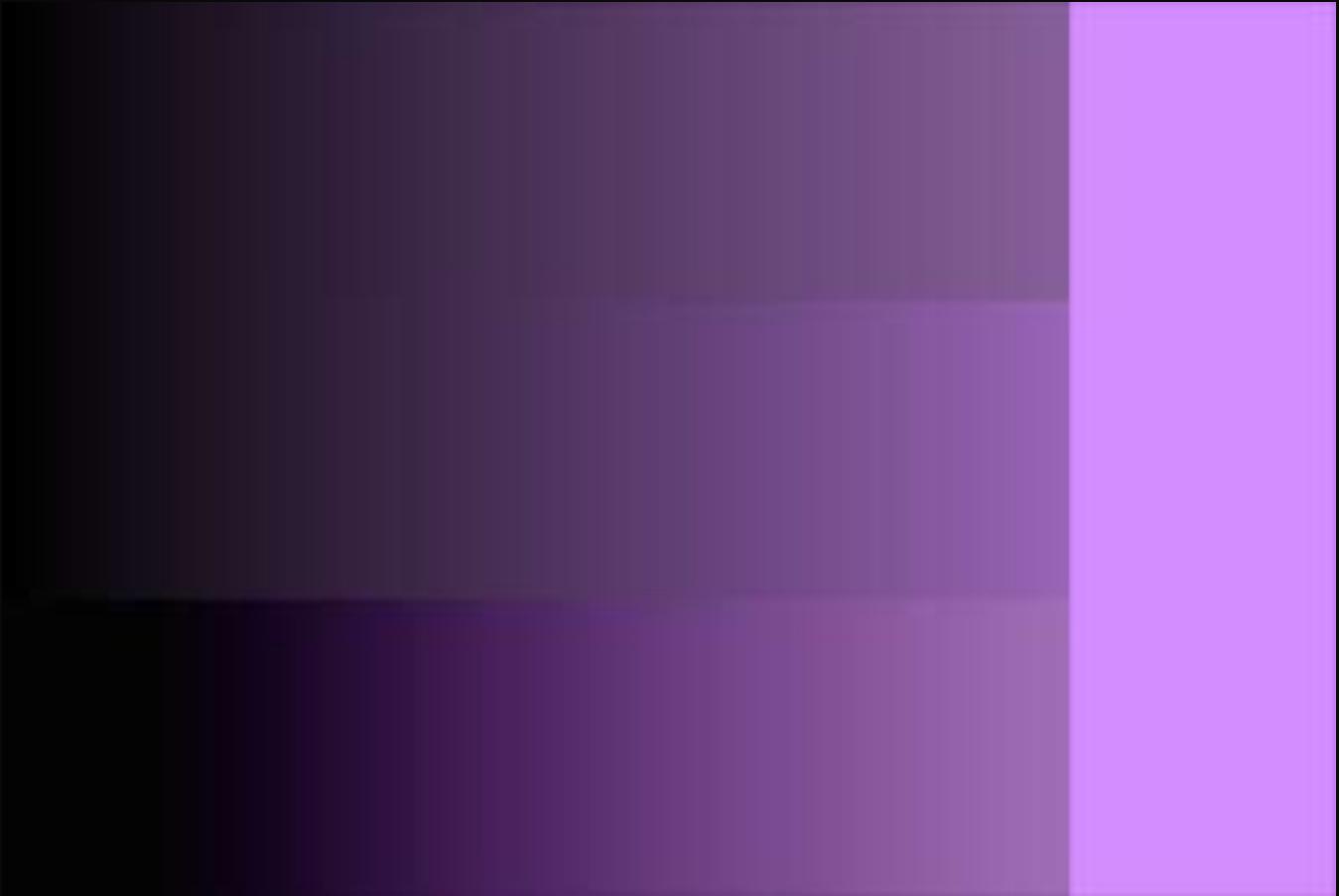
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

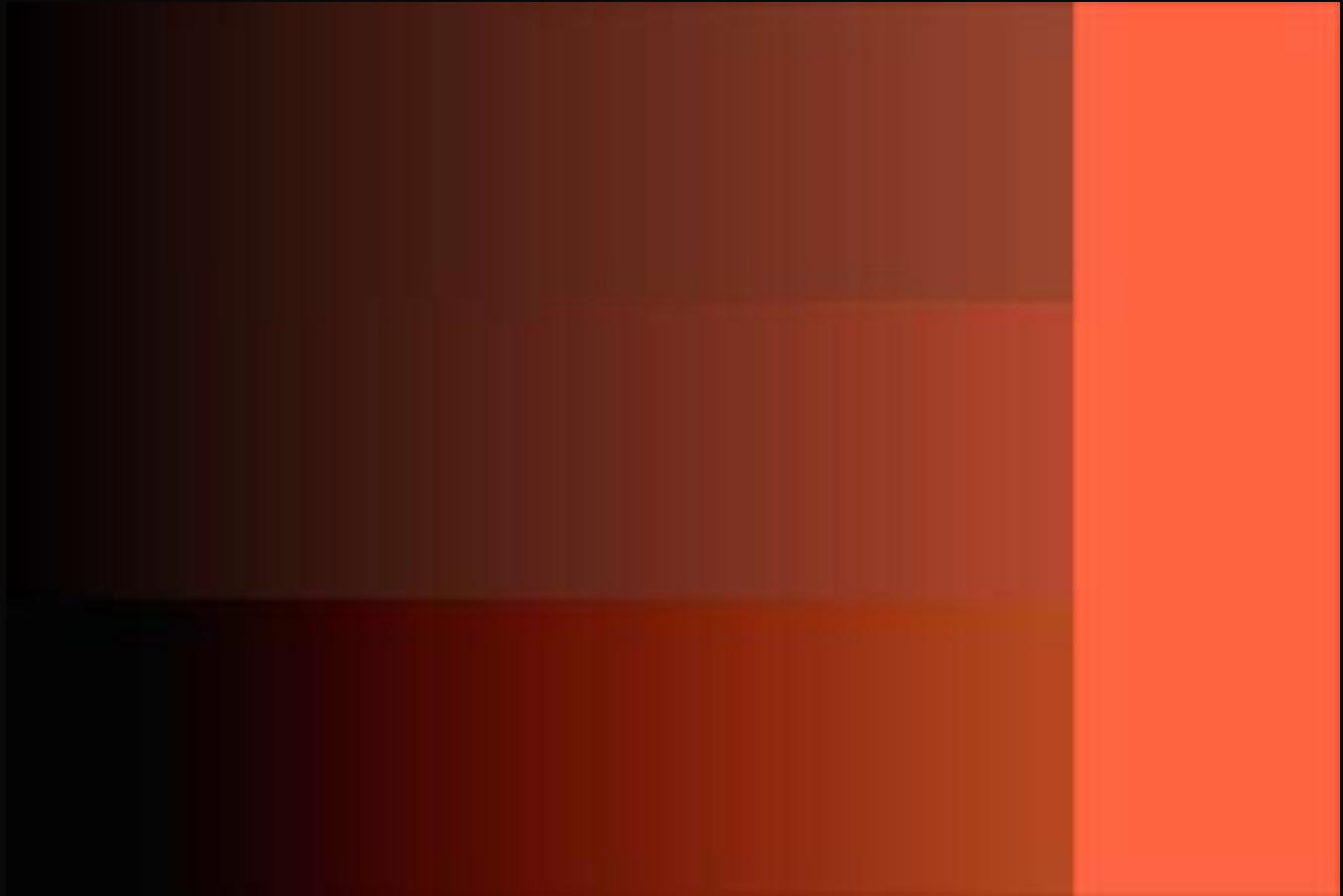
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

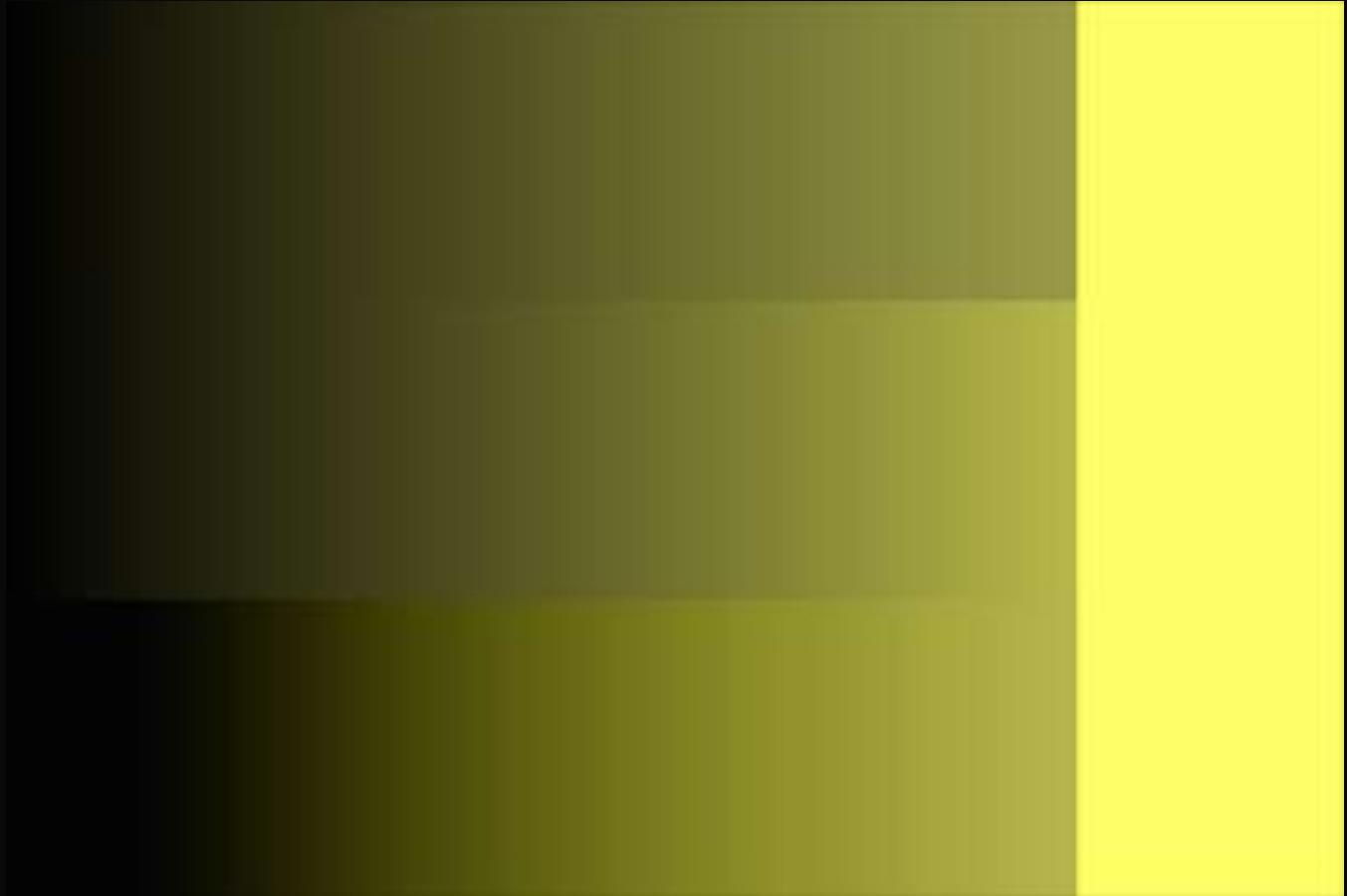
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

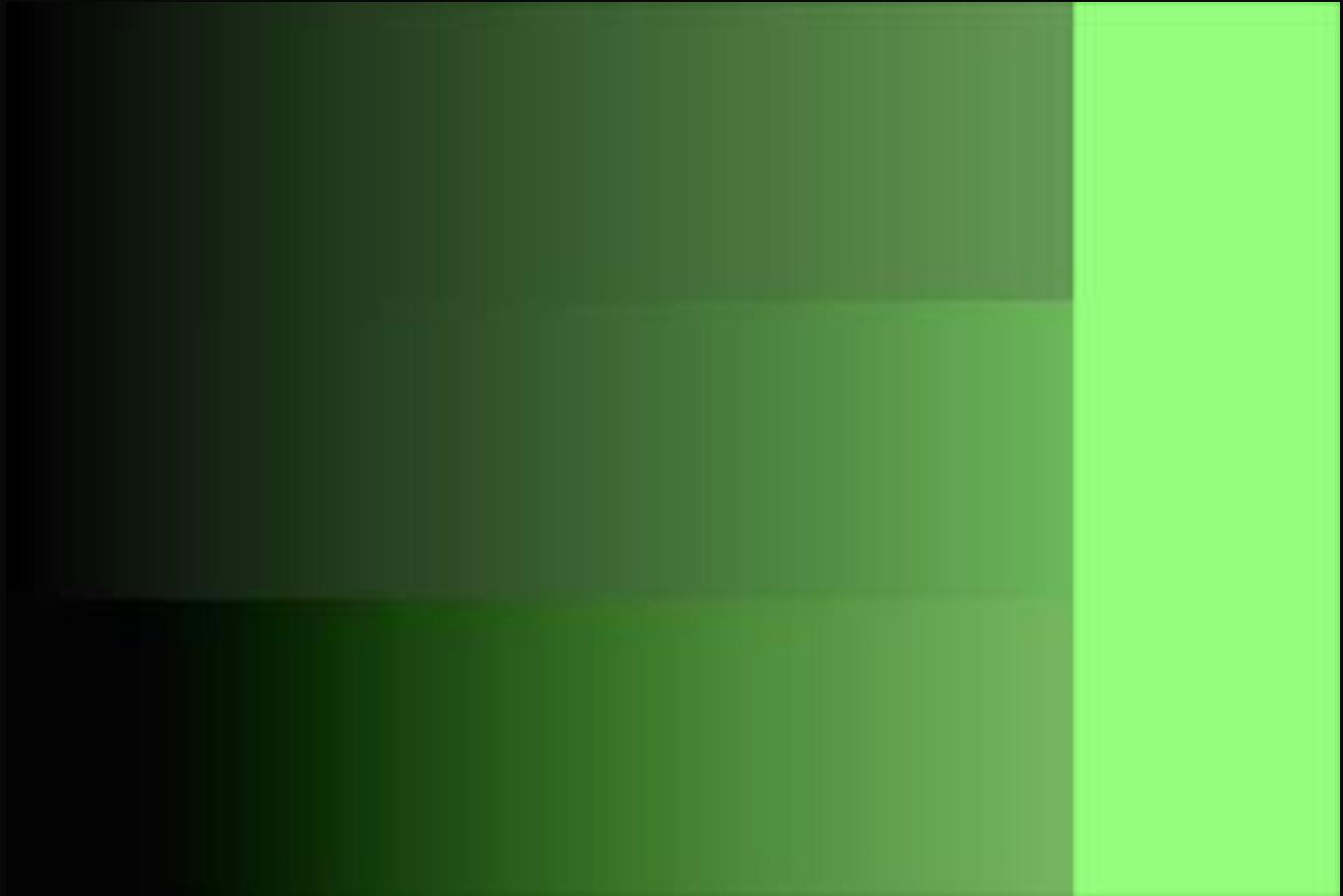
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

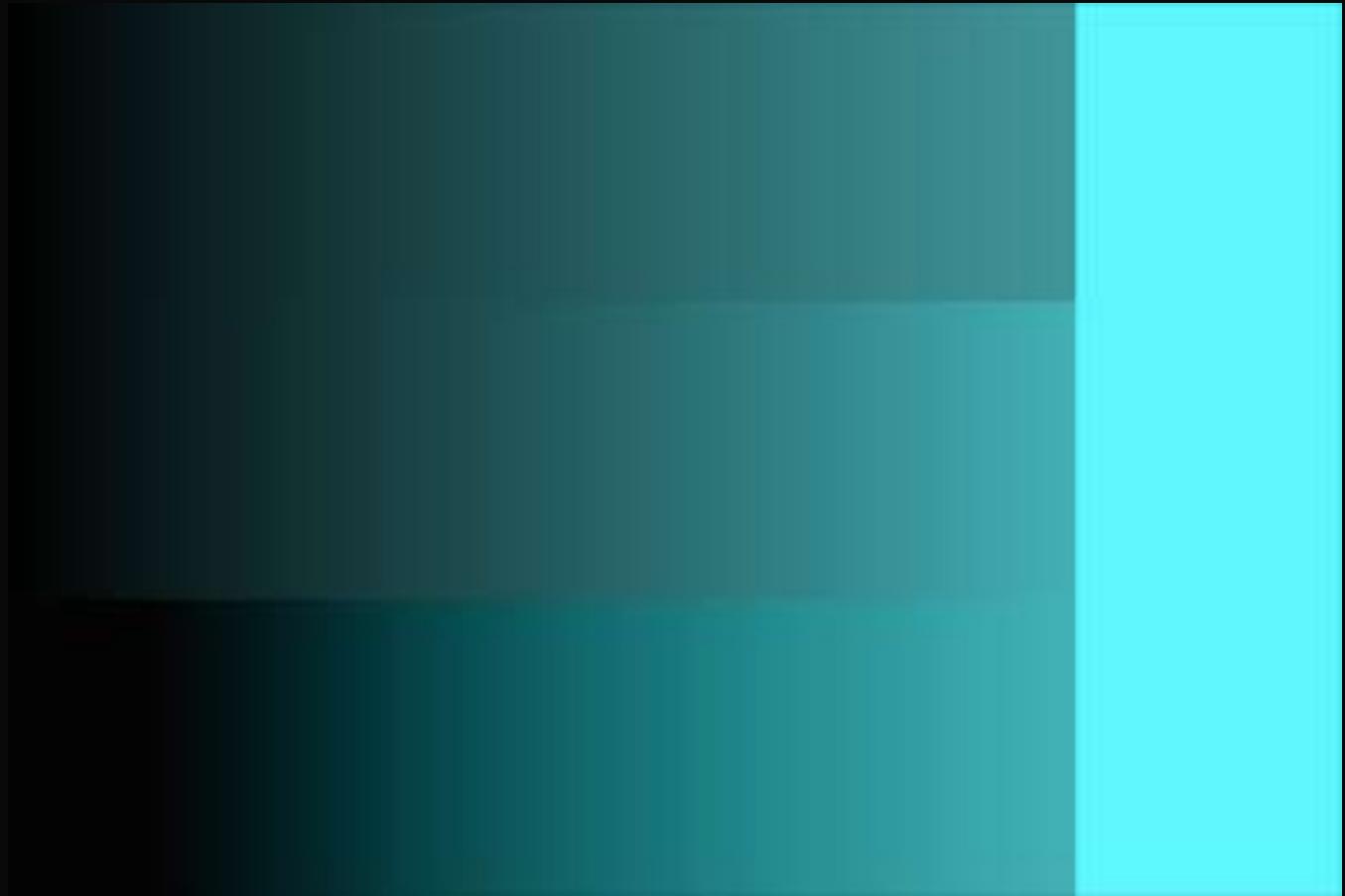
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

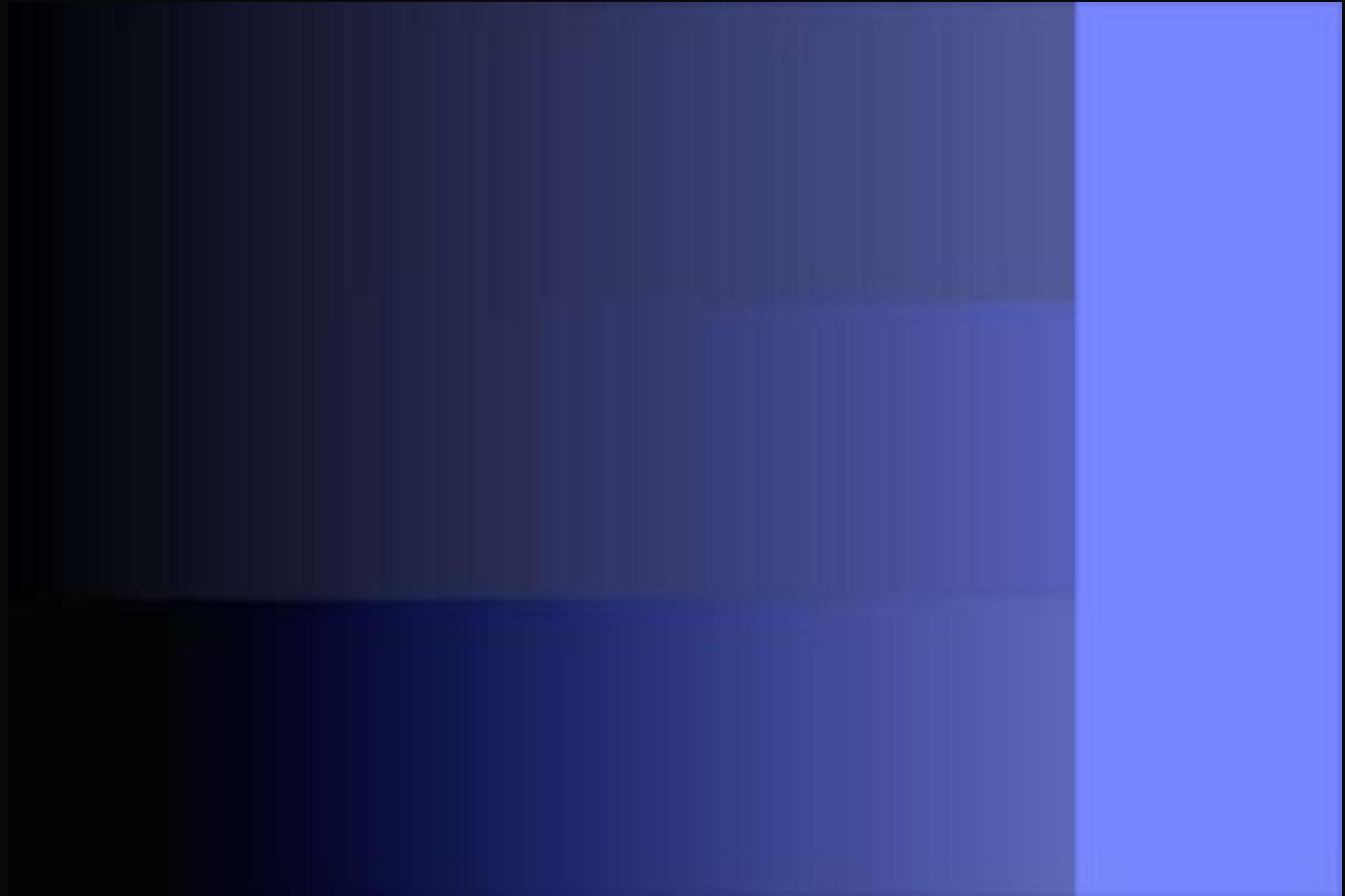
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

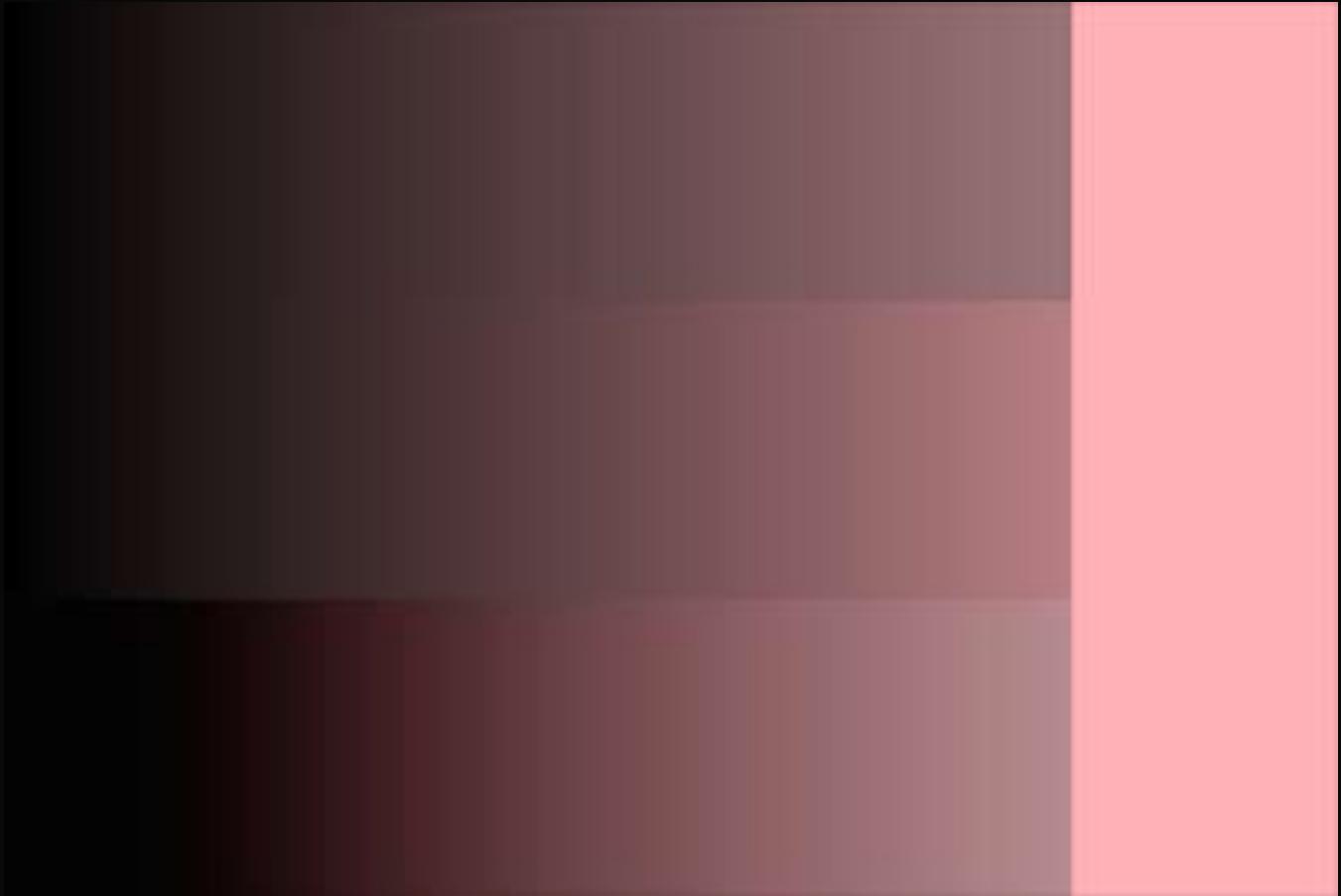
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

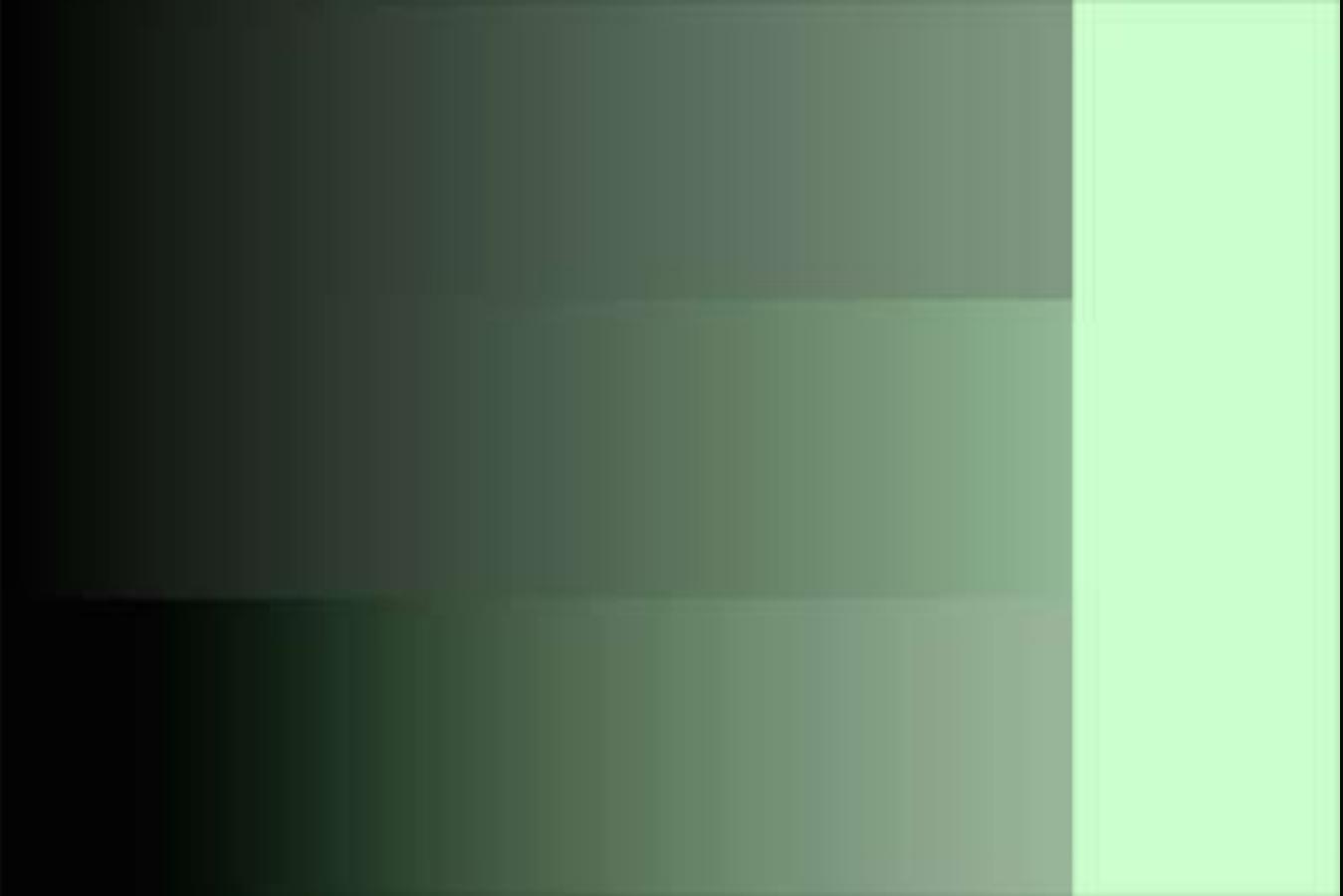
---

- This technique works with any color.

Reinhard

Linear

Filmic



# Filmic Tonemapping

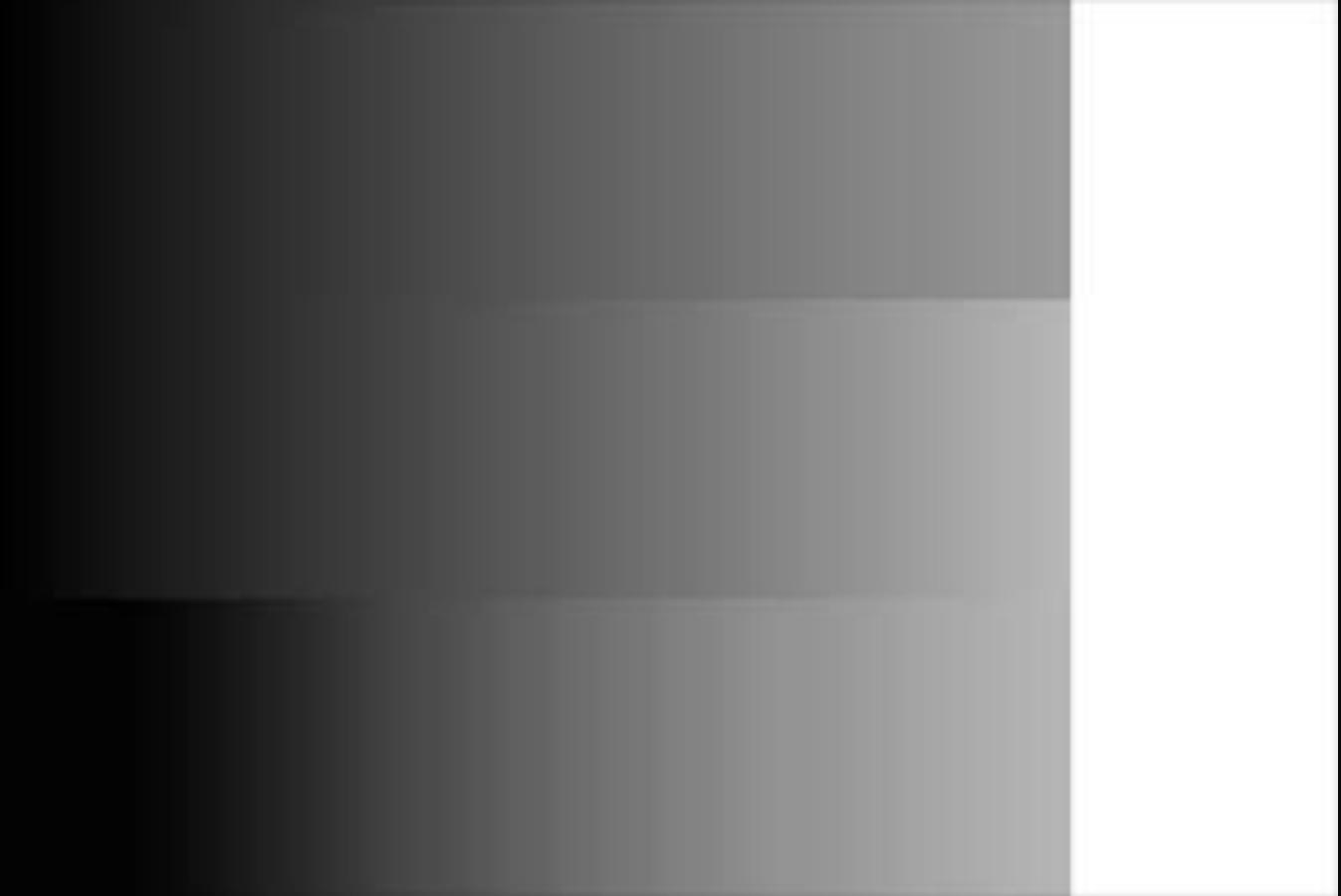
---

- This technique works with any color.

Reinhard

Linear

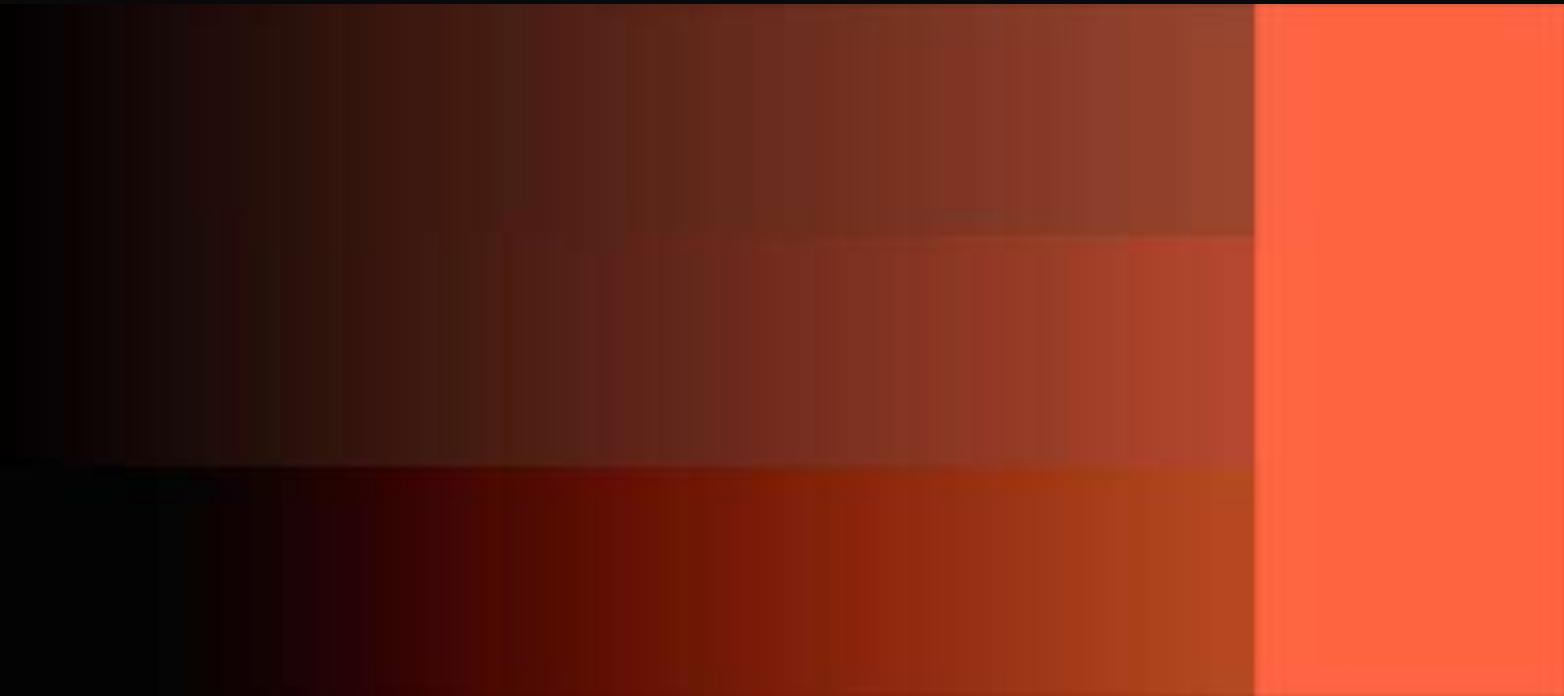
Filmic



# Filmic Tonemapping

---

- In terms of “Crisp Blacks” vs. “Milky Blacks”
  - Filmic > Linear > Reinhard
- In terms of “Soft Highlights” vs. “Clamped Highlights”
  - Filmic = Reinhard > Linear



# Filmic Tonemapping

---

- Linear.



# Filmic Tonemapping

---

- Fade from linear to filmic.



# Filmic Tonemapping

---

- Fade from linear to filmic.



# Filmic Tonemapping

---

- Fade from linear to filmic.



# Filmic Tonemapping

---

- Fade from linear to filmic.



# Filmic Tonemapping

---

- Fade from linear to filmic.



# Filmic Tonemapping

---

- Fade from linear to filmic.



# Filmic vs Reinhard

---

- Fade from linear to Reinhard.

# Filmic Tonemapping

---

- Fade from Reinhard to filmic.



# Filmic Tonemapping

---

- Fade from Reinhard to filmic.



# Filmic Tonemapping

---

- Fade from Reinhard to filmic.



# Filmic Tonemapping

---

- Fade from Reinhard to filmic.



# Filmic Tonemapping

---

- Fade from Reinhard to filmic.



# Filmic Tonemapping

---

- Fade from Reinhard to filmic.



# Filmic Tonemapping

---

- Let's do a comparison.

# Filmic Tonemapping

---

- This is with a linear curve.



# Filmic Tonemapping

---

- And here is filmic.



# Filmic Tonemapping

---

- First, we'll go up.

# Filmic Tonemapping

---

- Exposure 0



# Filmic Tonemapping

---

- Exposure +1



# Filmic Tonemapping

---

- Exposure +2



# Filmic Tonemapping

---

- Exposure +3



# Filmic Tonemapping

---

- Exposure +4



# Filmic Tonemapping

---

- Ok, now that's bad.
- What happens if we go down?
- Notice the crisper blacks in the filmic version.

# Filmic Tonemapping

---

- Exposure: 0



# Filmic Tonemapping

---

- Exposure: -1



# Filmic Tonemapping

---

- Exposure: -2



# Filmic Tonemapping

---

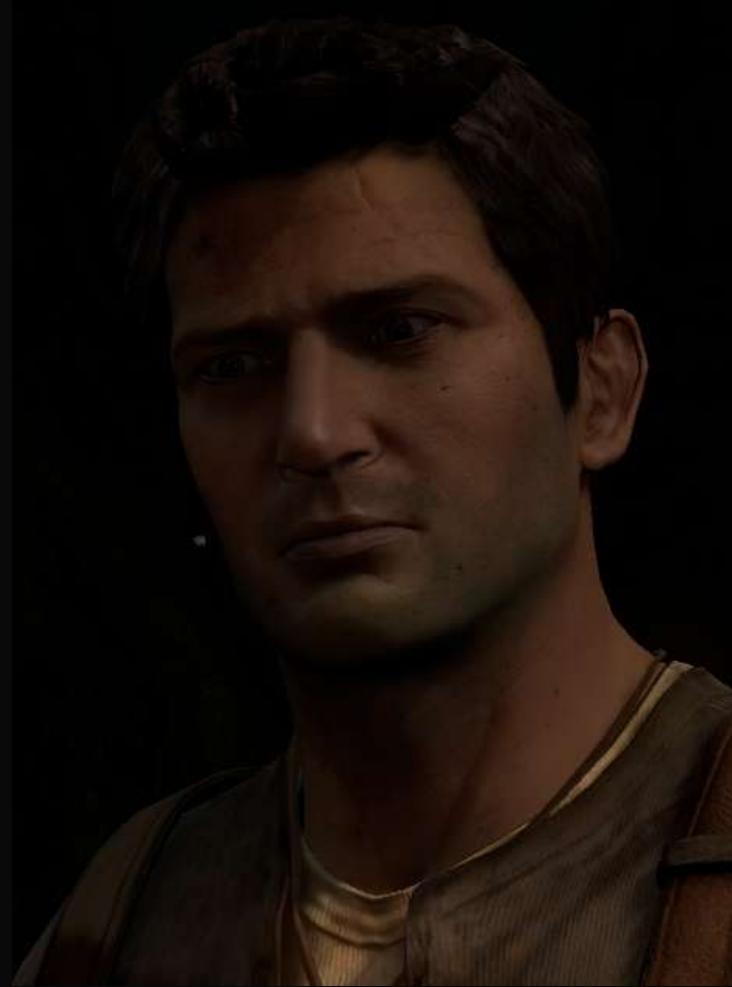
- Exposure: -3



# Filmic Tonemapping

---

- Exposure: -4



# Filmic Tonemapping

---

- Colors get crisper and more saturated at the bottom end.

# Filmic Tonemapping

---

```
float3 Id = 0.002;
float linReference = 0.18;
float logReference = 444;
float logGamma = 0.45;

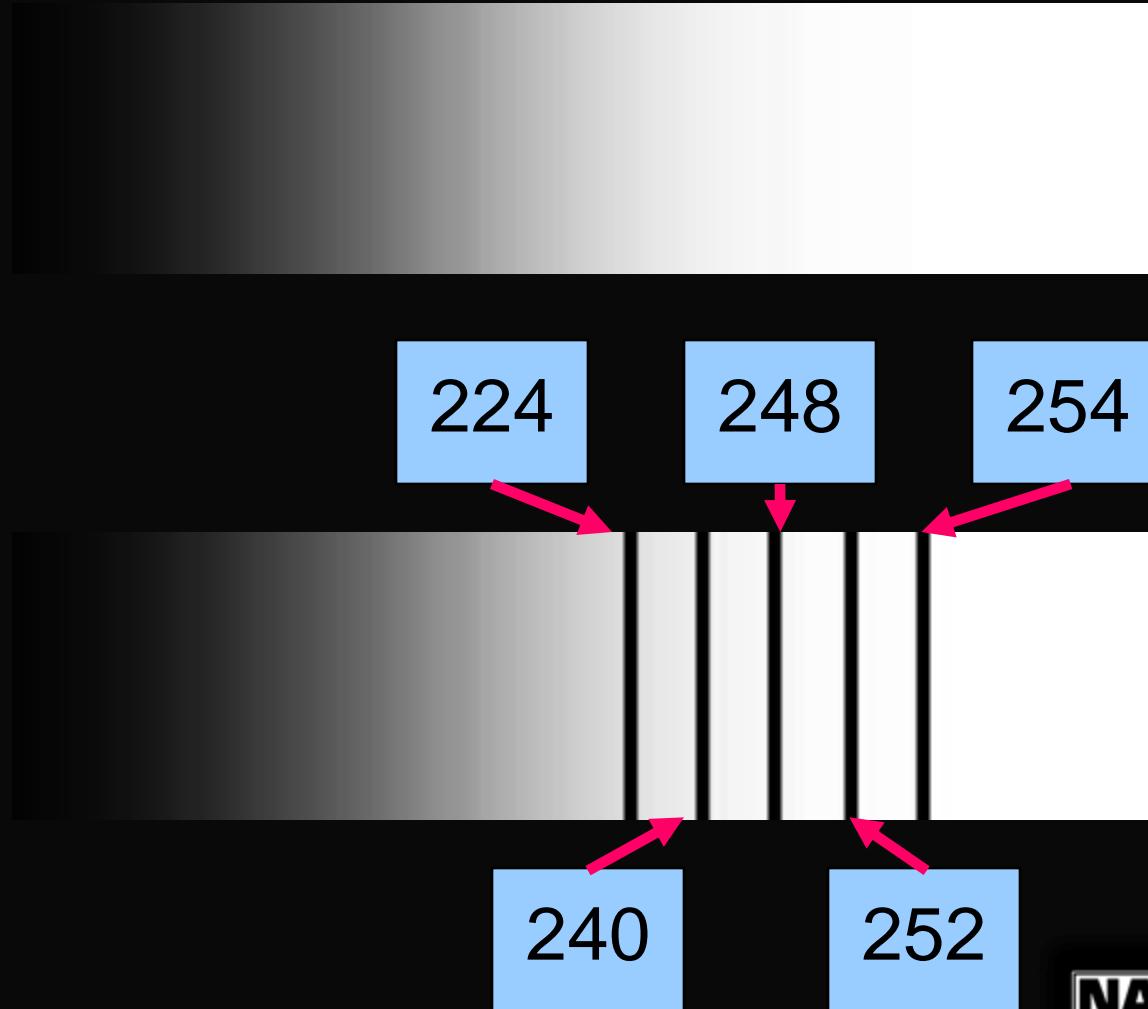
outColor.rgb = (log10(0.4*outColor.rgb/linReference)/Id*logGamma +
    logReference)/1023.f;
outColor.rgb = clamp(outColor.rgb , 0, 1);

float FilmLutWidth = 256;
float Padding = .5/FilmLutWidth;

outColor.r = tex2D(FilmLutSampler, float2( lerp(Padding,1-Padding,outColor.r), .5)).r;
outColor.g = tex2D(FilmLutSampler, float2( lerp(Padding,1-Padding,outColor.g), .5)).r;
outColor.b = tex2D(FilmLutSampler, float2( lerp(Padding,1-Padding,outColor.b), .5)).r;
```

# Filmic Tonemapping

- Variation in white section:



# More Magic

---

- The catch:
  - Three texture lookups
- Jim Hejl to the rescue
  - Principle Member of Technical Staff
  - GPU Group, Office of the CTO
  - AMD Research
- Used to work for EA

# More Magic

---

- Awesome approximation with only ALU ops.
- Replaces the entire Lin/Log and Texture LUT
- Includes the  $\text{pow}(x, 1/2.2)$

```
x = max(0, LinearColor-0.004);  
GammaColor = (x*(6.2*x+0.5))/(x*(6.2*x+1.7)+0.06);
```

# Filmic Tonemapping

---

- Nice curve
- Toe arguably too strong
  - Most monitors are too contrasty, strong toe makes it worse
- May want less shoulder
  - The more range, the more shoulder
  - If your scene lacks range, you want to tone down the shoulder a bit

# Filmic Tonemapping

---

A = Shoulder Strength

B = Linear Strength

C = Linear Angle

D = Toe Strength

E = Toe Numerator

F = Toe Denominator

Note: E/F = Toe Angle

LinearWhite = Linear White Point Value

$F(x) = ((x * (A * x + C * B) + D * E) / (x * (A * x + B) + D * F)) - E/F;$

FinalColor =  $F(\text{LinearColor}) / F(\text{LinearWhite})$

# Filmic Tonemapping

---

Shoulder Strength = 0.22

Linear Strength = 0.30

Linear Angle = 0.10

Toe Strength = 0.20

Toe Numerator = 0.01

Toe Denominator = 0.30

Linear White Point Value = 11.2

These numbers DO NOT have the  $\text{pow}(x, 1/2.2)$  baked in

# Conclusions

---

- Filmic Tonemapping
  - IMO: The most important post effect
  - Changes your life completely
    - Once you have it, you can't live without it
  - If you implement it, you have to redo all your lighting
    - You can't just switch it on if your lighting is tweaked for no dynamic range.

# Filmic Tonemapping

---

“At least I’m not getting choked in gamma space...”



# Part 3: SSAO

---



# SSAO Time

---

- That was fun.
- Time for stage 3.
- Let's talk about why you need AO.

# Why AO?

---

- The shadow “grounds” the car.



# Why AO?

---

- Unless you're already in the shadow!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# How important is it?

---

- The shadow from the sun grounds object.
- But if you are already in shadow, you need something else...
- It's the AO that grounds the car in those shots.
- So what if we took the AO out?
  - Trick taught to me by Habib (the guy with the awesome condo earlier)

# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

- Your shadow can't help you now!



# Why AO?

---

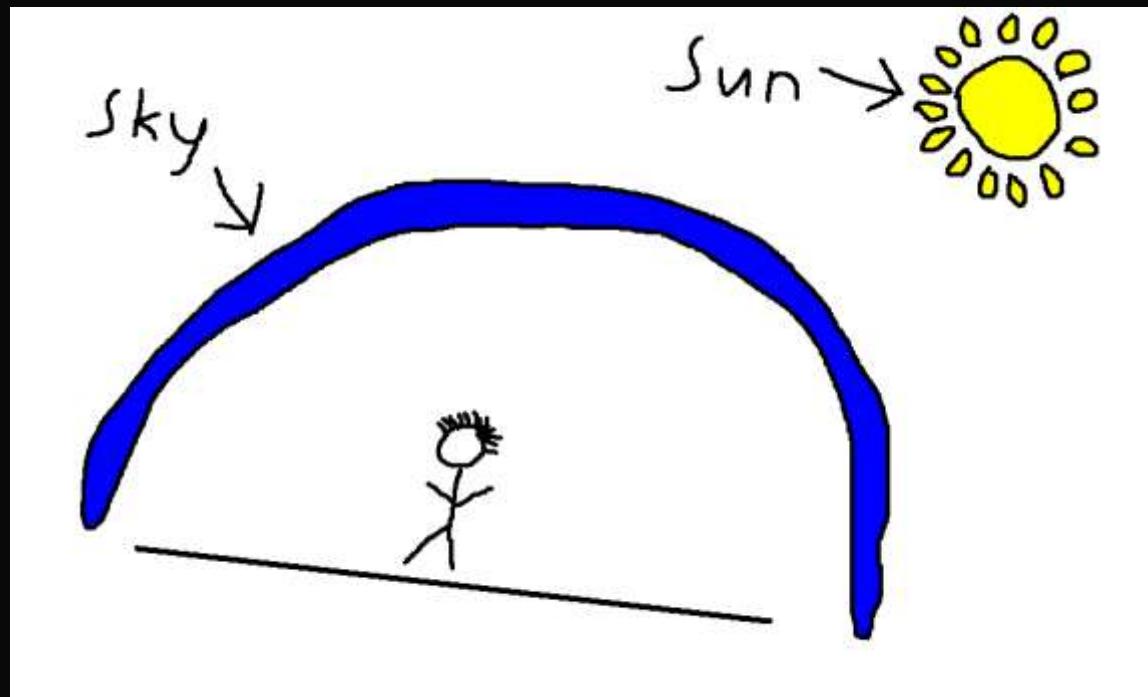
- Your shadow can't help you now!



# How to use AO?

---

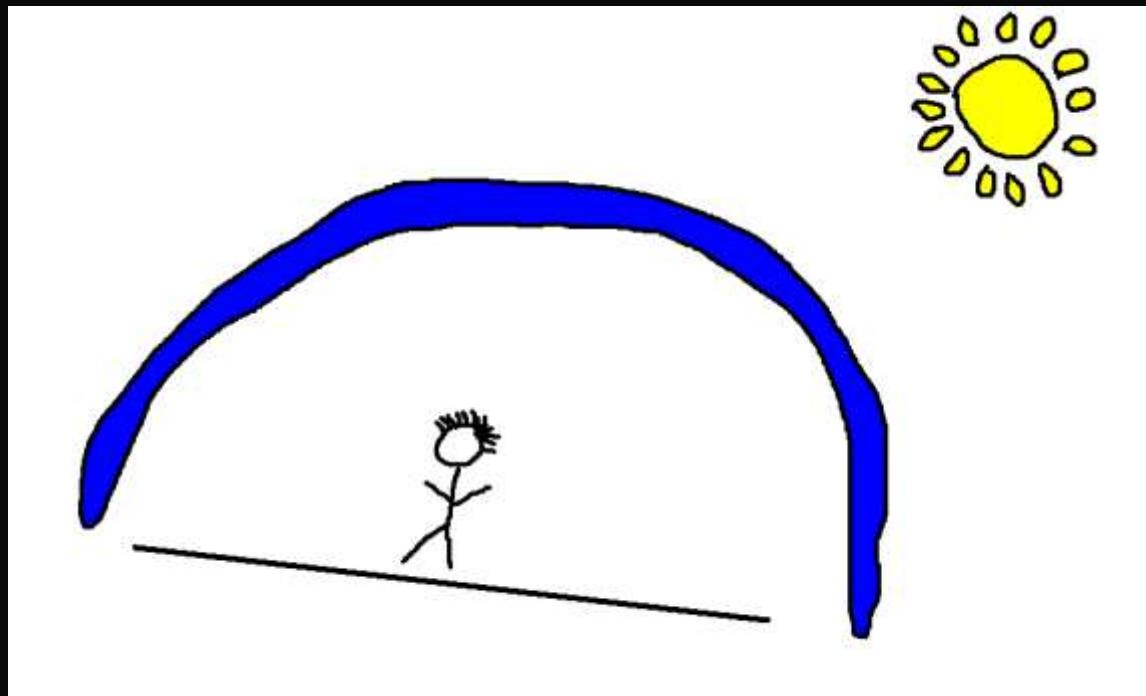
- Sun is Directional Light
- Sky is Hemisphere Light (ish)



# How to use AO?

---

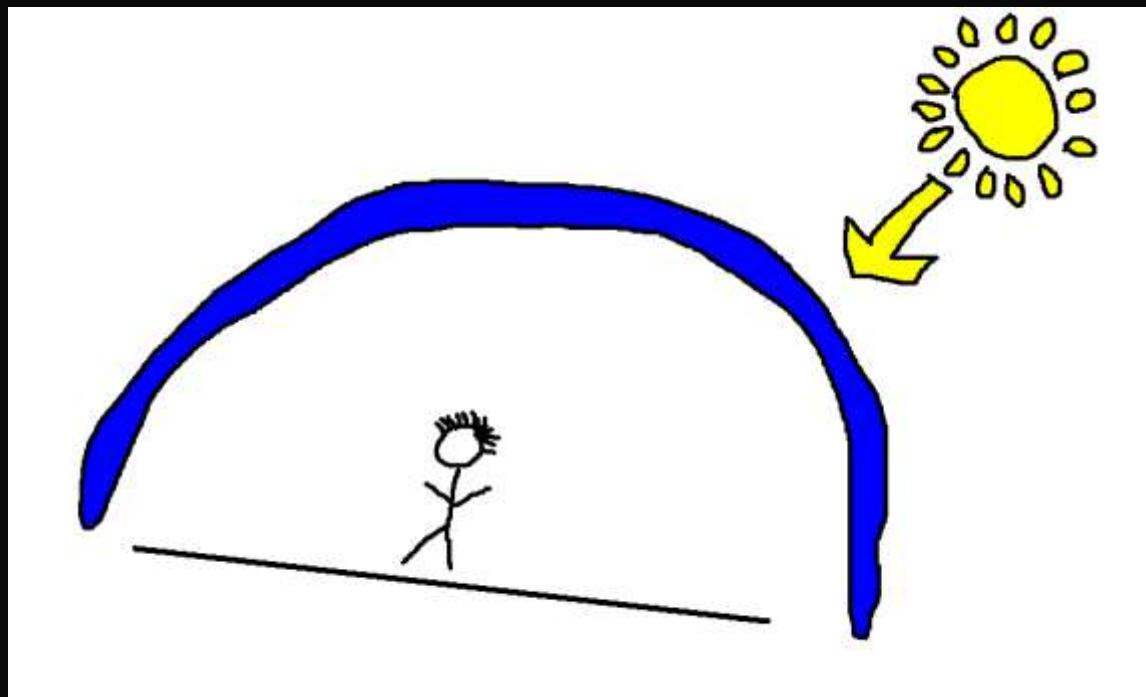
- Sun is Yellow (ish)
- Sky is Blue (ish)



# How to use AO?

---

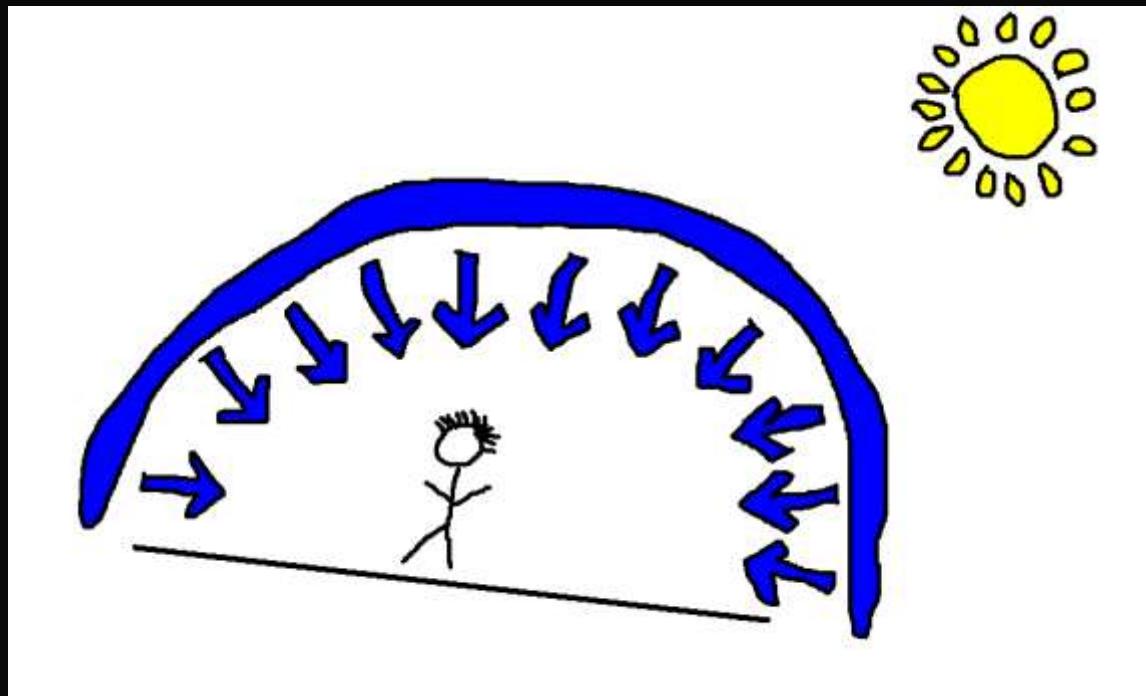
- Sun is Directional
- Approximate Directional Shadow with Shadow Mapping



# How to use AO?

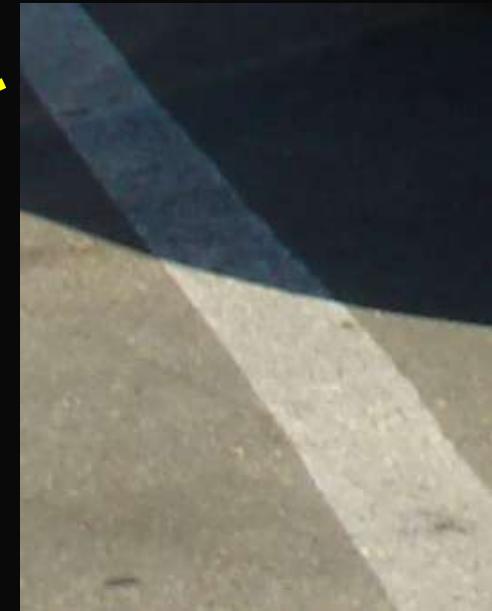
---

- Sky is Hemisphere (sorta)
- Approximate Hemisphere Shadow with AO



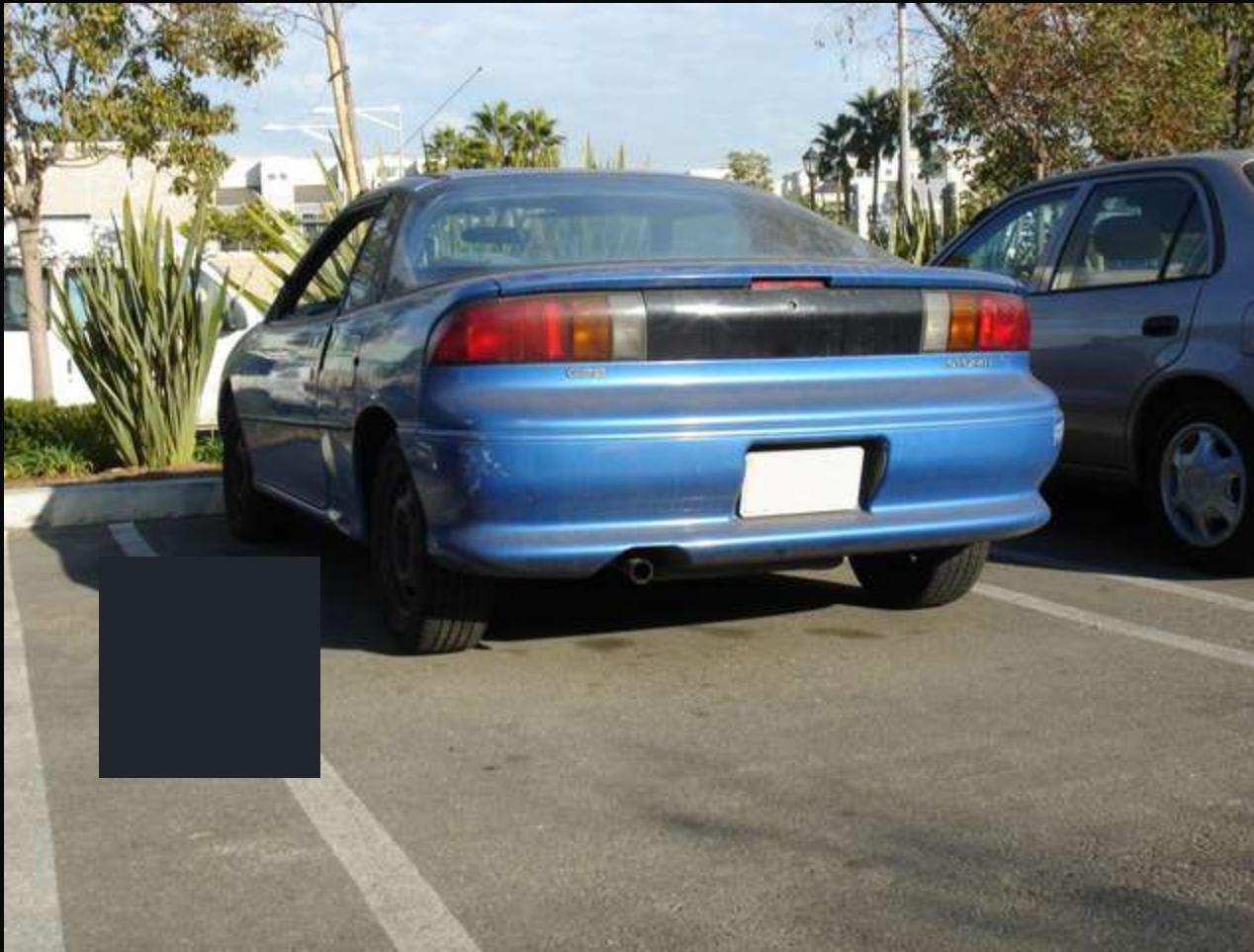
# How to use AO?

- Sun is yellow, sky is blue.



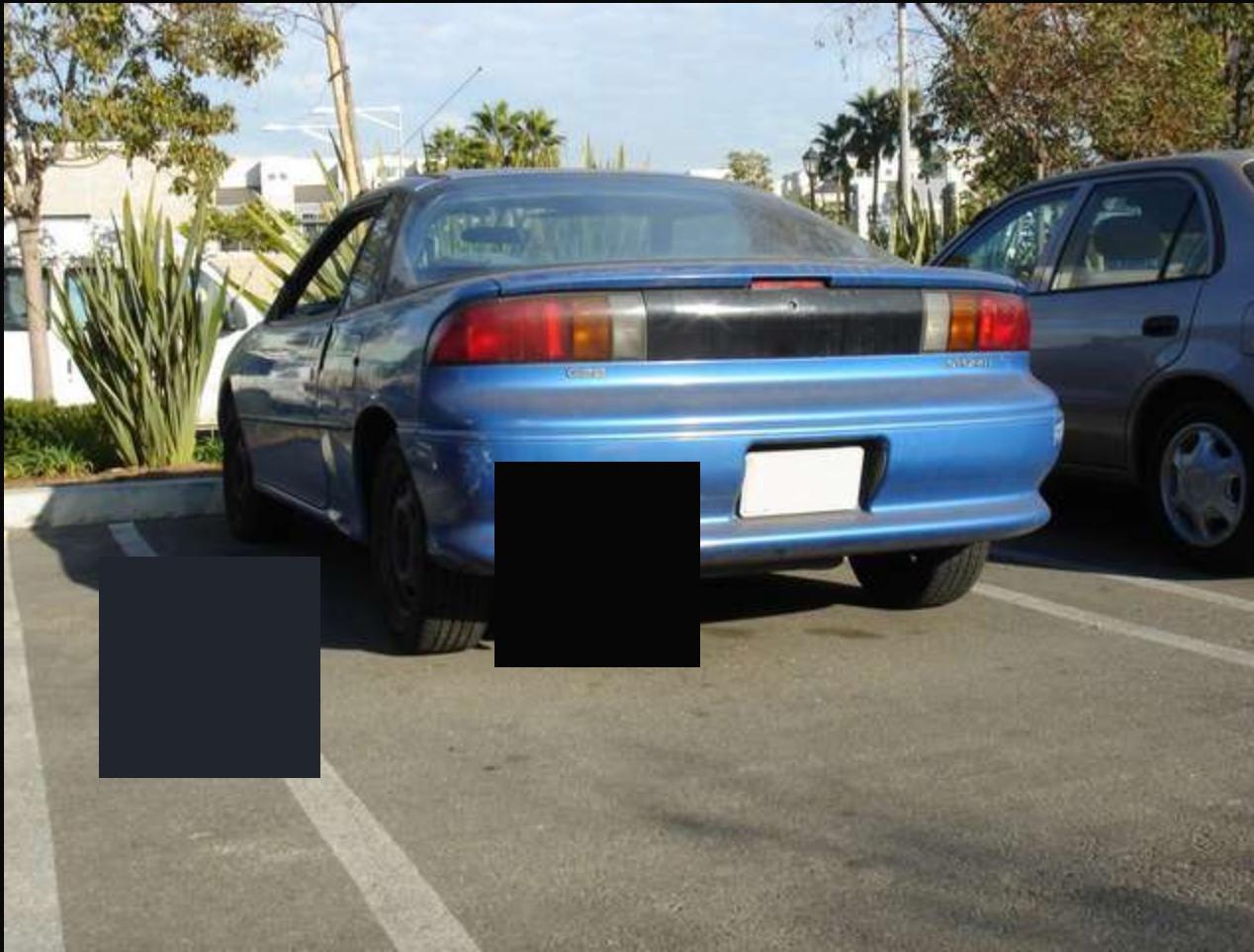
# How to use AO?

---



# How to use AO?

---



# How to use AO?

---



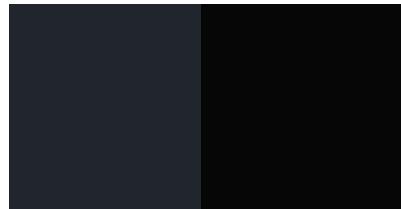
# How to use AO?

---



# How to use AO?

---



# Sunlight

---

Q) Why not have it affect sunlight?

A) It does the exact opposite of what you want it to do.

# SSAO Example

---

- Off



# SSAO Example

---

- On



# SSAO Example

---

- SSAO Darkens Shadows – Good
- SSAO Darkens Sunlight – BAD
  - Reduces perceived contrast
  - Makes lighting look flatter
- My Opinion: SSAO should NOT affect Direct Light



# The Algorithm

---

- Only Input is Half-Res Depth
- No normals
- No special filtering (I.e. Bilateral)
- Processed in 64x64 blocks
- Adjacent Depth tiles included as well

# The Algorithm

---

- Calculate Base AO
- Dilate Horizontal
- Dilate Vertical
- Apply Low Pass Filter

# SSAO Passes

---

0) Base Image



# SSAO Passes

---

1) Base SSAO



# SSAO Passes

---

2) Dilate Horizontal



# SSAO Passes

---

3) Dilate Vertical



# SSAO Passes

---

- 4) Low Pass Filter  
(I.e. blur the \$&%@ out of it)



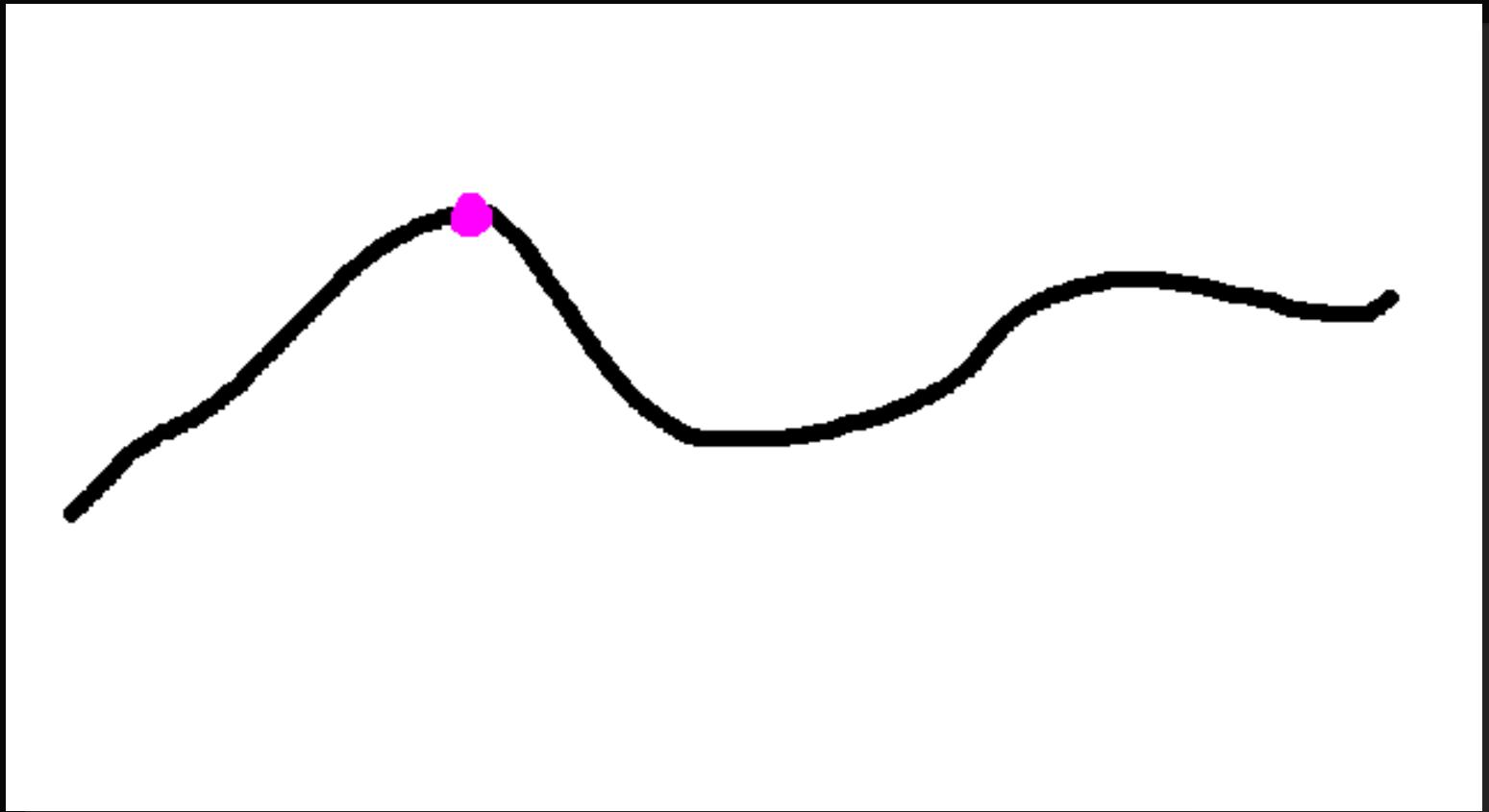
# SSAO Passes

---

- Step One: The Base AO
- Uses half-res depth buffer
- Calculate AO based on nearby depth samples

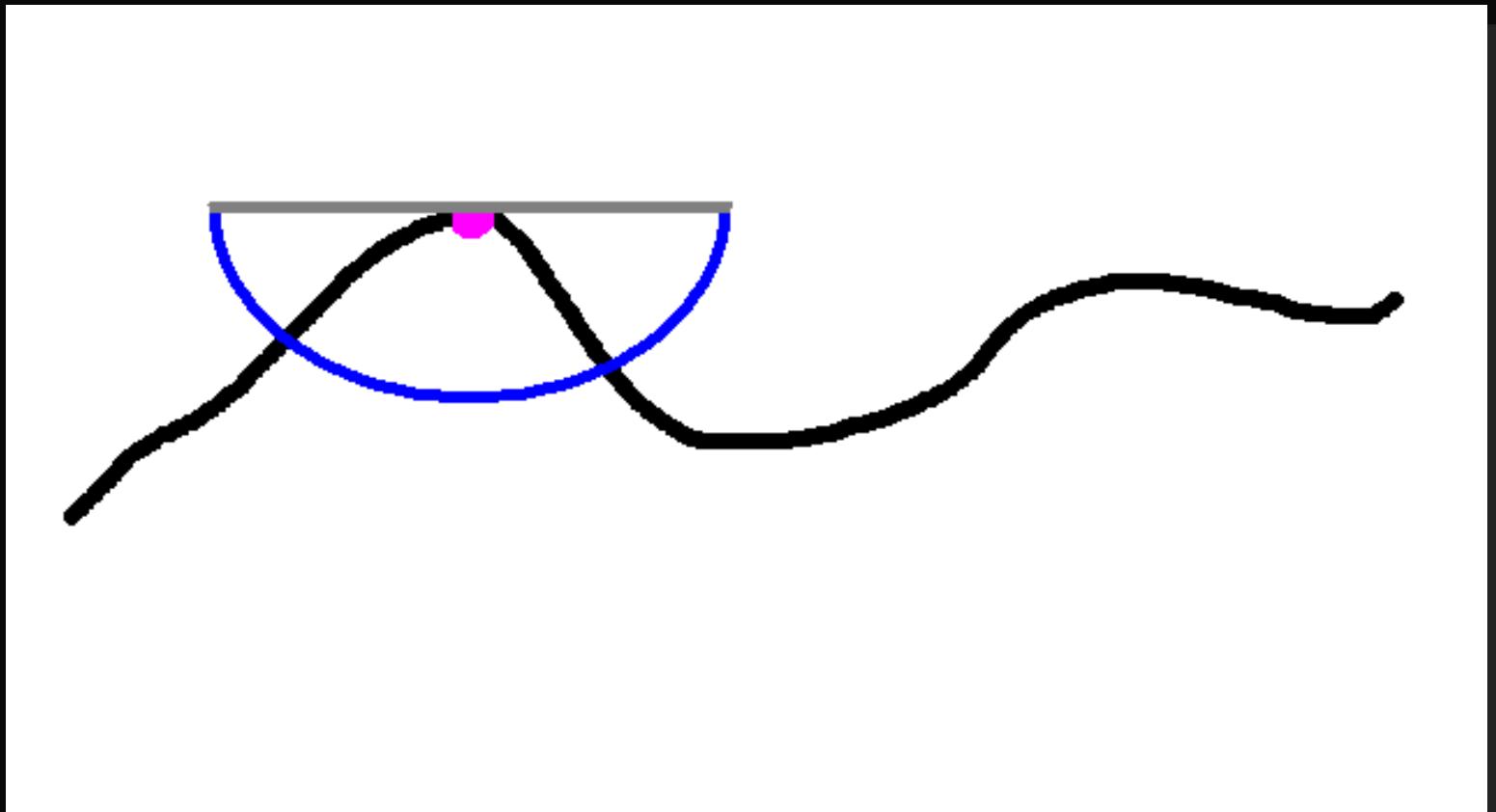
# SSAO Passes

- Pink point on a black surface.



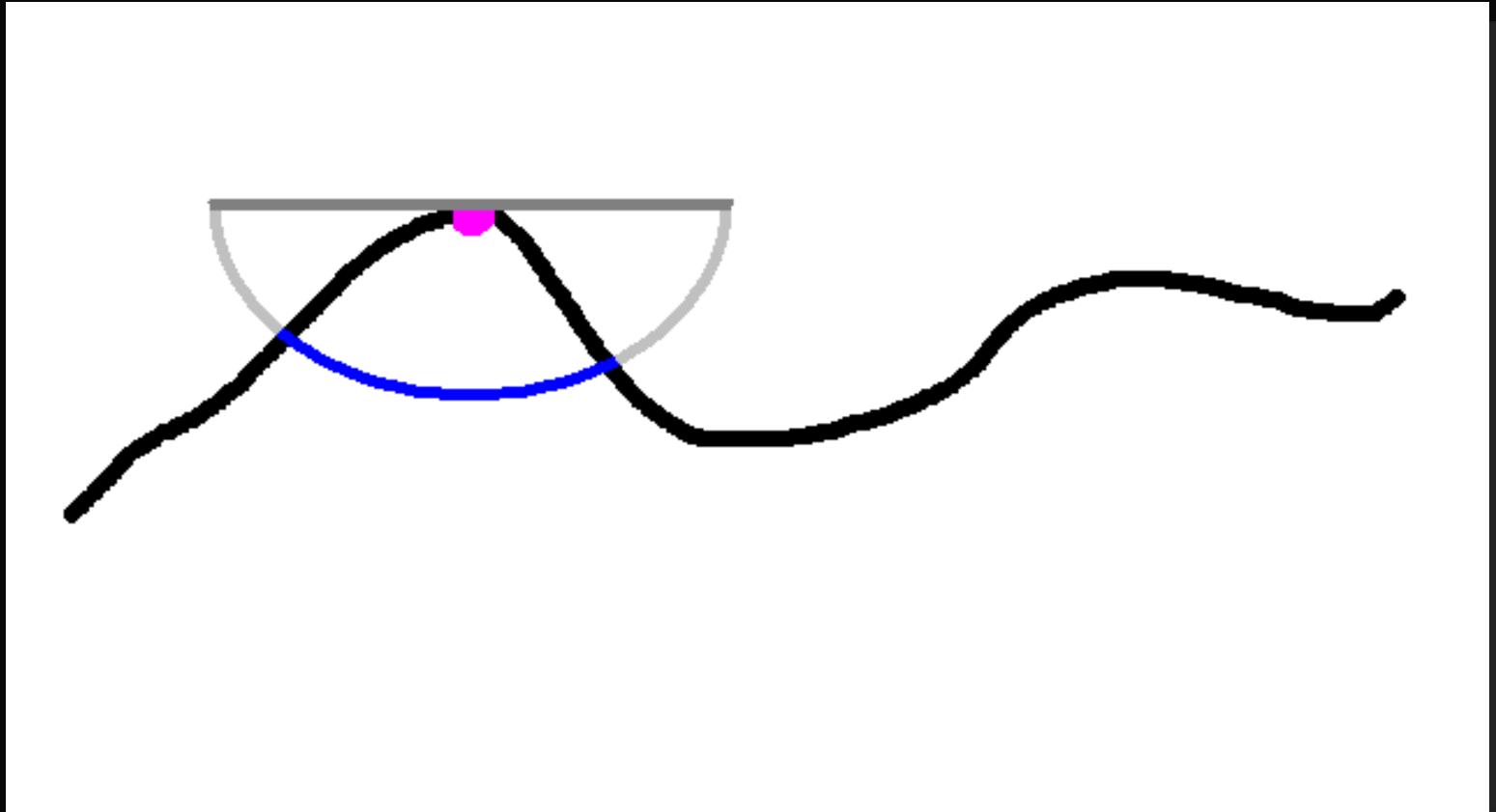
# SSAO Passes

- Correct way is to look at the hemisphere...



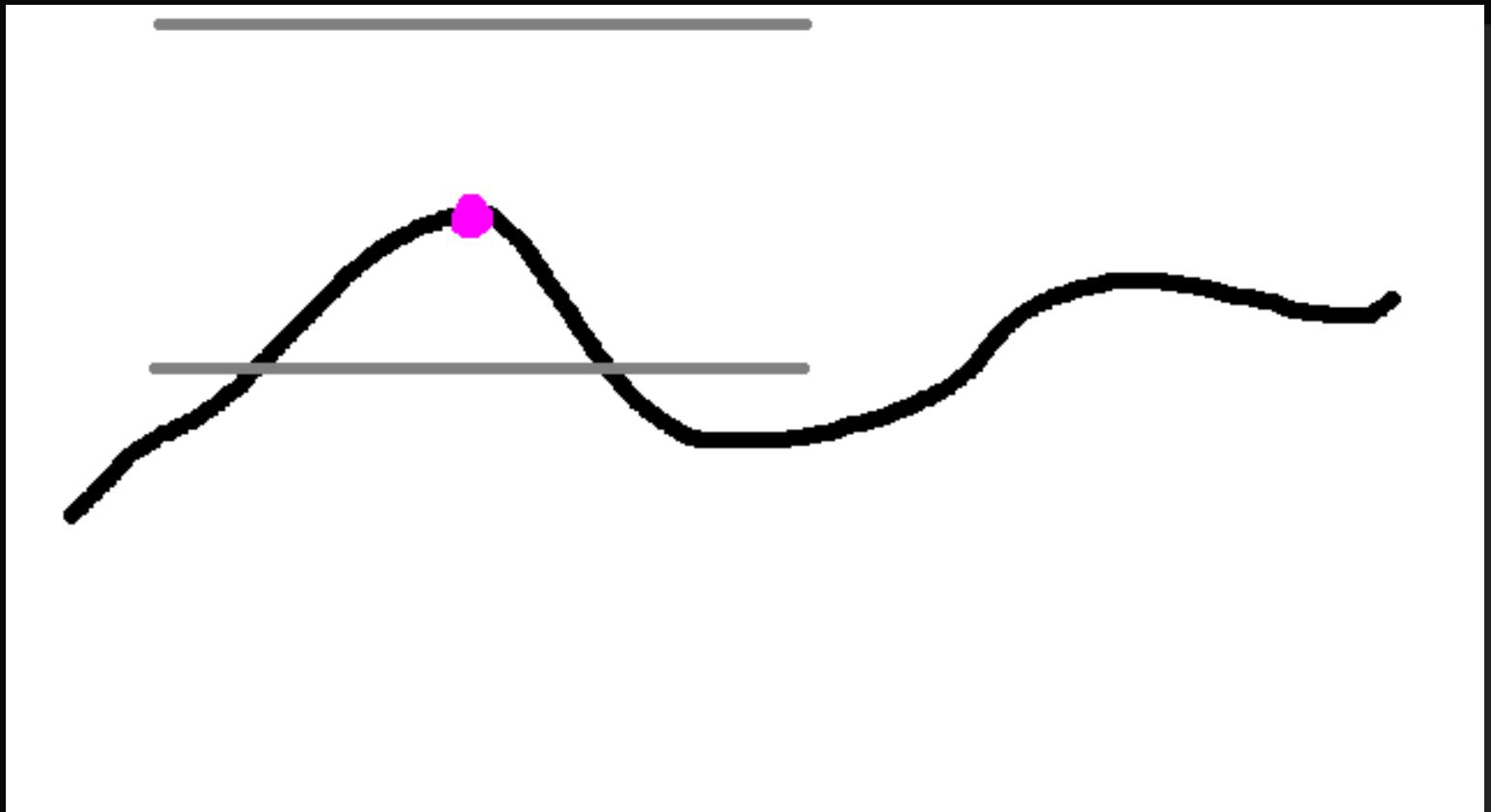
# SSAO Passes

- ...and find the area that is visible to the point.
- Our solution is much hackier.



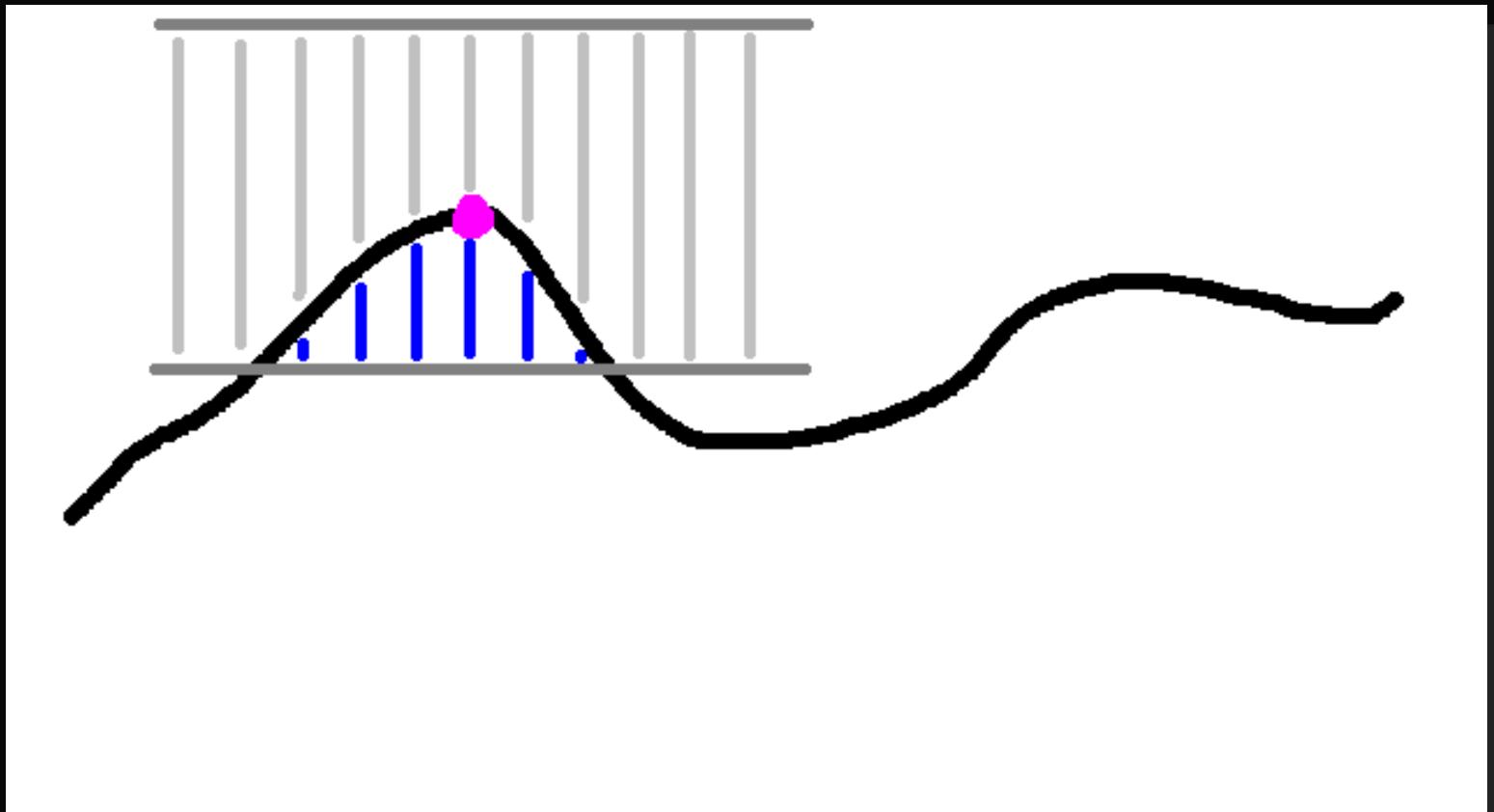
# SSAO Passes

- Who needs sphere? A box will do just as well.



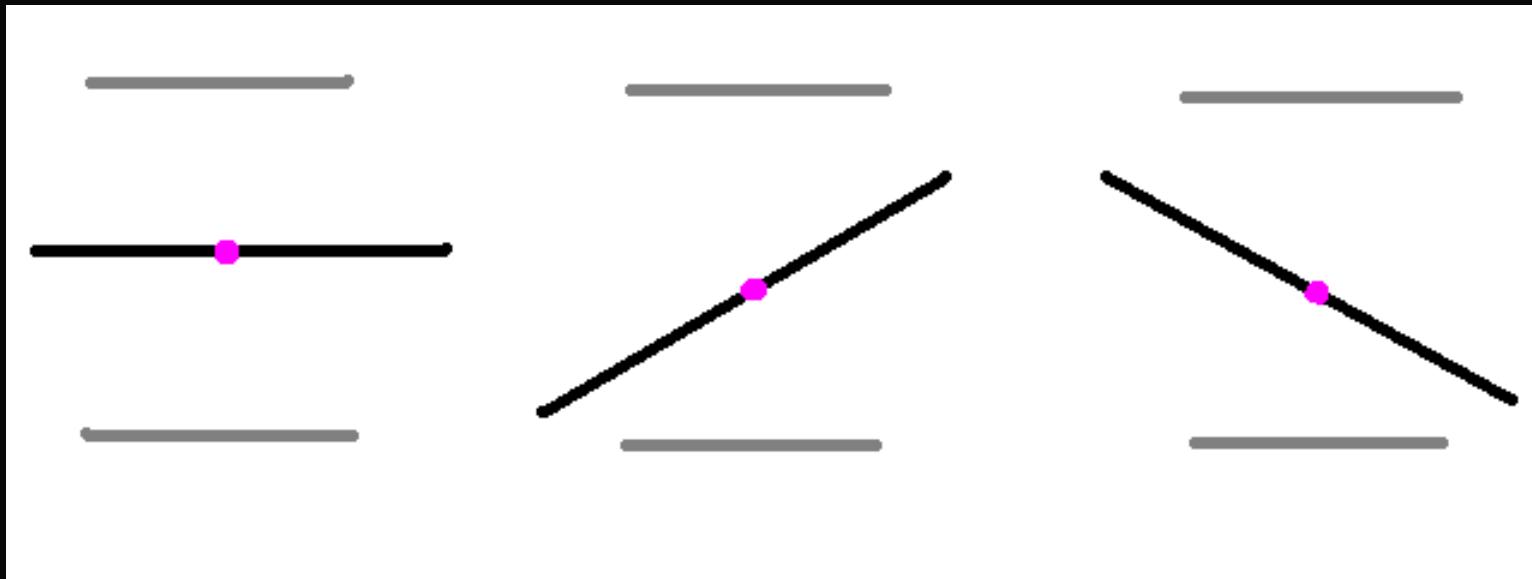
# SSAO Passes

- Instead of area though, we will approximate volume.



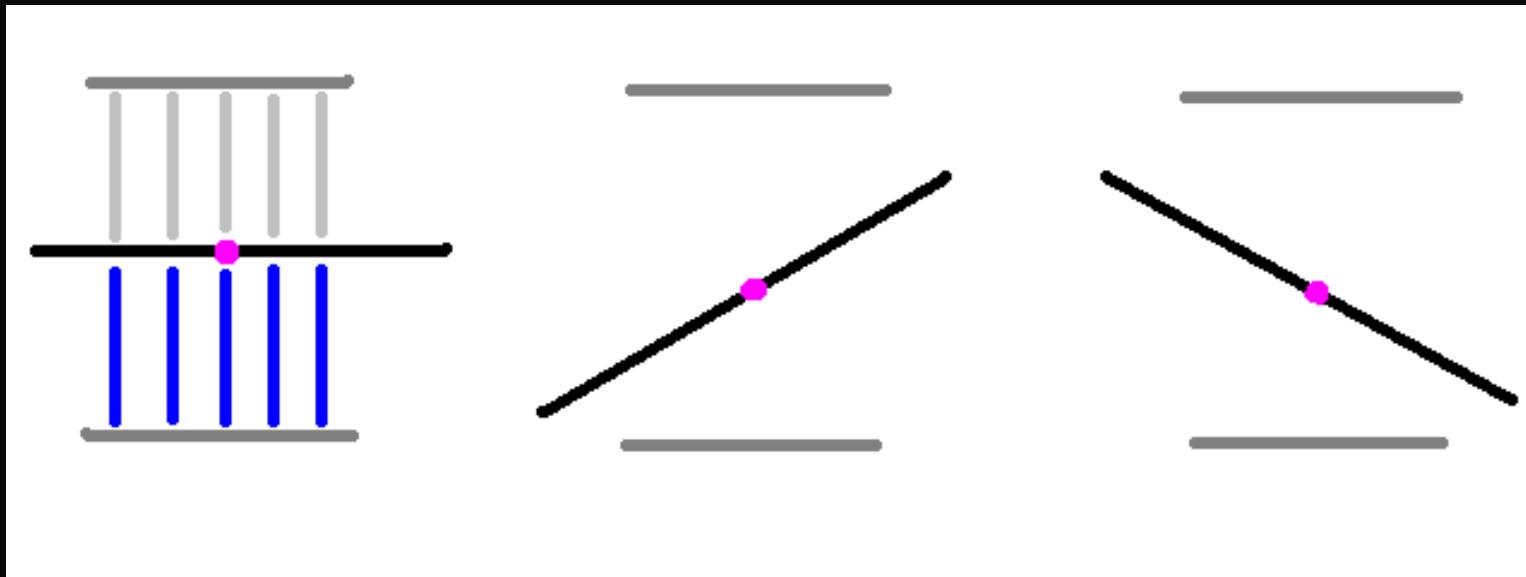
# SSAO Passes

- We can estimate volume without knowing the normal.
- Half Volume = Fully Unoccluded



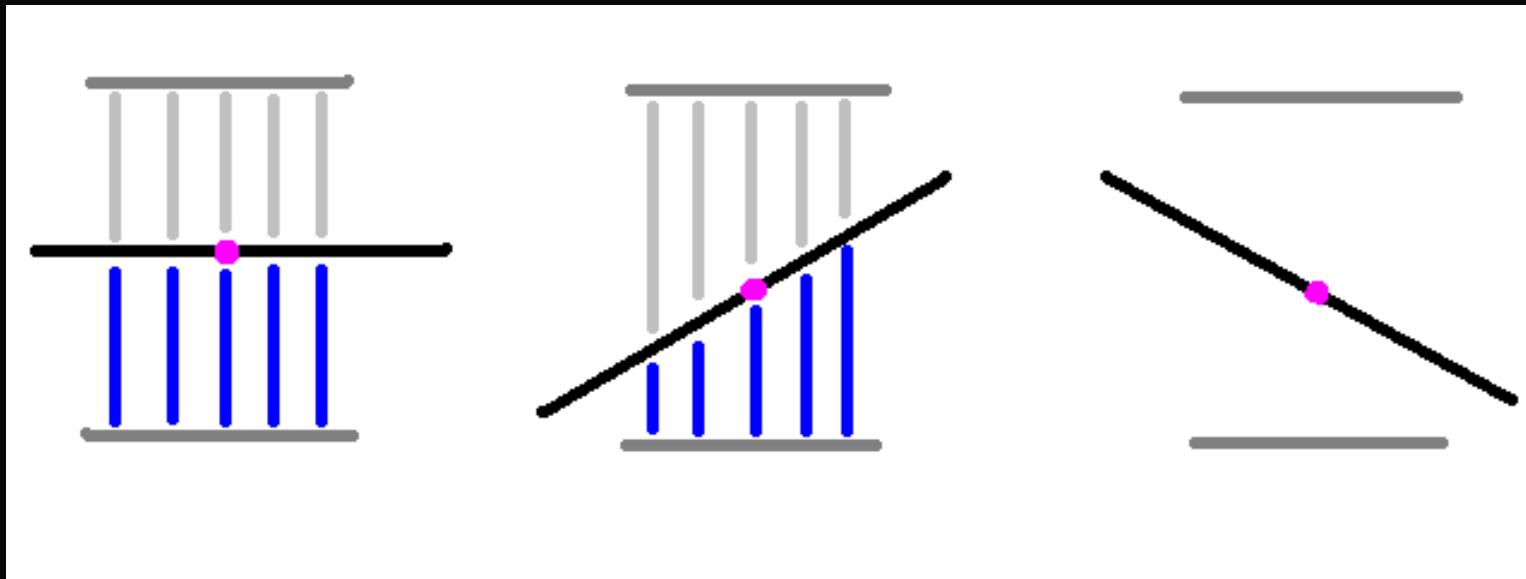
# SSAO Passes

- We can estimate volume without knowing the normal.
- Half Volume = Fully Unoccluded



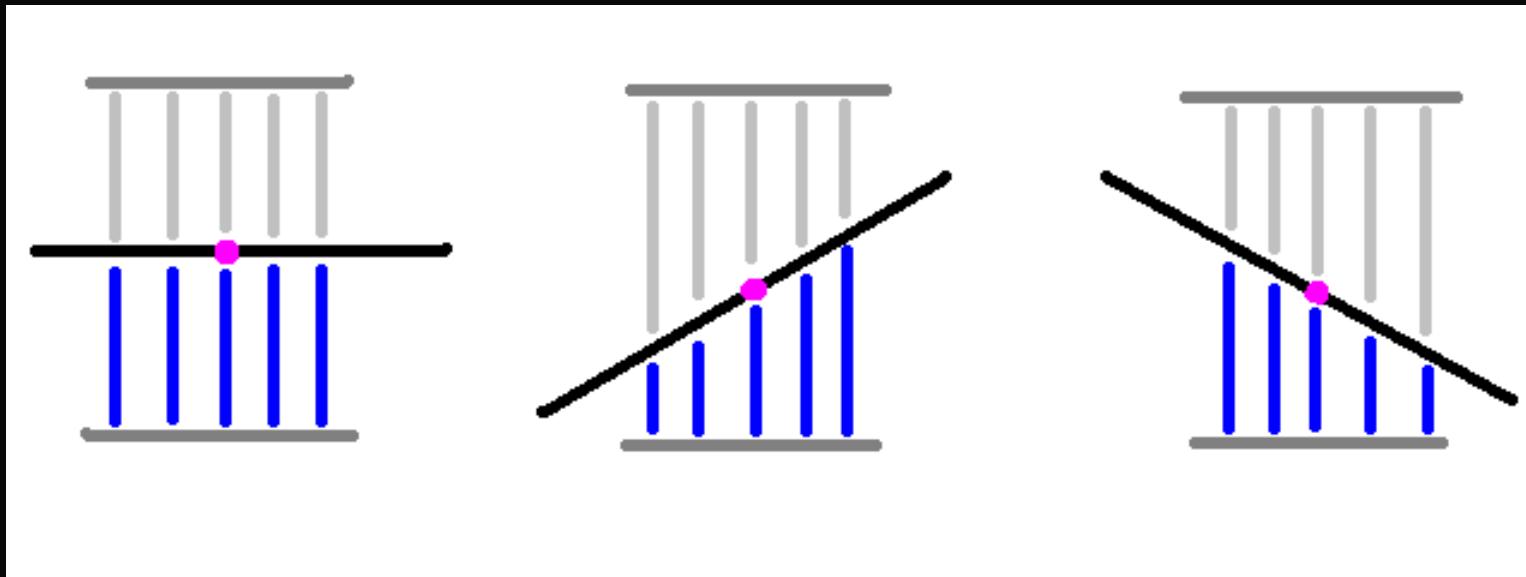
# SSAO Passes

- We can estimate volume without knowing the normal.
- Half Volume = Fully Unoccluded



# SSAO Passes

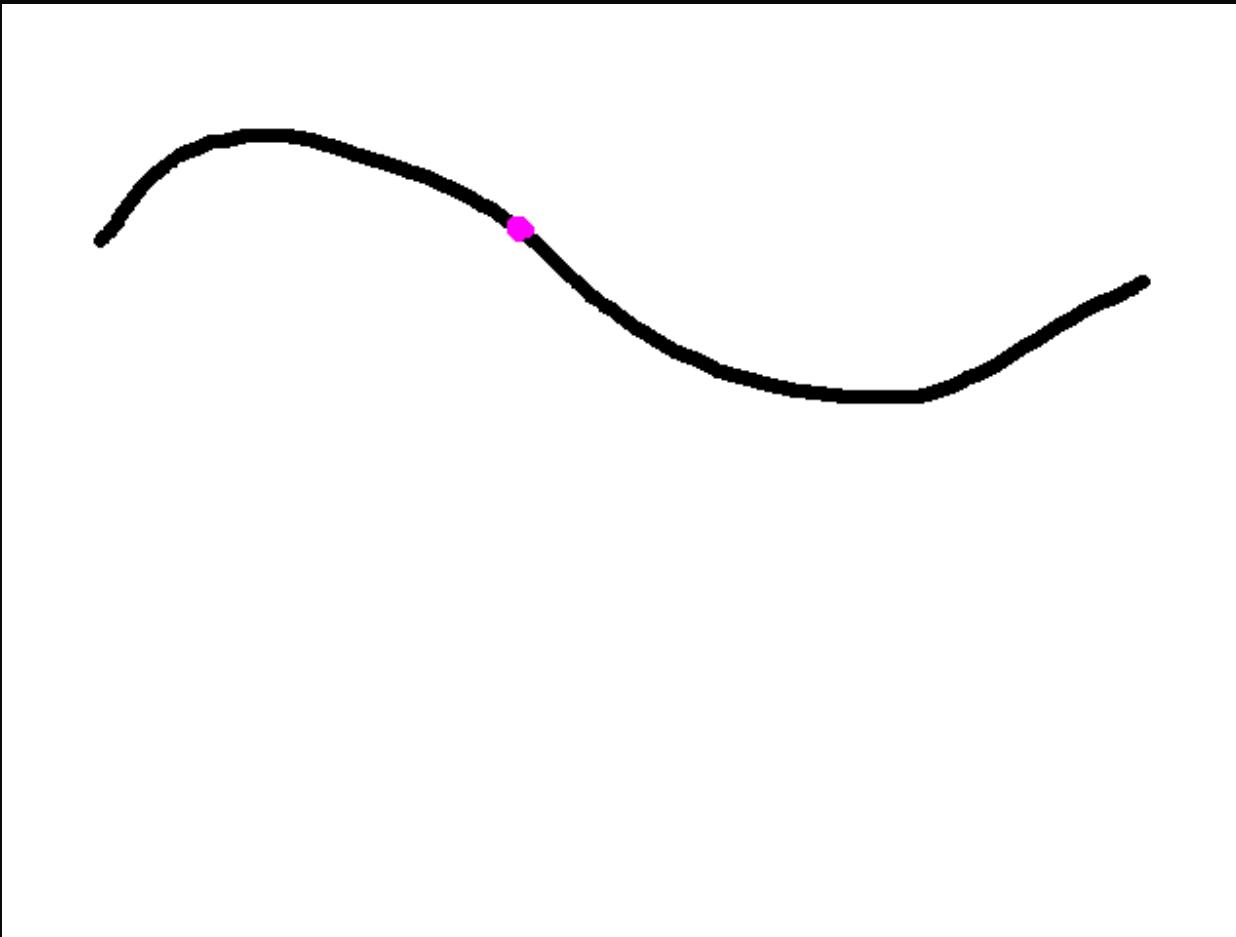
- We can estimate volume without knowing the normal.
- Half Volume = Fully Unoccluded



# SSAO Passes

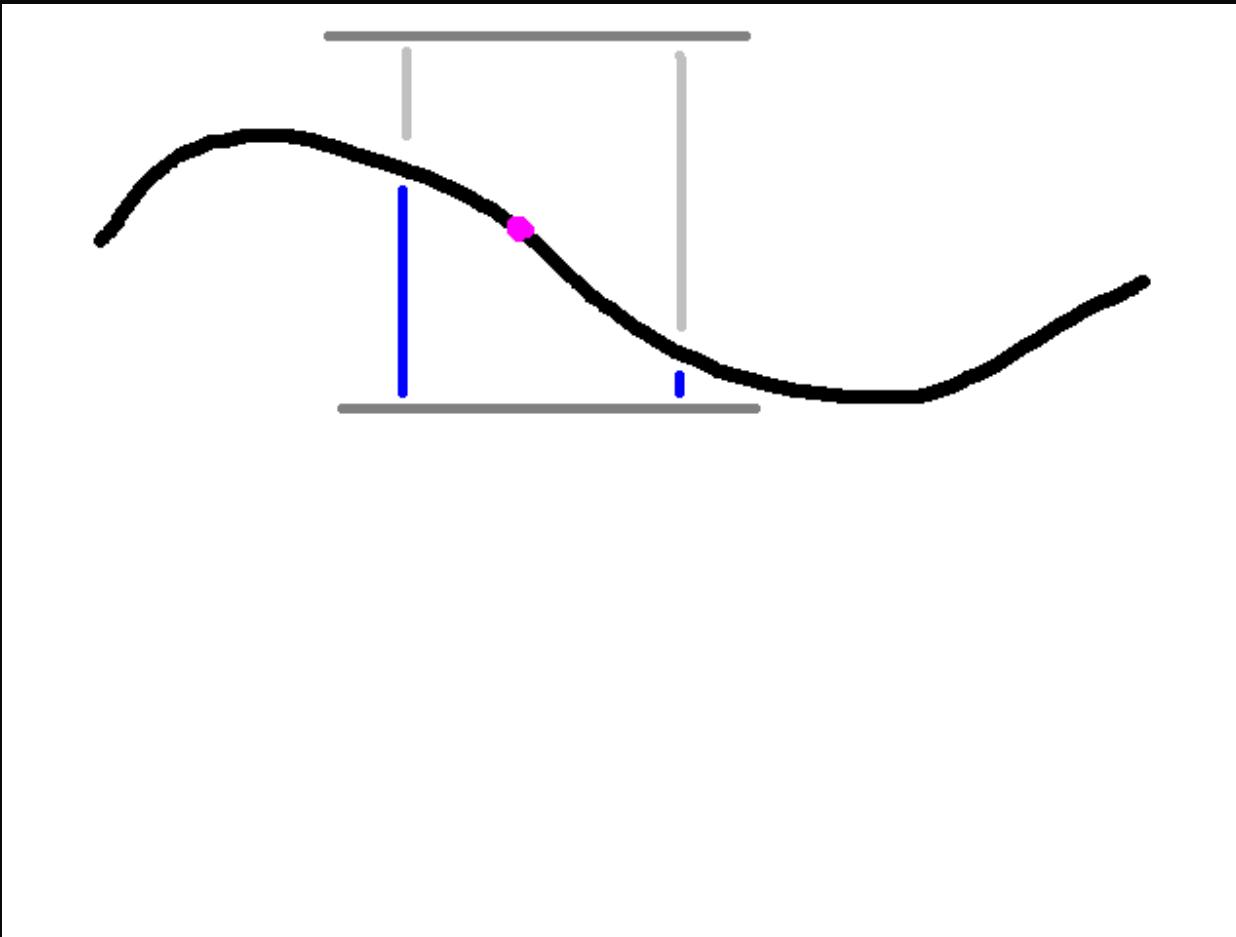
---

- Start with a point



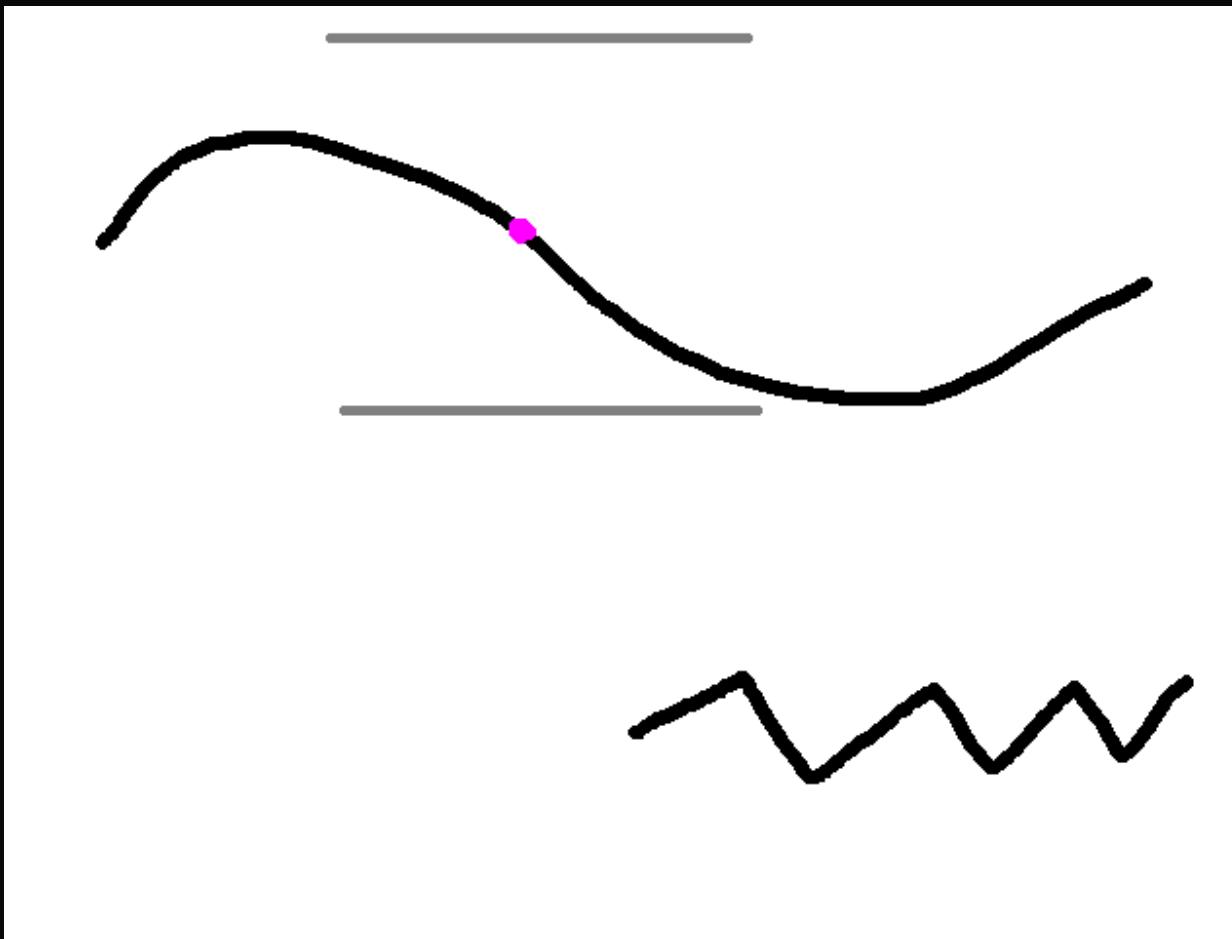
# SSAO Passes

- And sample pairs of points.



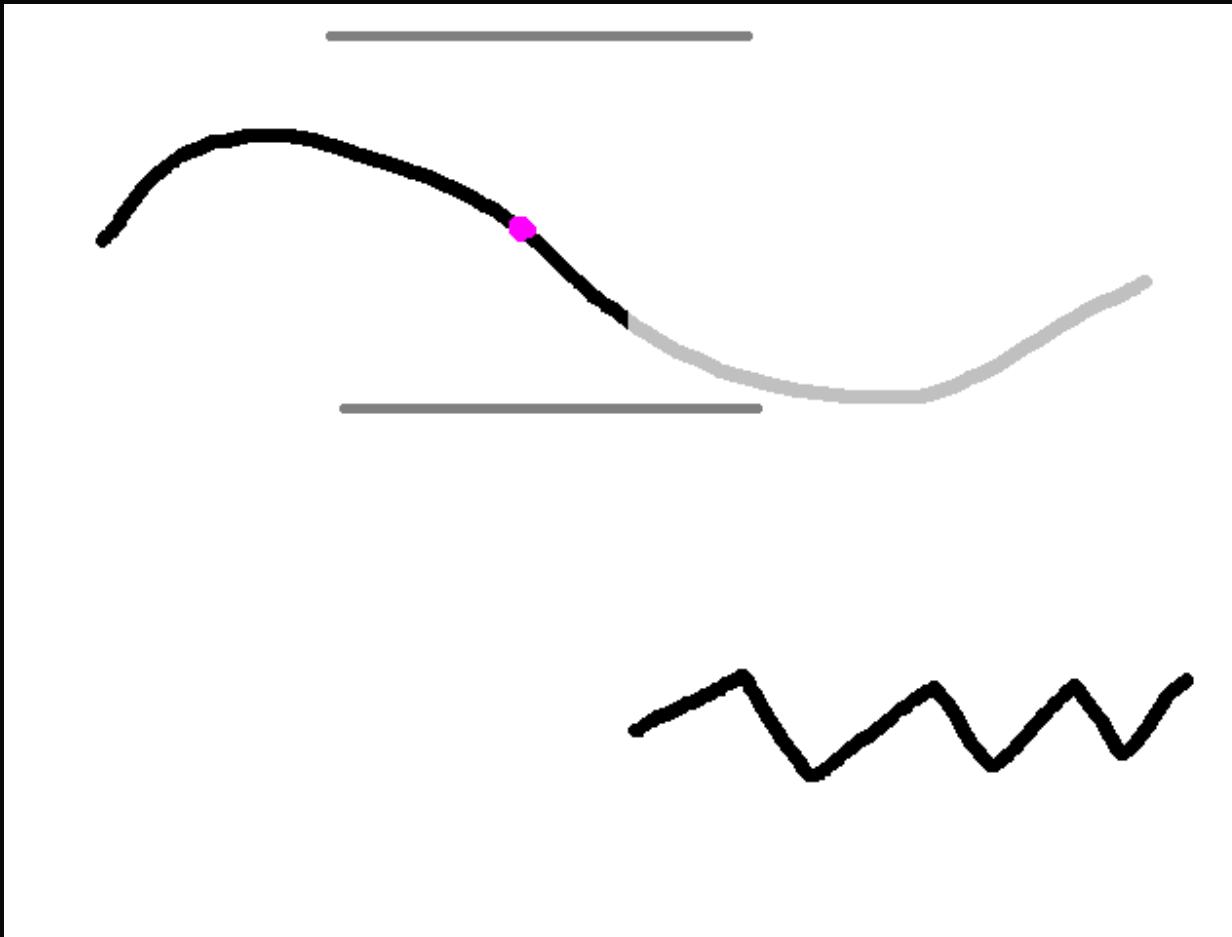
# SSAO Passes

- What if we have an occluder?



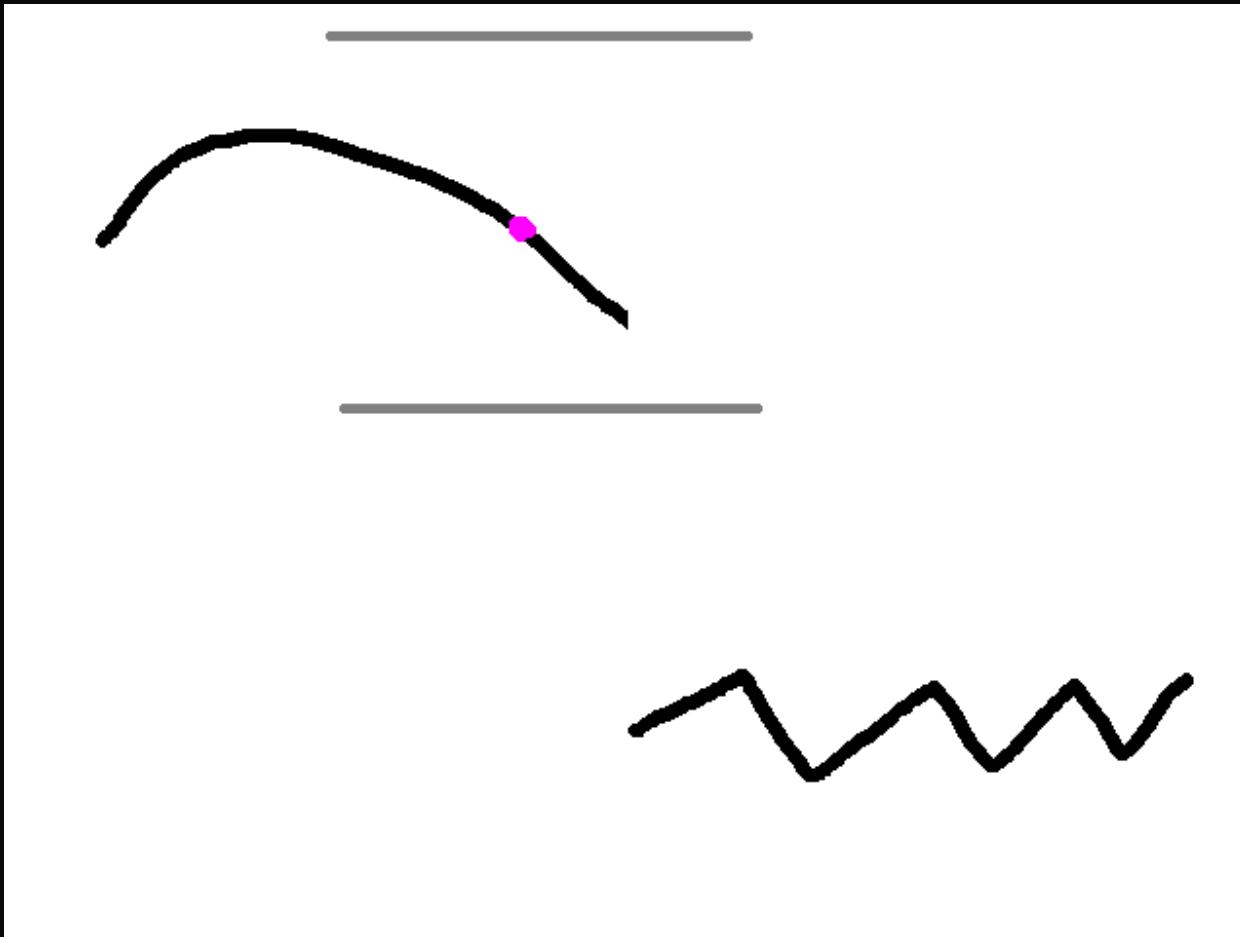
# SSAO Passes

- What if we have an occluder?



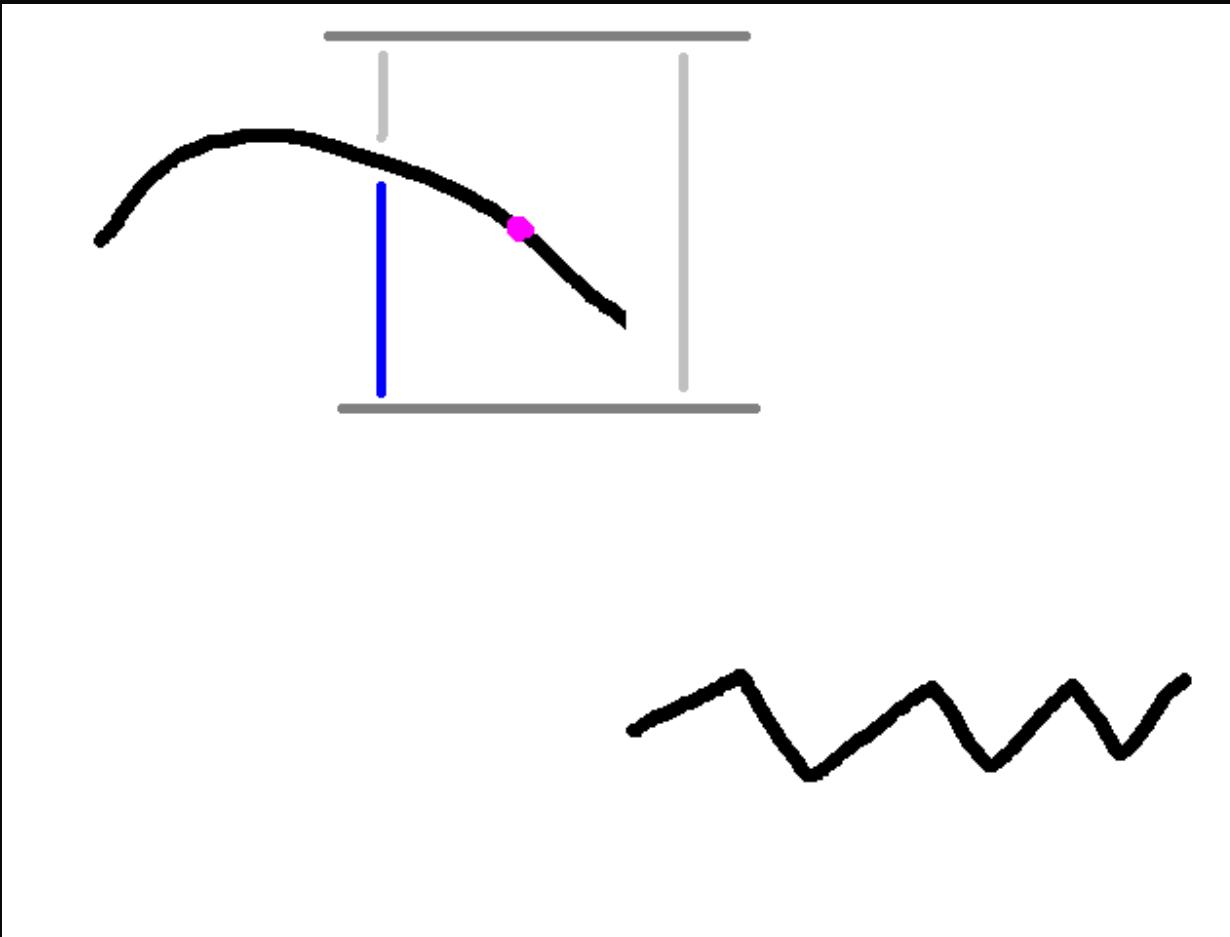
# SSAO Passes

- We lose depth info.



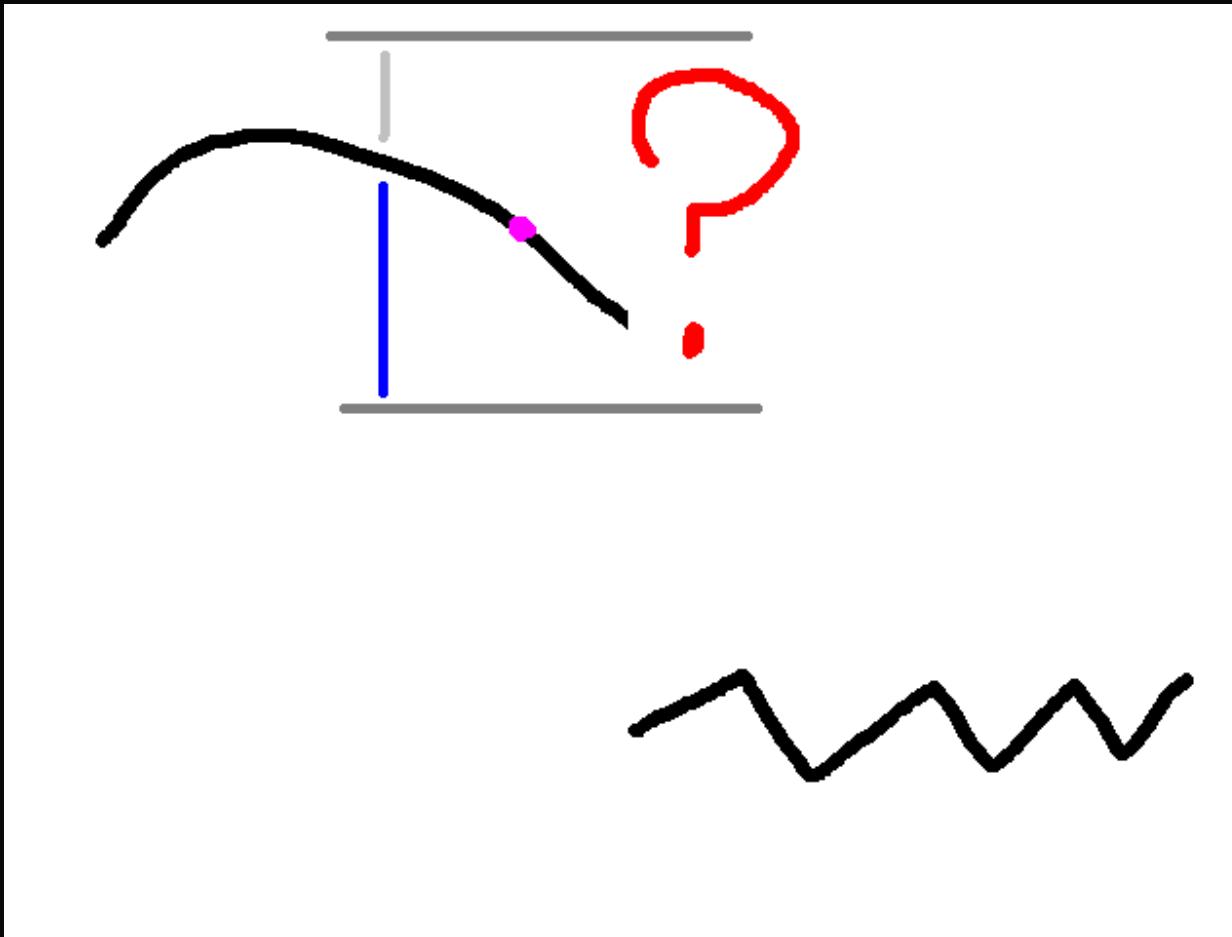
# SSAO Passes

- We could call it occluded.



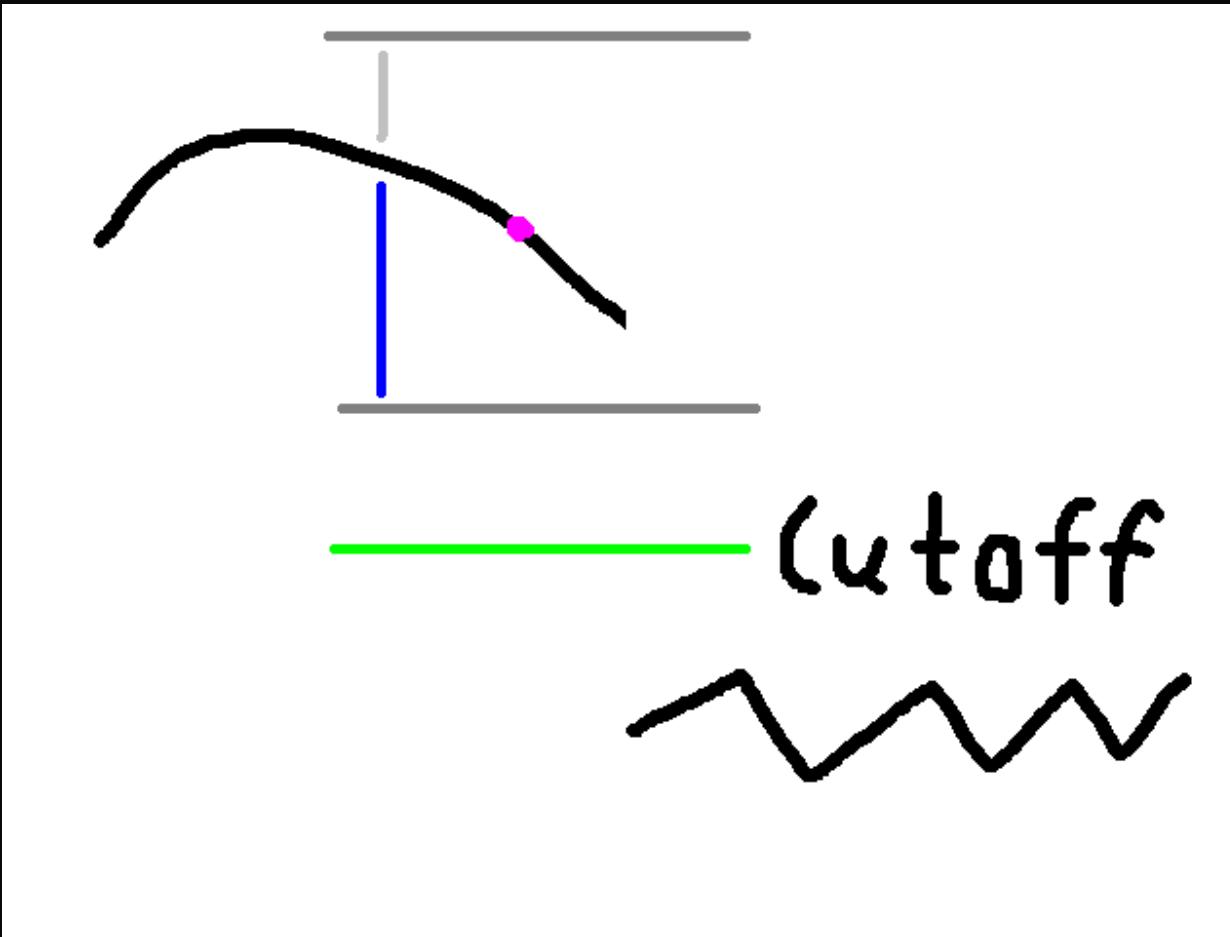
# SSAO Passes

- But, is that correct? Maybe not.



# SSAO Passes

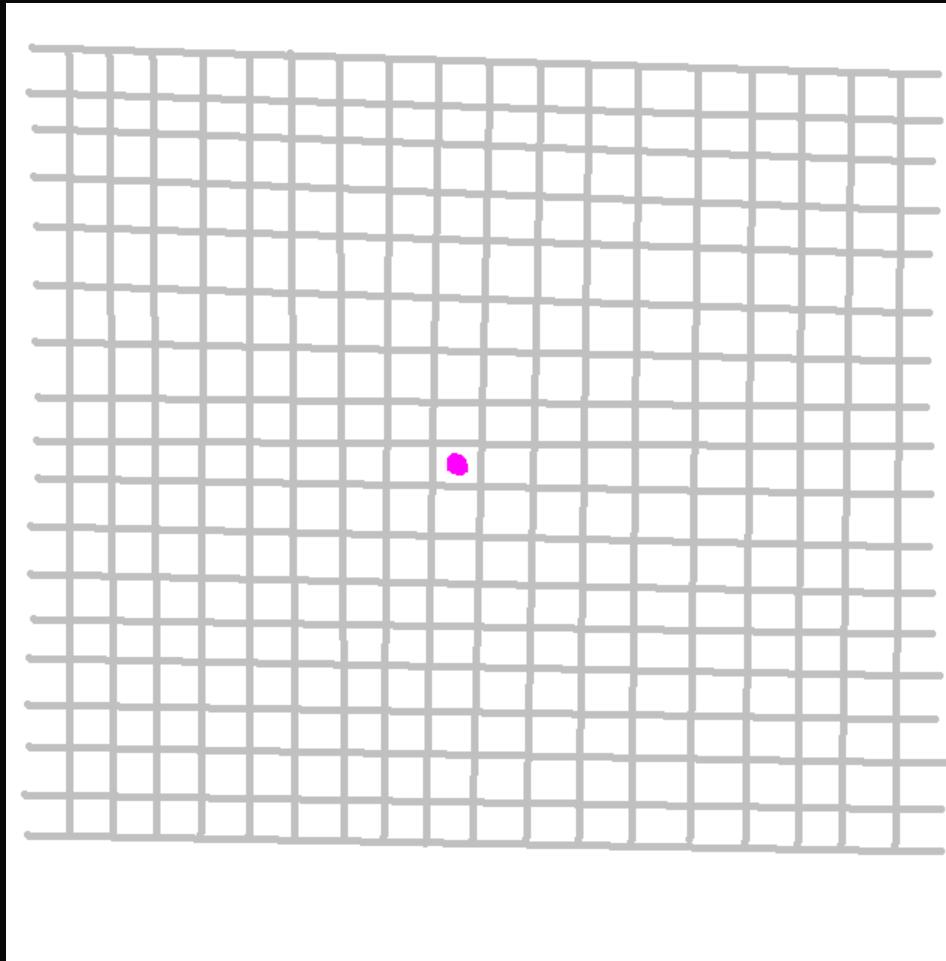
- Let's choose a cutoff, and mark the pair as invalid.



# SSAO Passes

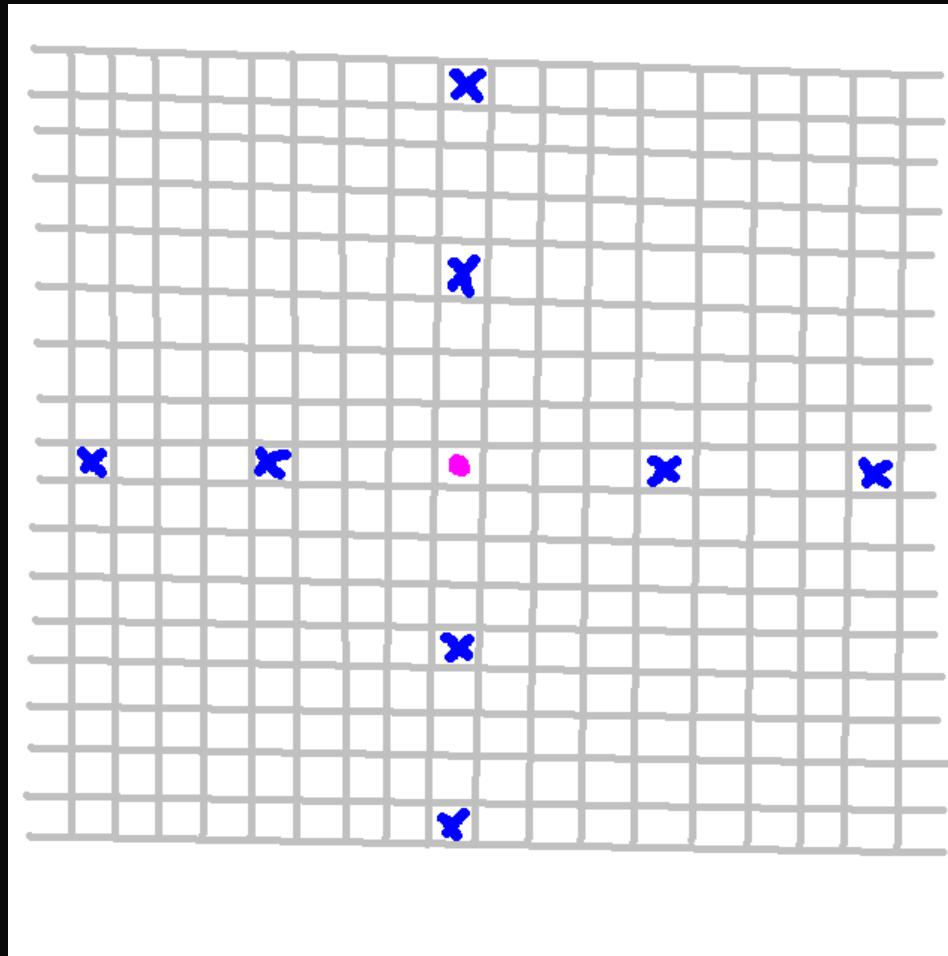
---

- Sample pattern?



# SSAO Passes

- Sample pattern. Discard in pairs.



# SSAO Passes

---

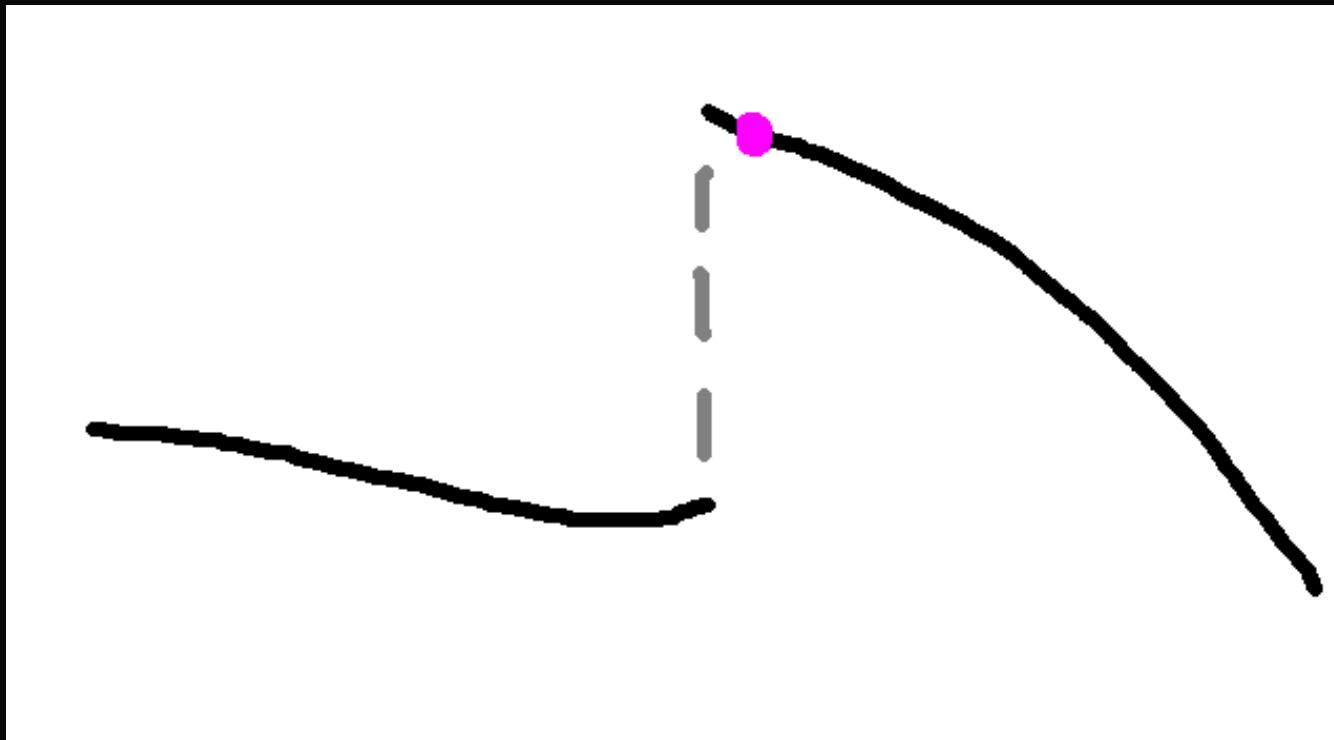
- Note the single light outline.



# SSAO Passes

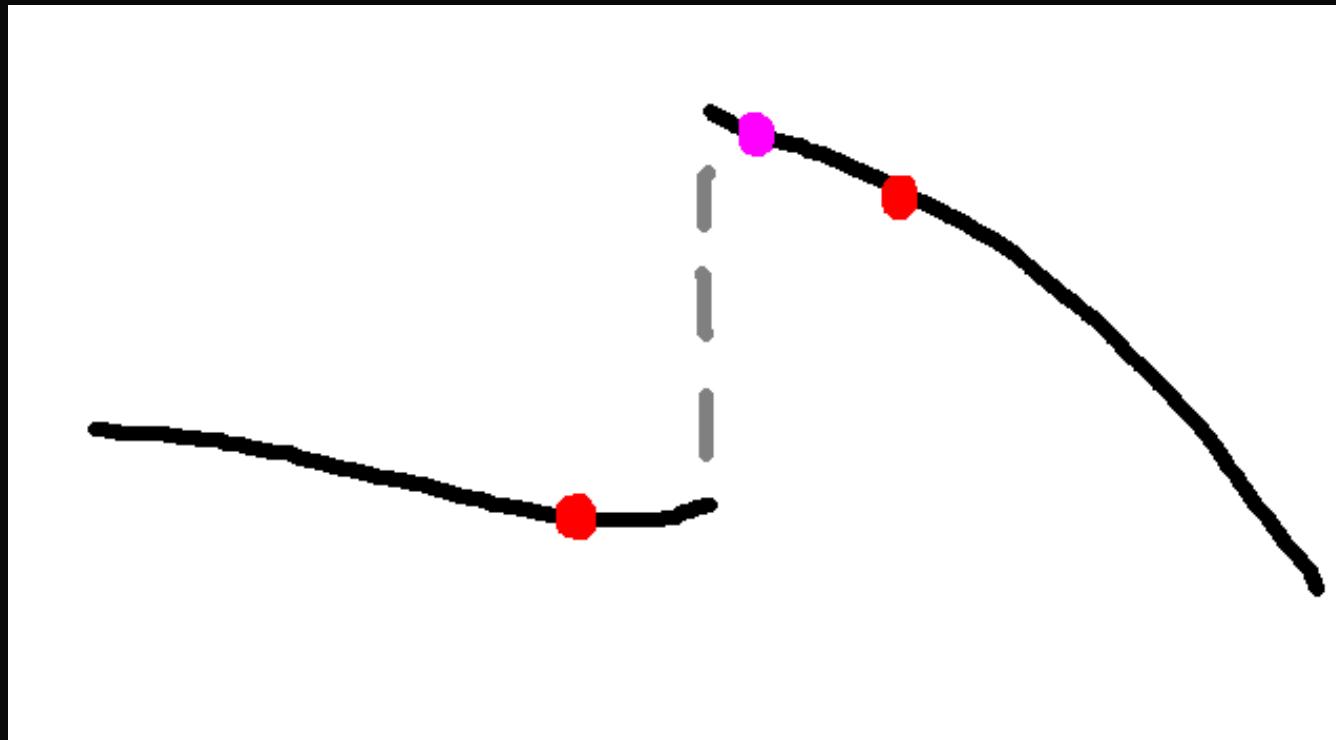
---

- Find discontinuities...



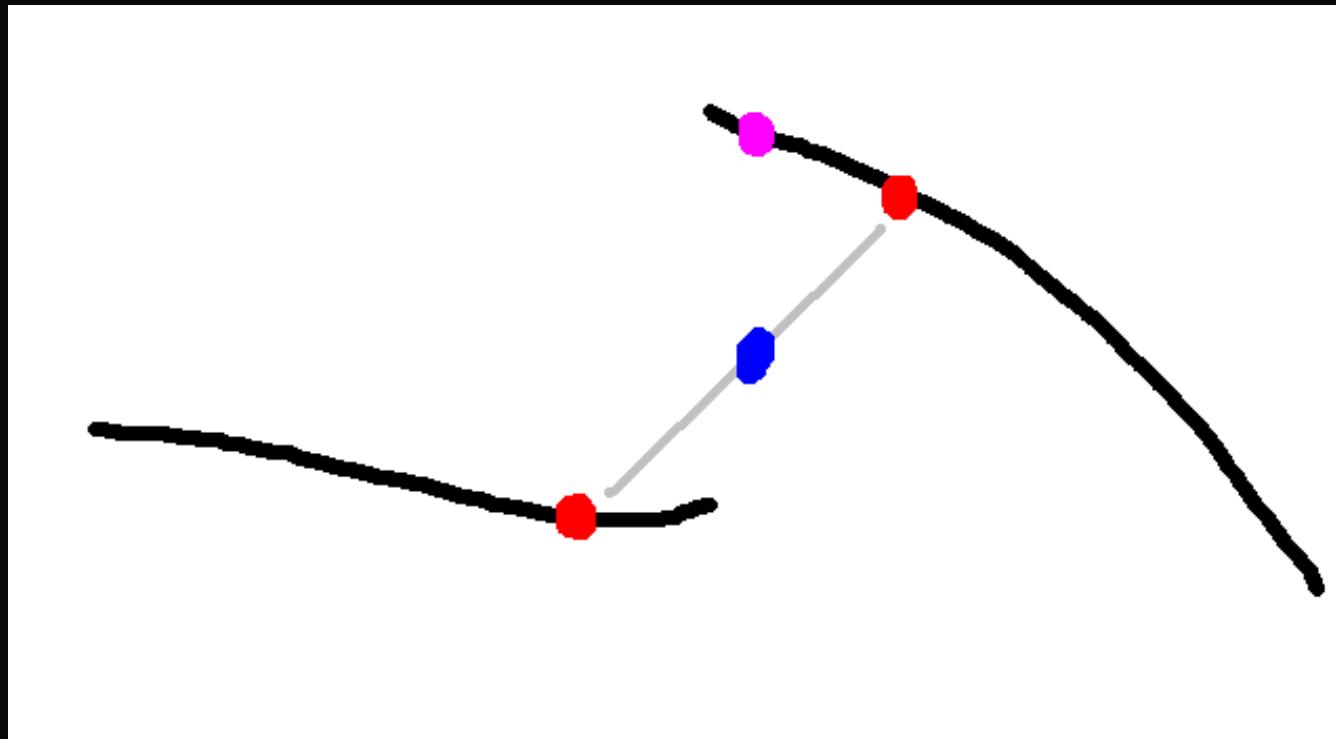
# SSAO Passes

- Find discontinuities...



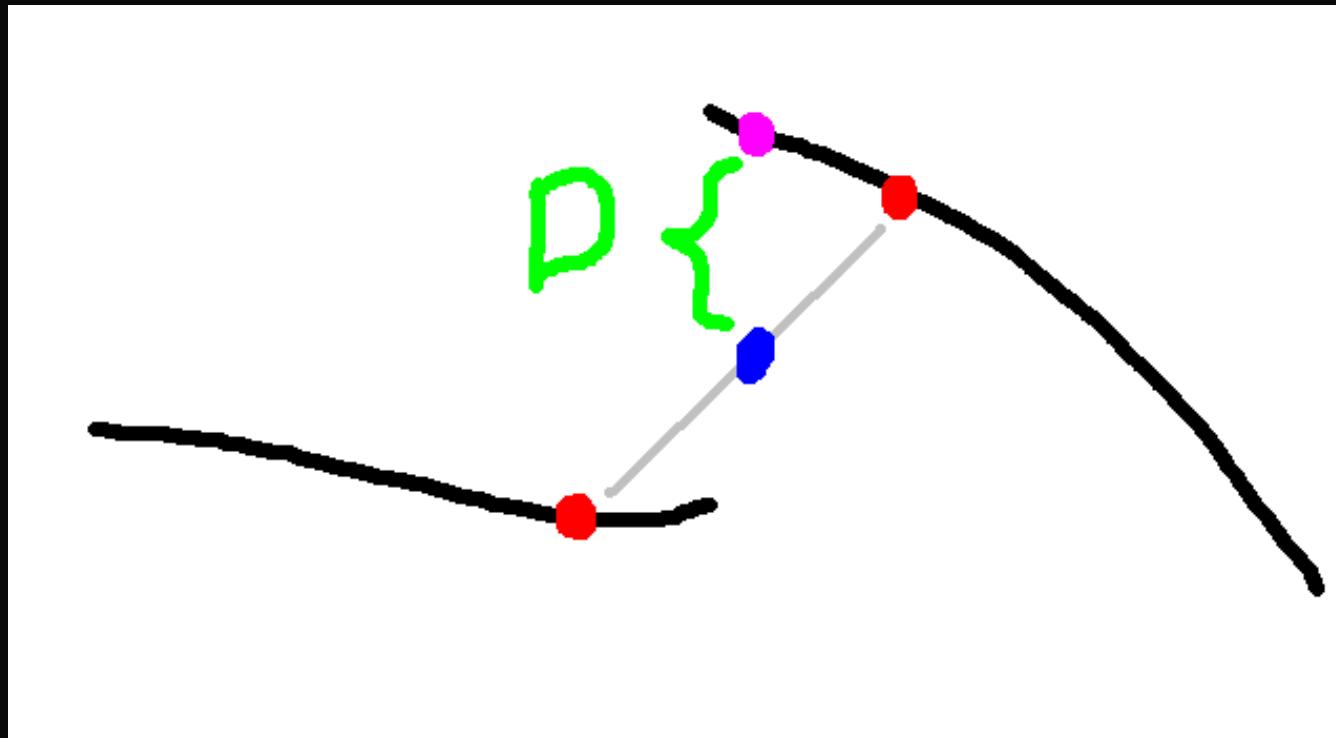
# SSAO Passes

- Find discontinuities...



# SSAO Passes

- Find discontinuities...



# SSAO Passes

---

- ...and dilate at discontinuities. Here is original.



# SSAO Passes

---

- After X dilate.



# SSAO Passes

---

- After Y dilate. And the light outline is gone.



# SSAO Passes

---

- And do a gaussian blur.



# SSAO Passes

---

- Final



# Artifacts

---

1. White Outline
2. Crossing Pattern
3. Depth Discontinuities

# Artifacts

---

- Blur adds white outline
- Crossing Pattern



# Artifacts

---

- More Crossing patterns



# Artifacts

---

- Depth Discontinuities. Notice a little anti-halo.



# Artifacts

---

- Hard to see in real levels though.



# Artifacts

---

- Artifacts are usually only noticeable to the trained eye
- Benefits outweigh drawbacks
- Have option to turn it off per surface
  - Most snow fades at distance
  - Some objects apply `sqrt()` to brighten it
  - Some objects just have it off

# More Shots

---

- On



# More Shots

---

- Off



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# More Shots

---

- Here's a few more shots.



# SSAO Conclusion

---

- Grounds characters and other objects
- Deepens our Blacks
- Artifacts are not too bad
  - Could use better filtering for halos
- Runs on SPUs, and doesn't need normal buffer



# Part 4: Architecture



# Architecture

---

- Time for part 4.
- Here is the base overview.
- Skip some things

# Rendering Passes

---

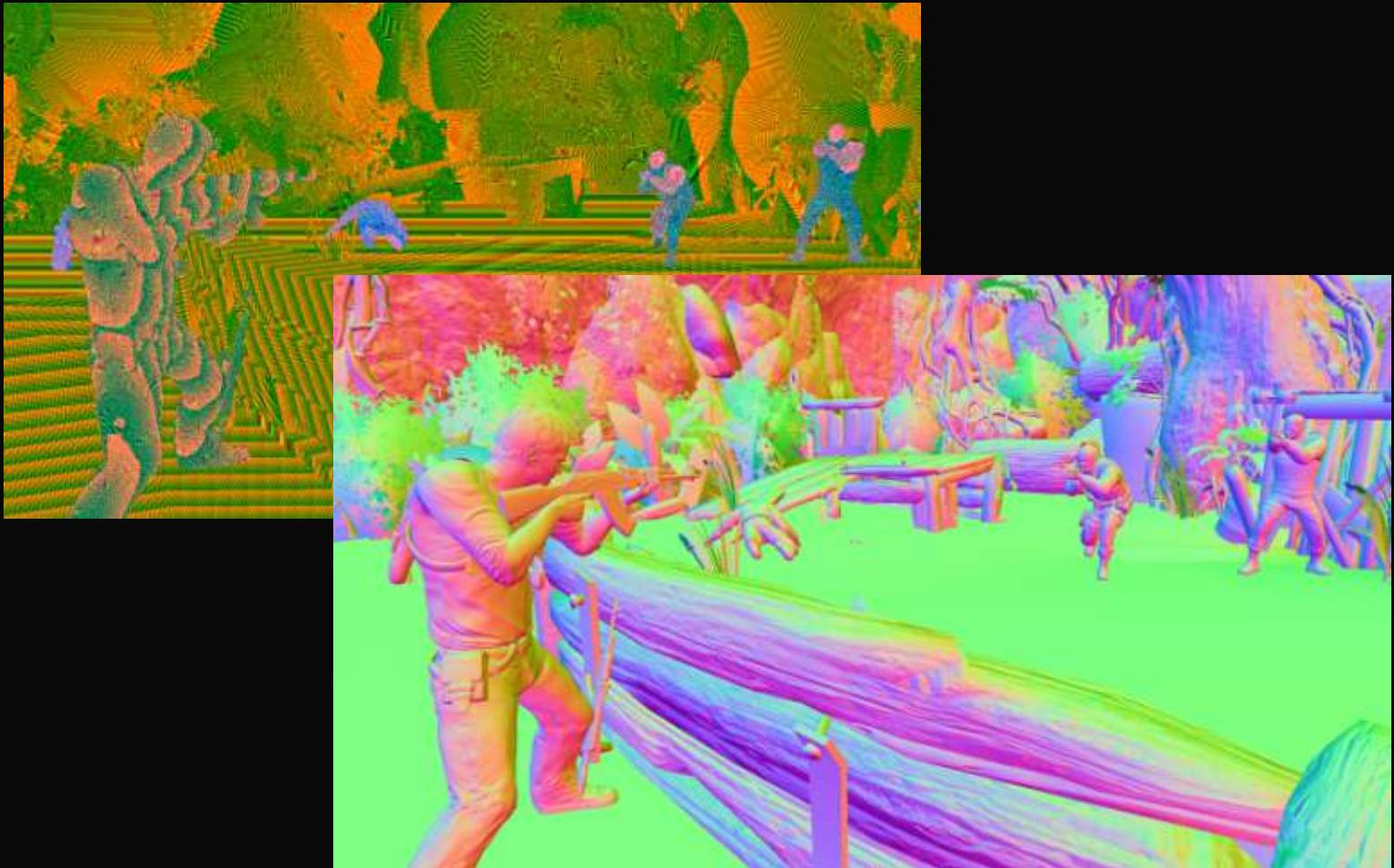
- Final output



# Rendering Passes

---

- Depth and Normal, 2x MSAA



# Rendering Passes

---

- Shadow depths for cascades



# Rendering Passes

---

- Cascade calculation to screen-space



# Rendering Passes

---

- Meanwhile on the SPUs...

# Rendering Passes

---

- SSAO



# Rendering Passes

---

- Fullscreen lighting. Diffuse and specular.
- Calculate lighting in Tiles



# Rendering Passes

---

- Back on the GPU...

# Rendering Passes

---

- Render to RGBM, 2x MSAA



# Rendering Passes

---

- Resolve RGBM 2x MSAA to FP16 1x



# Rendering Passes

---

- Transparent objects and post.



# Full Timeline

---

- Full Rendering Pipeline



# Full Timeline

- Full Render in 3 frames



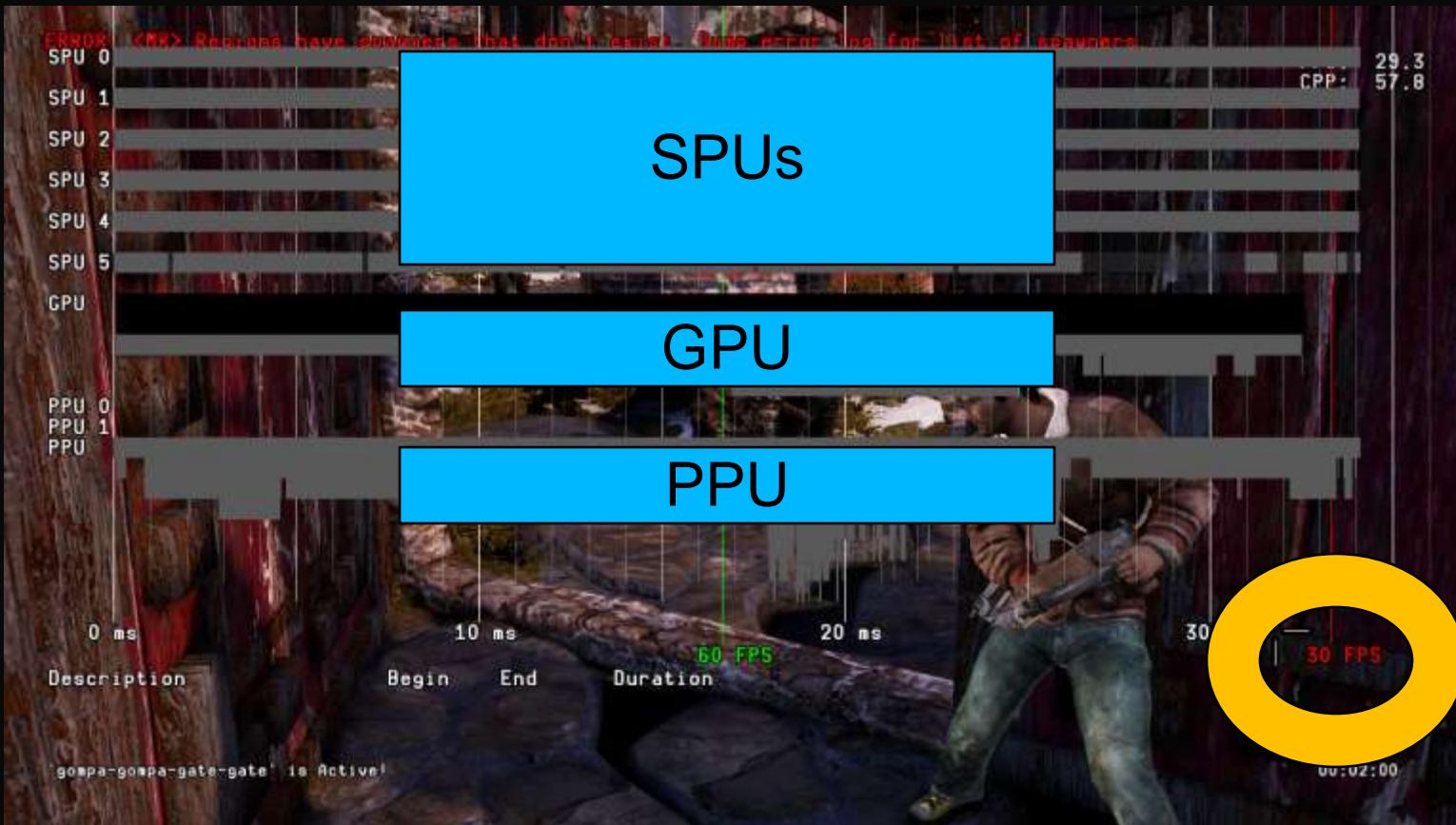
# Frame 1

- Begin Frame 1.



# Frame 1

- Timeline



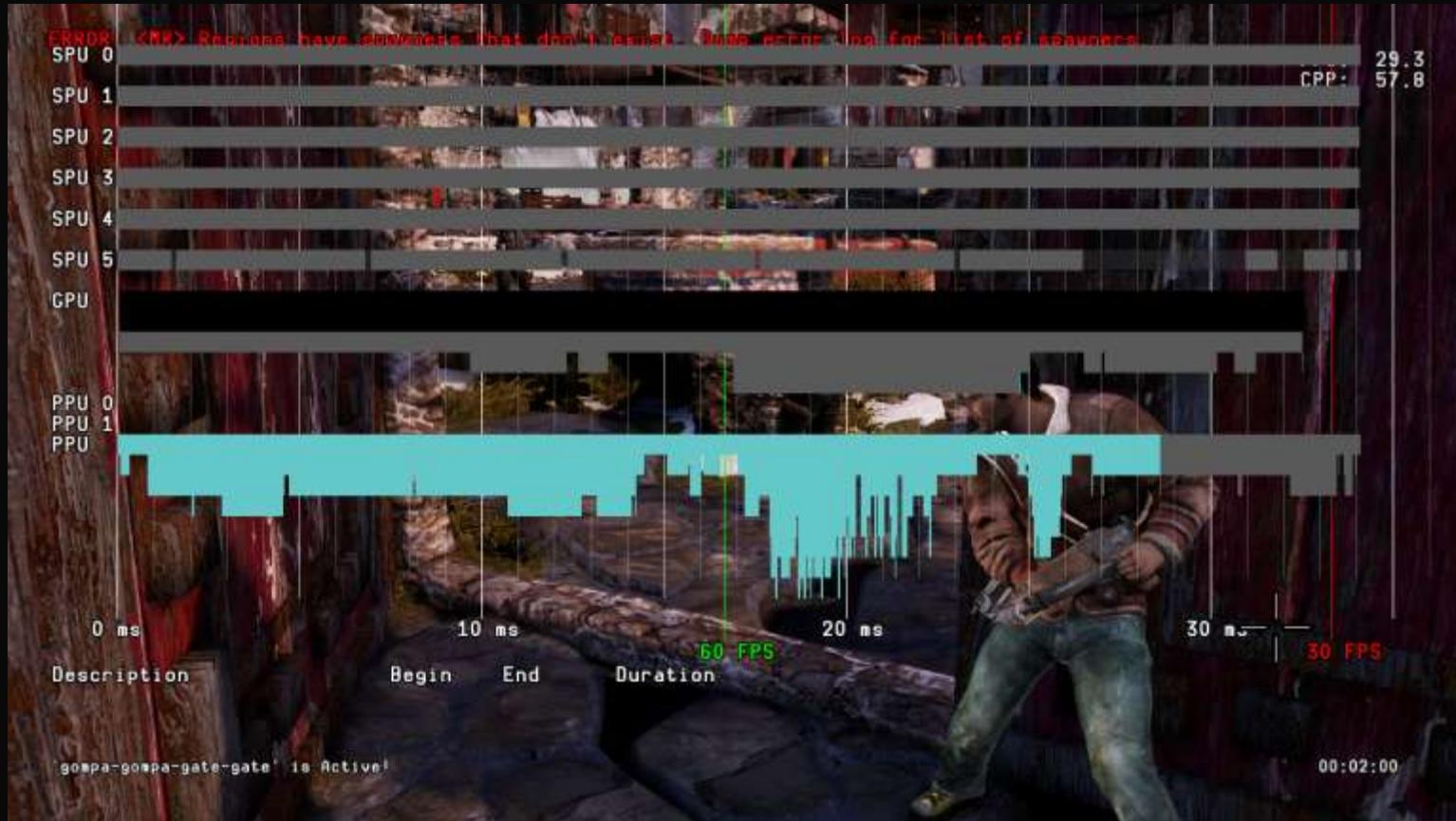
# Frame 1

- Begin Frame 1.



# Frame 1

- All PPU stuff.



# Frame 1

- PPU kicks off SPU jobs



# Frame 2

- Does most of the rendering.



# Frame 2

- Vertex processing on SPUs...



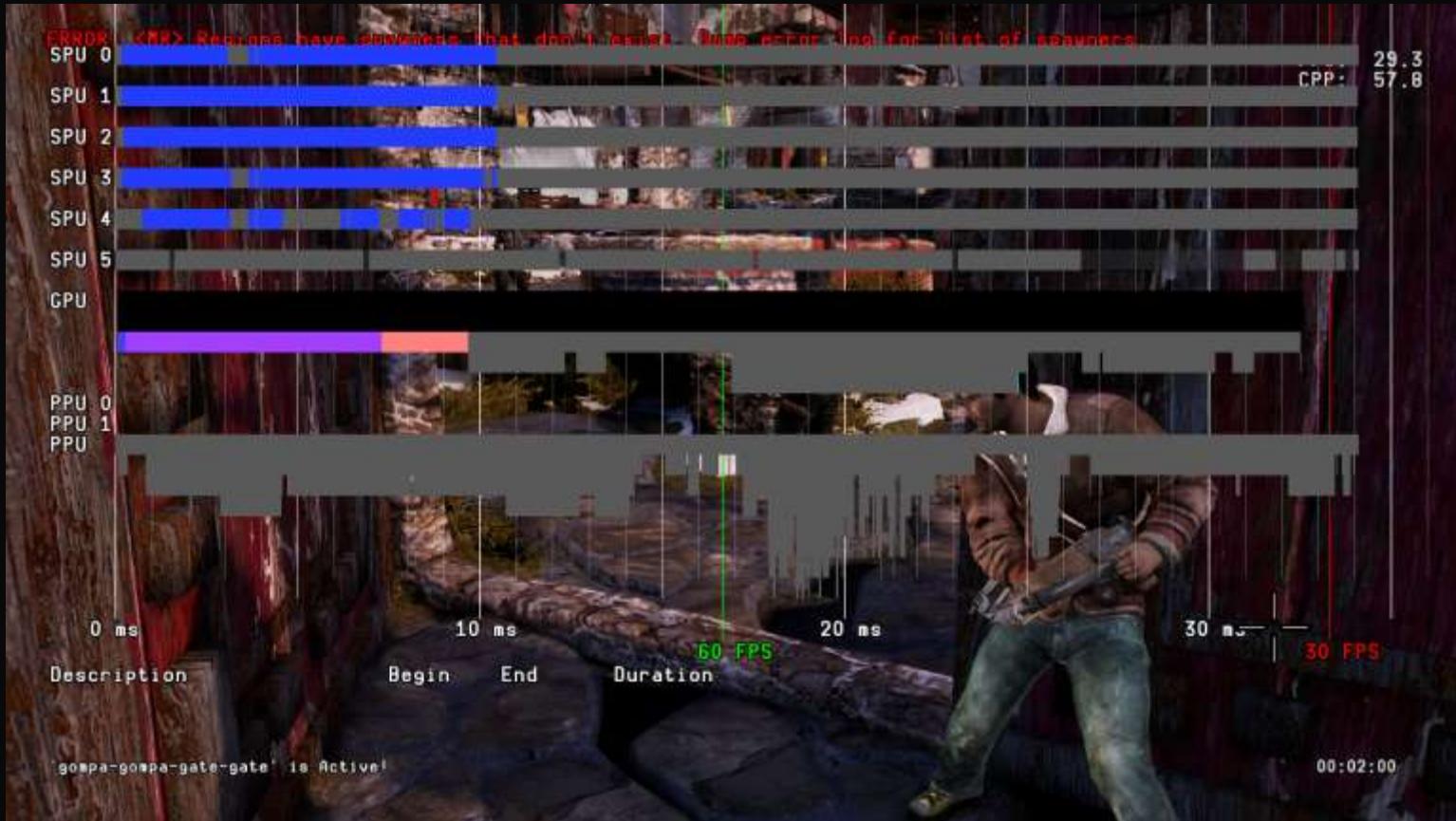
# Frame 2

- ...kicks off 2x DepthNormal pass.



# Frame 2

- Resolve 2x MSAA buffers to 1x and copy.



# Frame 2

- Spotlight shadow depth pass.



# Frame 2

- Sunlight shadow depth pass.



# Frame 2

- Kick Fullscreen Lighting SPU jobs.



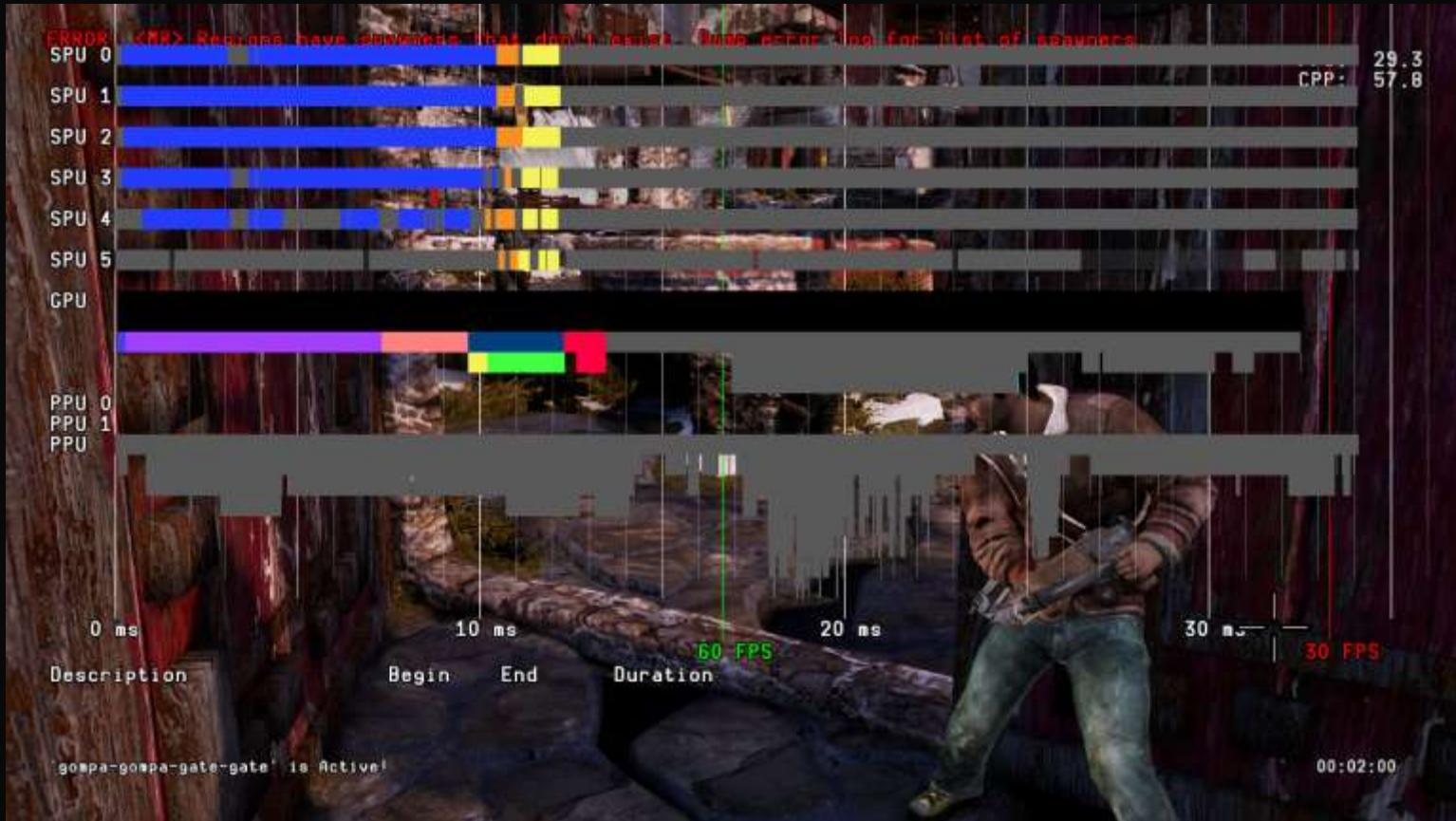
# Frame 2

- SSAO Jobs.



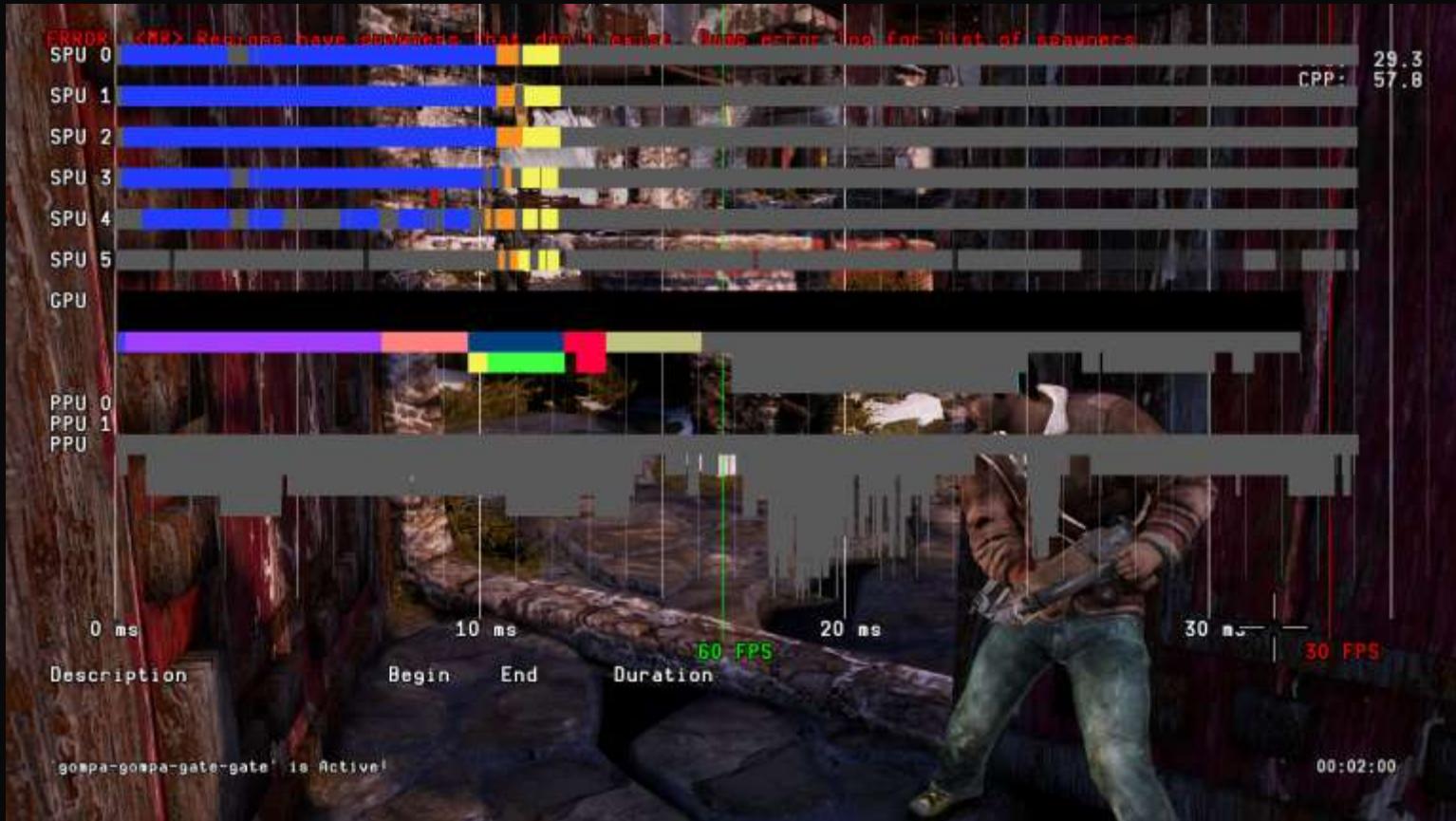
# Frame 2

- Motion Blur



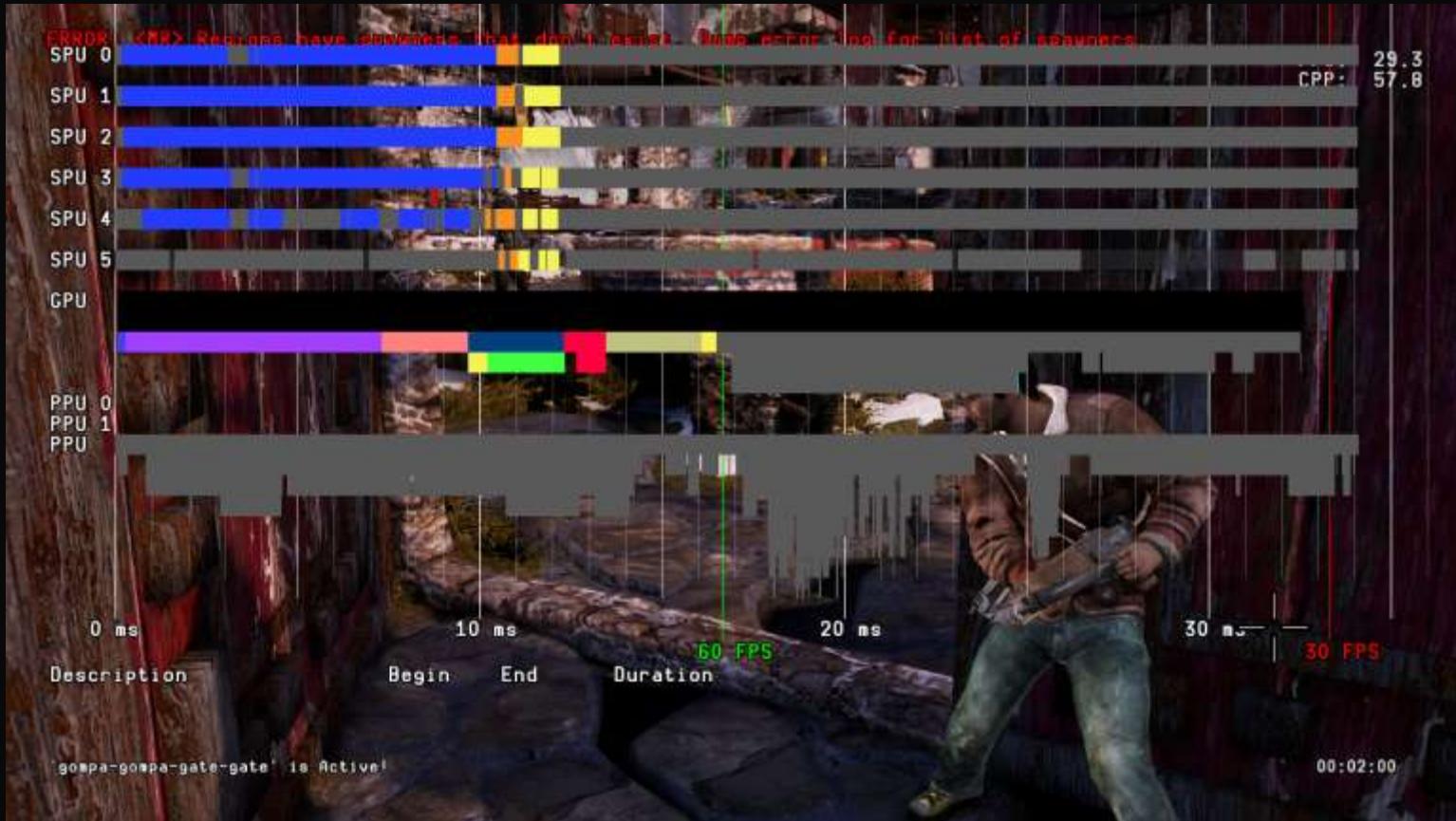
# Frame 2

- Resolve shadow to screen. Copy Fullscreen lighting.



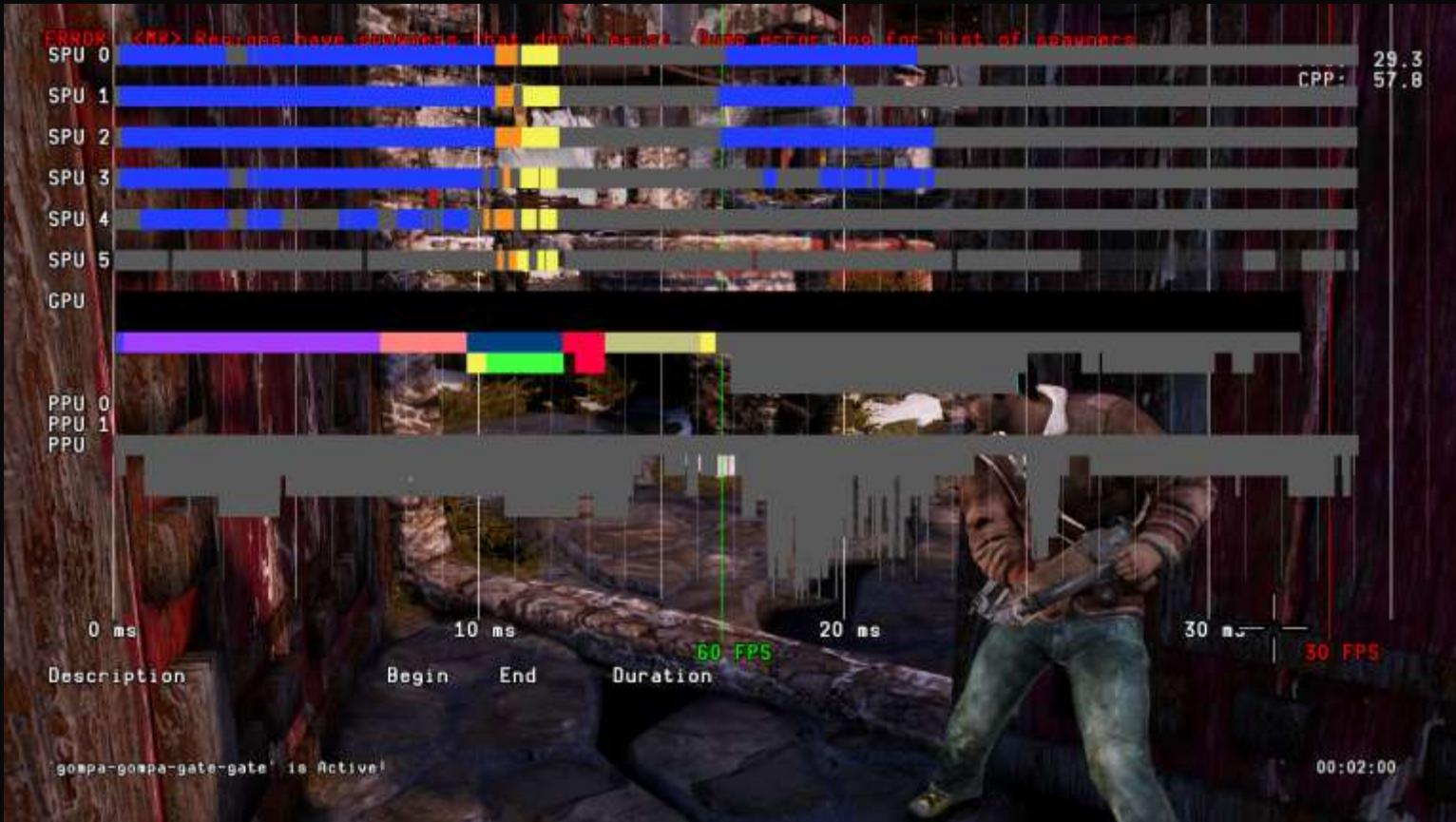
# Frame 2

- Add Fullscreen shadowed lights.



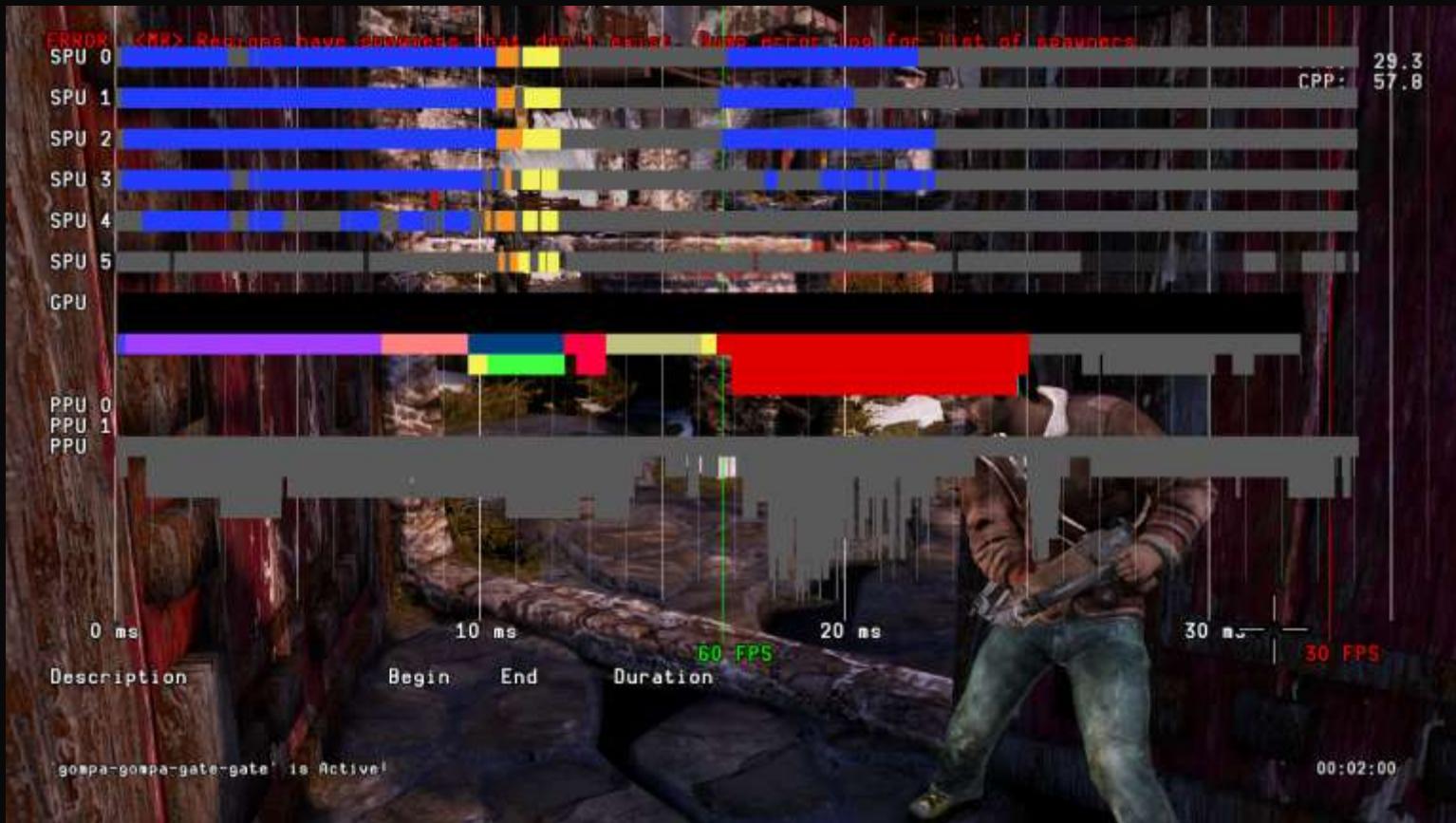
# Frame 2

- Standard Pass Vertex Processing on SPUs.



# Frame 2

- Standard pass. Writes to 2x RGBM.



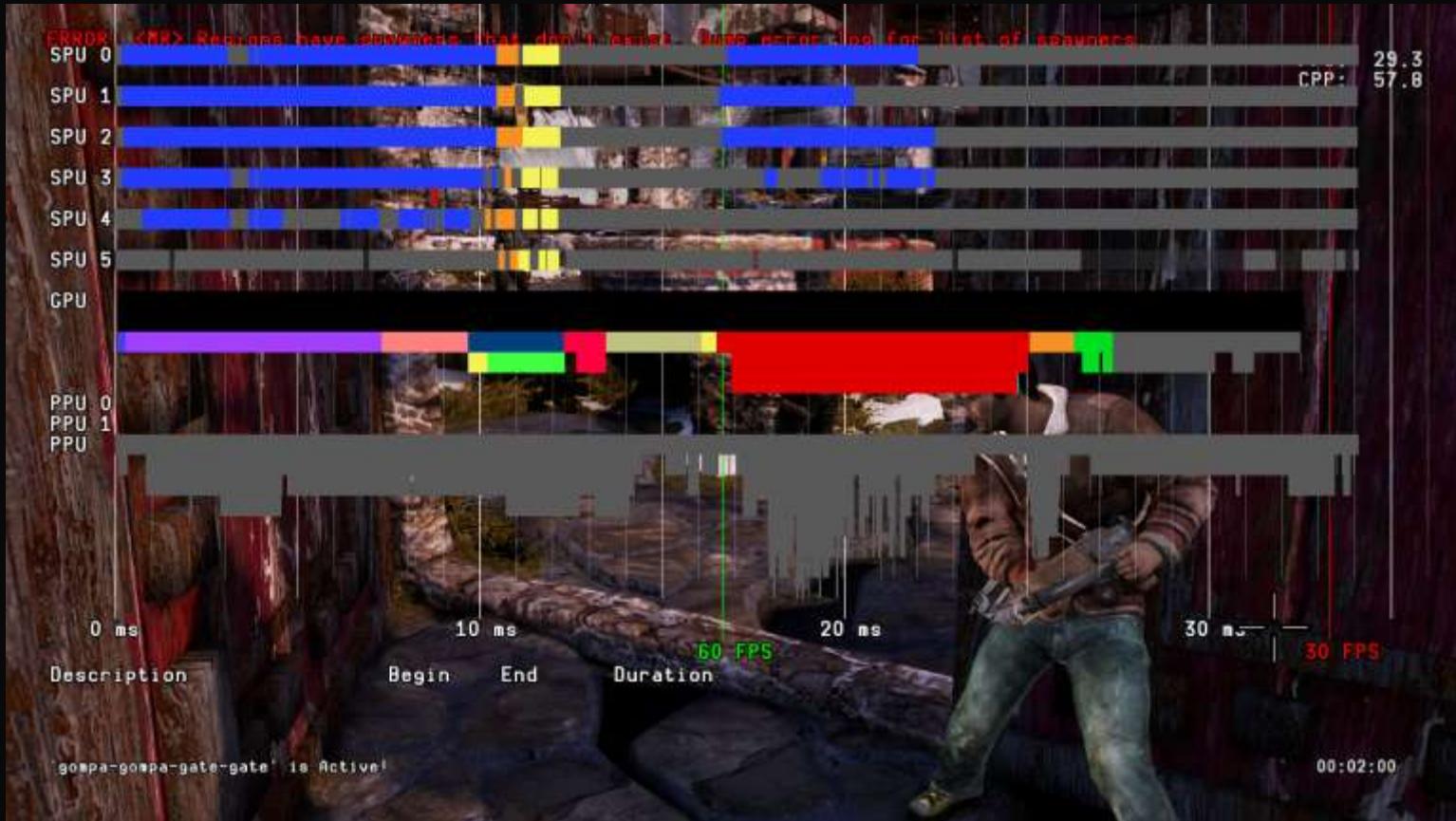
# Frame 2

- Resolve 2x RGBM to 1x FP16



# Frame 2

- Full-Res Alpha-Blended effects.



# Frame 2

- Half-Res Particles and Composite.



# Frame 3

- Now for the 3<sup>rd</sup> frame.



# Frame 3

- Start PostFX jobs at end of Frame 2.



# Frame 3

- Fill in PostFX jobs in Frame 3 with low priority.



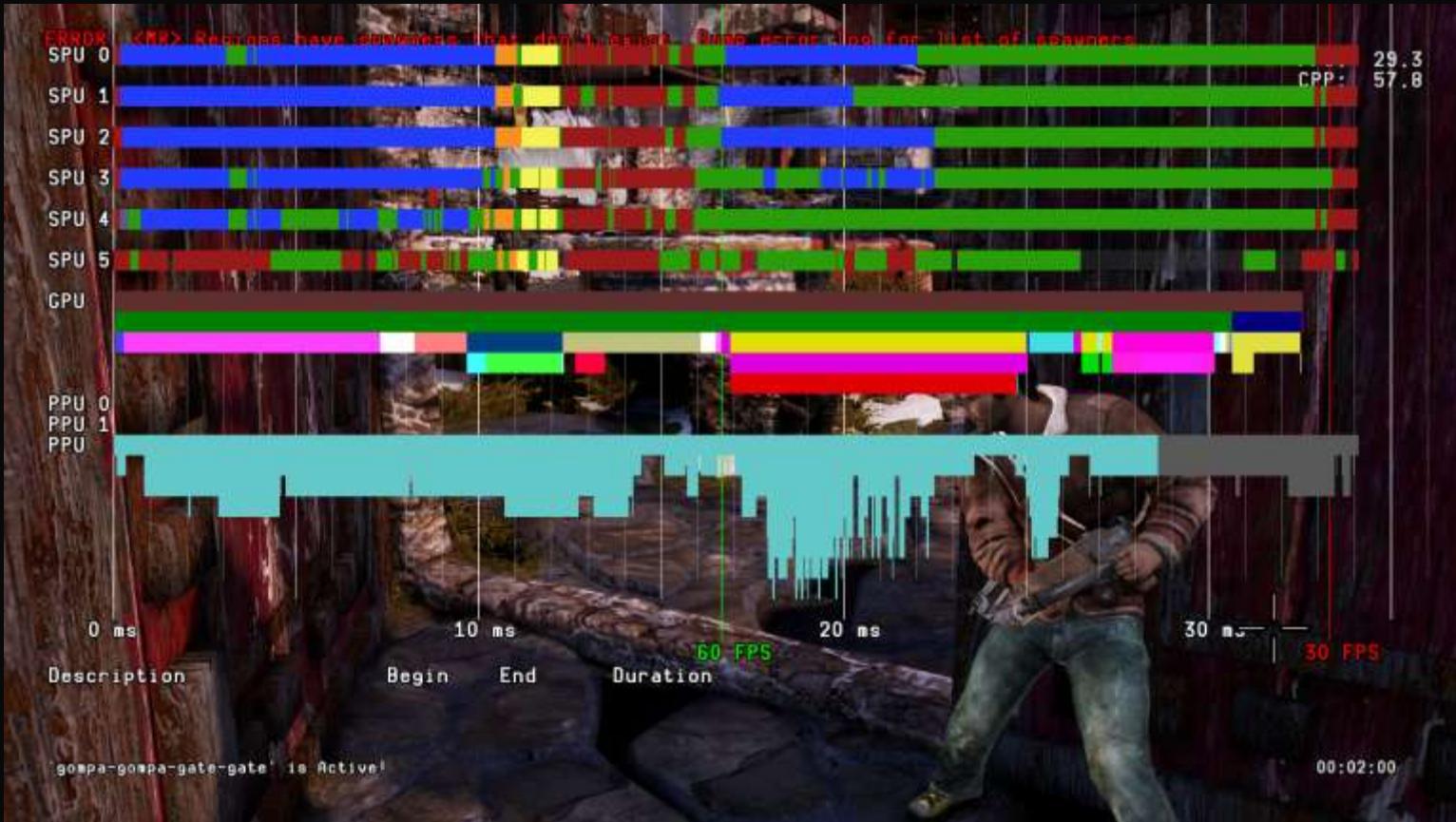
# Frame 3

- Read from main memory, do distortion and HUD.



# Full Timeline

- All together.



# SPU Optimization

- Quick notes on how we optimize SPU code.



# Performance

---

- Yes! You can optimize SPU code by hand!
  - This was news to me when I started here.

# Performance

---

- Let's talk about Laundry
- Say you have 5 loads to do
- 1 washer, 1 dryer

# In-Order

---

- Load 1 in Washer
- Load 1 in Dryer
- Load 2 in Washer
- Load 2 in Dryer
- Load 3 in Washer
- Load 3 in Dryer
- Load 4 in Washer
- Load 4 in Dryer
- Load 5 in Washer
- Load 5 in Dryer

# Pipelined

---

- Load 1 in Washer
- Load 2 in Washer, Load 1 in Dryer
- Load 3 in Washer, Load 2 in Dryer
- Load 4 in Washer, Load 3 in Dryer
- Load 5 in Washer, Load 4 in Dryer
- Load 5 in Dryer

# Software Pipelining

---

- We can do this in software
- Called “Software Pipelining”
- It’s how we optimize SPU code by hand
- Pal Engstad (our Lead Graphics Programmer) will be putting a doc online explaining the full process.
- Should be online: look for the post on the blog
- Direct links:
  - [www.naughtydog.com/docs/gdc2010/intro-spu-optimizations-part-1.pdf](http://www.naughtydog.com/docs/gdc2010/intro-spu-optimizations-part-1.pdf)
  - [www.naughtydog.com/docs/gdc2010/intro-spu-optimizations-part-2.pdf](http://www.naughtydog.com/docs/gdc2010/intro-spu-optimizations-part-2.pdf)

# SPU C++ Intrinsics

```
for( int y = 0; y < clampHeight; y++ )      {
    int IColorOffset = y * kAoBufferLineWidth;
    VF32 currWordAo = pvColor[ IColorOffset / 4 ];
    prevWordAo[0] = prevWordAo[1] = prevWordAo[2] = prevWordAo[3] = currWordAo[0];
    ao_n4 = prevWordAo;
    ao_n3 = (VF32)si_shufb( (qword)prevWordAo, (qword)currWordAo, (qword)s_BCDa );
    ao_n2 = (VF32)si_shufb( (qword)prevWordAo, (qword)currWordAo, (qword)s_CDab );
    ao_n1 = (VF32)si_shufb( (qword)prevWordAo, (qword)currWordAo, (qword)s_Dabc );
    ao_curr = currWordAo;

    for (int x = 0; x < kAoBufferLineWidth; x+=4) {
        VF32 nextWordAo = pvColor[ (IColorOffset + x + 4)/ 4 ];
        VF32 ao_p1 = (VF32)si_shufb( (qword)currWordAo, (qword)nextWordAo, (qword)s_BCDa );
        VF32 ao_p2 = (VF32)si_shufb( (qword)currWordAo, (qword)nextWordAo, (qword)s_CDab );
        VF32 ao_p3 = (VF32)si_shufb( (qword)currWordAo, (qword)nextWordAo, (qword)s_Dabc );
        VF32 ao_p4 = nextWordAo;
        VF32 blurAo = ao_n4*w4 + ao_n3*w3 + ao_n2*w2 + ao_n1*w1 + ao_curr*w0;
        blurAo += ao_p1*w1 + ao_p2*w2 + ao_p3*w3 + ao_p4*w4;
        pvColor[ (IColorOffset + x)/ 4 ] = blurAo;
        prevWordAo = currWordAo;
        currWordAo = nextWordAo;
        ao_n4 = prevWordAo;
        ao_n3 = ao_p1;
        ao_n2 = ao_p2;
        ao_n1 = ao_p3;
        ao_curr = currWordAo;
    }
}
```

# SPU Assembly 1/2

EVEN

ODD

loop:

{nop}  
{nop}  
{nop}  
{nop}  
{nop}

{e2} selb ao\_n4, ao\_n4, prevAo0, endLineMask  
{e2} selb ao\_n3, ao\_n3, ao\_n30, endLineMask  
{e2} selb ao\_n2, ao\_n2, ao\_n20, endLineMask  
{e2} selb ao\_n1, ao\_n1, ao\_n10, endLineMask

{nop}  
{nop}  
{nop}  
{nop}

{e6} fa ao\_n4, ao\_n4, nextAo  
{e6} fa ao\_n3, ao\_n3, ao\_p3  
{e6} fa ao\_n2, ao\_n2, ao\_p2  
{e6} fa ao\_n1, ao\_n1, ao\_p1

{o6} lqx currAo, pvColor, colorOffset  
{o4} shufb prevAo0, currAo, currAo, s\_AAAA  
{o4} shufb ao\_n30, prevAo0, currAo, s\_BCDa  
{o4} shufb ao\_n20, prevAo0, currAo, s\_CDab  
{o4} shufb ao\_n10, prevAo0, currAo, s\_Dabc

{lnop}  
{lnop}  
{lnop}  
{lnop}

{o6} lqx nextAo, pvNextColor, colorOffset  
{o4} shufb ao\_p1, currAo, nextAo, s\_BCDa  
{o4} shufb ao\_p2, currAo, nextAo, s\_CDab  
{o4} shufb ao\_p3, currAo, nextAo, s\_Dabc

{lnop}  
{lnop}  
{lnop}  
{lnop}

# SPU Assembly 2/2

EVEN

```
{e6} fm blurAo, ao_n4, w4  
{e6} fma blurAo, ao_n3, w3, blurAo  
{e6} fma blurAo, ao_n2, w2, blurAo  
{e6} fma blurAo, ao_n1, w1, blurAo  
{e6} fma blurAo, currAo, w0, blurAo
```

```
{nop}  
{nop}  
{nop}  
{nop}  
{nop}
```

```
{e2} ceq endLineMask, colorOffset, endLineOffset
```

```
{e2} ceq endBlockMask, colorOffset, endOffset
```

```
{e2} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask
```

```
{e2} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask {lnop}
```

```
{e2} a endLineOffset, endLineOffset, endLineOffsetIncr
```

```
{e2} a colorOffset, colorOffset, colorOffsetIncr
```

```
{nop}
```

```
{lnop}  
{lnop}  
{lnop}  
{lnop}  
{lnop}  
  
{o6} stqx blurAo, pvColor, colorOffset  
{o4} shlbii ao_n4, currAo, 0  
{o4} shlbii ao_n3, ao_p1, 0  
{o4} shlbii ao_n2, ao_p2, 0  
{o4} shlbii ao_n1, ao_p3, 0
```

```
{lnop}
```

branch: {o?} brz endBlockMask, loop

# SPU Assembly

---

{e6} fa ao\_n3, ao\_n3, ao\_p3 {o6} lqx nextAo, pvNextColor, colorOffset

- Here is a sample line of assembly code
- SPUs issue one even and odd instruction per cycle
- One line = One cycle
- e6 = even, 6 cycle latency
- o6 = odd, 6 cycle latency

# Iteration 1

# Iteration 1

---

loop:

```
nop  
nop  
nop  
nop  
nop  
nop  
nop  
nop  
  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
nop  
nop  
  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask  
  e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

Inop

```
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0
```

Inop

Inop

Inop

```
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc
```

Inop

```
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0
```

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Iteration 1

---

loop:

```
nop  
nop  
nop  
nop  
nop  
nop  
nop  
nop  
  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
nop  
nop  
  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask  
  e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

Inop

```
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0
```

Inop

Inop

Inop

```
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc
```

Inop

```
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0
```

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Problems

---

- Take this line:

```
{o6} lqx currAo, pvColor, colorOffset  
{o4} shufb prevAo0, currAo, currAo, s_AAAA
```
- Load a value into currAo
- Next instruction needs currAo
- currAo won't be ready yet
- Stall for 5 cycles

# Iteration 1

---

loop:

```
nop  
nop  
nop  
nop  
nop  
nop  
nop  
nop  
  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
nop  
nop  
  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask  
  e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

Inop

```
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0
```

Inop

Inop

Inop

```
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc
```

Inop

```
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0
```

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Iteration 1

---

loop:

```
nop  
nop  
nop  
nop  
nop  
nop  
nop  
nop  
  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
nop  
nop  
  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask  
  e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

Inop

```
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0
```

Inop

Inop

Inop

```
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc
```

Inop

```
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0
```

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Iteration 2

loop:

```
nop  
{e6:1} fa ao_n4, ao_n4, nextAo  
{e6:1} fa ao_n3, ao_n3, ao_p3  
nop  
{e6:1} fa ao_n1_, ao_n1, ao_p1  
{e6:1} fa ao_n2_, ao_n2, ao_p2  
nop  
{e6:1} fm blurAo, ao_n4, w4  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
nop  
{e6:1} fma blurAo, ao_n3, w3, blurAo  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask_  
    e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

Inop

```
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlqbii endLineMask_, endLineMask, 0
```

Inop

Inop

Inop

```
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc
```

Inop

```
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0
```

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Iteration 3

loop:

nop

{e6:1} fa ao\_n4, ao\_n4, nextAo

{e6:1} fa ao\_n3, ao\_n3, ao\_p3

{e6:2} fma blurAo\_, ao\_n2\_, w2, blurAo

{e6:1} fa ao\_n1\_, ao\_n1, ao\_p1

{e6:1} fa ao\_n2\_, ao\_n2, ao\_p2

nop

{e6:1} fm blurAo, ao\_n4, w4

{e2:0} ceq endLineMask, colorOffset, endLineOffset

{e2:0} ceq endBlockMask, colorOffset\_, endOffset

{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask

{e2:0} selb ao\_n4, currAo\_, prevAo0, endLineMask\_

{e6:2} fma blurAo\_, ao\_n1\_, w1, blurAo\_

{e6:1} fma blurAo, ao\_n3, w3, blurAo

{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask

  e2:0} selb ao\_n3, ao\_p1\_, ao\_n30, endLineMask\_

{e2:0} selb ao\_n2, ao\_p2\_, ao\_n20, endLineMask\_

{e2:0} selb ao\_n1, ao\_p3\_, ao\_n10, endLineMask

{e2:0} a colorOffset, colorOffset, colorOffsetIncr

{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr

Inop

{o6:0} lqx currAo, pvColor, colorOffset

{o6:0} lqx nextAo, pvNextColor, colorOffset

{o4:0} shlqbii endLineMask\_, endLineMask, 0

Inop

Inop

Inop

{o4:0} shufb prevAo0, currAo, currAo, s\_AAAA

{o4:0} shufb ao\_p1, currAo, nextAo, s\_BCDa

{o4:0} shufb ao\_p2, currAo, nextAo, s\_CDab

{o4:0} shufb ao\_p3, currAo, nextAo, s\_Dabc

{o4:0} shufb ao\_n30, prevAo0, currAo, s\_BCDa

{o4:0} shufb ao\_n20, prevAo0, currAo, s\_CDab

{o4:0} shufb ao\_n10, prevAo0, currAo, s\_Dabc

Inop

{o4:0} shufb colorOffset\_, colorOffset\_, colorOffset, s\_BCa0

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Iteration 4

loop:

nop

{e6:1} fa ao\_n4, ao\_n4, nextAo

{e6:1} fa ao\_n3, ao\_n3, ao\_p3

{e6:2} fma blurAo\_, ao\_n2\_, w2, blurAo

{e6:1} fa ao\_n1\_, ao\_n1, ao\_p1

{e6:1} fa ao\_n2\_, ao\_n2, ao\_p2

{e6:3} fma blurAo\_\_\_\_, currAo\_\_\_\_, w0, blurAo\_\_\_\_

{e6:1} fm blurAo, ao\_n4, w4

{e2:0} ceq endLineMask, colorOffset, endLineOffset

{e2:0} ceq endBlockMask, colorOffset\_, endOffset

{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask

{e2:0} selb ao\_n4, currAo\_, prevAo0, endLineMask\_

{e6:2} fma blurAo\_\_\_\_, ao\_n1\_\_\_\_, w1, blurAo\_

{e6:1} fma blurAo, ao\_n3, w3, blurAo

{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask {o6:3} stqx blurAo\_\_\_\_, pvColor, colorOffset\_

e2:0} selb ao\_n3, ao\_p1\_, ao\_n30, endLineMask\_

{e2:0} selb ao\_n2, ao\_p2\_, ao\_n20, endLineMask\_

{e2:0} selb ao\_n1, ao\_p3\_, ao\_n10, endLineMask

{e2:0} a colorOffset, colorOffset, colorOffsetIncr

{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr

Inop

{o6:0} lqx currAo, pvColor, colorOffset

{o6:0} lqx nextAo, pvNextColor, colorOffset

{o4:0} shlbii endLineMask\_, endLineMask, 0

Inop

Inop

Inop

{o4:0} shufb prevAo0, currAo, currAo, s\_AAAA

{o4:0} shufb ao\_p1, currAo, nextAo, s\_BCDa

{o4:0} shufb ao\_p2, currAo, nextAo, s\_CDab

{o4:0} shufb ao\_p3, currAo, nextAo, s\_Dabc

{o4:0} shufb ao\_n30, prevAo0, currAo, s\_BCDa

{o4:0} shufb ao\_n20, prevAo0, currAo, s\_CDab

{o4:0} shufb ao\_n10, prevAo0, currAo, s\_Dabc

{o4:0} shufb colorOffset\_, colorOffset\_, colorOffset, s\_BCa0

Inop

Inop

Inop

branch: {o?:0} brz endBlockMask, loop

# Copy Operations

loop:

```
{e2:x} ai ao_n1__, ao_n1__, 0
{e6:1} fa ao_n4, ao_n4, nextAo
{e6:1} fa ao_n3, ao_n3, ao_p3
{e6:2} fma blurAo__, ao_n2__, w2, blurAo
{e6:1} fa ao_n1__, ao_n1, ao_p1
{e6:1} fa ao_n2__, ao_n2, ao_p2
{e6:3} fma blurAo____, currAo____, w0, blurAo____
{e6:1} fm blurAo, ao_n4, w4
{e2:0} ceq endLineMask, colorOffset, endLineOffset
{e2:0} ceq endBlockMask, colorOffset__, endOffset
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask
{e2:0} selb ao_n4, currAo__, prevAo0, endLineMask__
{e6:2} fma blurAo____, ao_n1__, w1, blurAo____
{e6:1} fma blurAo, ao_n3, w3, blurAo____
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask_____
{e2:0} selb ao_n3, ao_p1__, ao_n30, endLineMask__
{e2:0} selb ao_n2, ao_p2__, ao_n20, endLineMask__
{e2:0} selb ao_n1, ao_p3__, ao_n10, endLineMask__
{e2:0} a colorOffset, colorOffset, colorOffsetIncr
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

```
{o4:x} shlbii currAo__, currAo, 0
{o6:0} lqx currAo, pvColor, colorOffset
{o6:0} lqx nextAo, pvNextColor, colorOffset
{o4:0} shlbii endLineMask__, endLineMask, 0
{o4:x} shlbii ao_p1__, ao_p1, 0
{o4:x} shlbii ao_p2__, ao_p2, 0
{o4:x} shlbii ao_p3__, ao_p3, 0
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc
{o6:3} stqx blurAo____, pvColor, colorOffset_____
{o4:0} shufb colorOffset__, colorOffset__, colorOffset, s_BCa0
{o4:x} shlbii currAo_____, currAo_____, 0
{o4:x} shlbii currAo_____, currAo_____, 0
lnop
branch: {o?:0} brz endBlockMask, loop
```

# Optimized

---

```
loop:  
{e2:x} ai ao_n1__, ao_n1_, 0  
{e6:1} fa ao_n4, ao_n4, nextAo  
{e6:1} fa ao_n3, ao_n3, ao_p3  
{e6:2} fma blurAo_, ao_n2_, w2, blurAo  
{e6:1} fa ao_n1_, ao_n1, ao_p1  
{e6:1} fa ao_n2_, ao_n2, ao_p2  
{e6:3} fma blurAo____, currAo____, w0, blurAo____  
{e6:1} fm blurAo, ao_n4, w4  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
{e6:2} fma blurAo____, ao_n1__, w1, blurAo____  
{e6:1} fma blurAo, ao_n3, w3, blurAo____  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask  
{e6:3} stqx blurAo____, pvColor, colorOffset____  
{e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask_  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr  
  
{o4:x} shlbii currAo_, currAo, 0  
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0  
{o4:x} shlbii ao_p1_, ao_p1, 0  
{o4:x} shlbii ao_p2_, ao_p2, 0  
{o4:x} shlbii ao_p3_, ao_p3, 0  
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc  
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0  
{o4:x} shlbii currAo____, currAo____, 0  
{o4:x} shlbii currAo____, currAo____, 0  
Inop  
branch: {o?:0} brz endBlockMask, loop
```

# Optimized

---

EVEN

```
loop:  
{e2:x} ai ao_n1__, ao_n1_, 0  
{e6:1} fa ao_n4, ao_n4, nextAo  
{e6:1} fa ao_n3, ao_n3, ao_p3  
{e6:2} fma blurAo_, ao_n2_, w2, blurAo  
{e6:1} fa ao_n1_, ao_n1, ao_p1  
{e6:1} fa ao_n2_, ao_n2, ao_p2  
{e6:3} fma blurAo____, currAo____, w0, blurAo____  
{e6:1} fm blurAo, ao_n4, w4  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
{e6:2} fma blurAo____, ao_n1____, w1, blurAo____  
{e6:1} fma blurAo, ao_n3, w3, blurAo  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask{e6:3} stqx blurAo____, pvColor, colorOffset_____  
{e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask_  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

ODD

```
{o4:x} shlqbii currAo_, currAo, 0  
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlqbii endLineMask_, endLineMask, 0  
{o4:x} shlqbii ao_p1_, ao_p1, 0  
{o4:x} shlqbii ao_p2_, ao_p2, 0  
{o4:x} shlqbii ao_p3_, ao_p3, 0  
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc  
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0  
{o4:x} shlqbii currAo____, currAo____, 0  
{o4:x} shlqbii currAo____, currAo____, 0  
lnop  
branch: {o?:0} brz endBlockMask, loop
```

# Optimized

---

## EVEN

```
loop:  
{e2:x} ai ao_n1__, ao_n1_, 0  
{e6:1} fa ao_n4, ao_n4, nextAo  
{e6:1} fa ao_n3, ao_n3, ao_p3  
{e6:2} fma blurAo_, ao_n2_, w2, blurAo  
{e6:1} fa ao_n1_, ao_n1, ao_p1  
{e6:1} fa ao_n2_, ao_n2, ao_p2  
{e6:3} fma blurAo____, currAo____, w0, blurAo____  
{e6:1} fm blurAo, ao_n4, w4  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
{e6:2} fma blurAo____, ao_n1____, w1, blurAo____  
{e6:1} fma blurAo, ao_n3, w3, blurAo  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask{e6:3} stqx blurAo____, pvColor, colorOffset_____  
{e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask_  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

## ODD

```
{o4:x} shlbii currAo_, currAo, 0  
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0  
{o4:x} shlbii ao_p1_, ao_p1, 0  
{o4:x} shlbii ao_p2_, ao_p2, 0  
{o4:x} shlbii ao_p3_, ao_p3, 0  
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc  
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0  
{o4:x} shlbii currAo____, currAo____, 0  
{o4:x} shlbii currAo____, currAo____, 0  
lnop  
branch: {o?:0} brz endBlockMask, loop
```

# Optimized

---

EVEN

```
loop:  
{e2:x} ai ao_n1__, ao_n1_, 0  
{e6:1} fa ao_n4, ao_n4, nextAo  
{e6:1} fa ao_n3, ao_n3, ao_p3  
{e6:2} fma blurAo_, ao_n2_, w2, blurAo  
{e6:1} fa ao_n1_, ao_n1, ao_p1  
{e6:1} fa ao_n2_, ao_n2, ao_p2  
{e6:3} fma blurAo____, currAo____, w0, blurAo____  
{e6:1} fm blurAo, ao_n4, w4  
{e2:0} ceq endLineMask, colorOffset, endLineOffset  
{e2:0} ceq endBlockMask, colorOffset_, endOffset  
{e2:0} selb endLineOffsetIncr, zero, colorLineBytes, endLineMask  
{e2:0} selb ao_n4, currAo_, prevAo0, endLineMask_  
{e6:2} fma blurAo____, ao_n1____, w1, blurAo____  
{e6:1} fma blurAo, ao_n3, w3, blurAo  
{e2:0} selb colorOffsetIncr, defOffsetIncr, endColorOffsetIncr, endLineMask{e6:3} stqx blurAo____, pvColor, colorOffset_____  
{e2:0} selb ao_n3, ao_p1_, ao_n30, endLineMask_  
{e2:0} selb ao_n2, ao_p2_, ao_n20, endLineMask_  
{e2:0} selb ao_n1, ao_p3_, ao_n10, endLineMask_  
{e2:0} a colorOffset, colorOffset, colorOffsetIncr  
{e2:0} a endLineOffset, endLineOffset, endLineOffsetIncr
```

ODD

```
{o4:x} shlbii currAo_, currAo, 0  
{o6:0} lqx currAo, pvColor, colorOffset  
{o6:0} lqx nextAo, pvNextColor, colorOffset  
{o4:0} shlbii endLineMask_, endLineMask, 0  
{o4:x} shlbii ao_p1_, ao_p1, 0  
{o4:x} shlbii ao_p2_, ao_p2, 0  
{o4:x} shlbii ao_p3_, ao_p3, 0  
{o4:0} shufb prevAo0, currAo, currAo, s_AAAA  
{o4:0} shufb ao_p1, currAo, nextAo, s_BCDa  
{o4:0} shufb ao_p2, currAo, nextAo, s_CDab  
{o4:0} shufb ao_p3, currAo, nextAo, s_Dabc  
{o4:0} shufb ao_n30, prevAo0, currAo, s_BCDa  
{o4:0} shufb ao_n20, prevAo0, currAo, s_CDab  
{o4:0} shufb ao_n10, prevAo0, currAo, s_Dabc  
{o4:0} shufb colorOffset_, colorOffset_, colorOffset, s_BCa0  
{o4:x} shlbii currAo____, currAo____, 0  
{o4:x} shlbii currAo____, currAo____, 0  
lnop  
branch: {o?:0} brz endBlockMask, loop
```

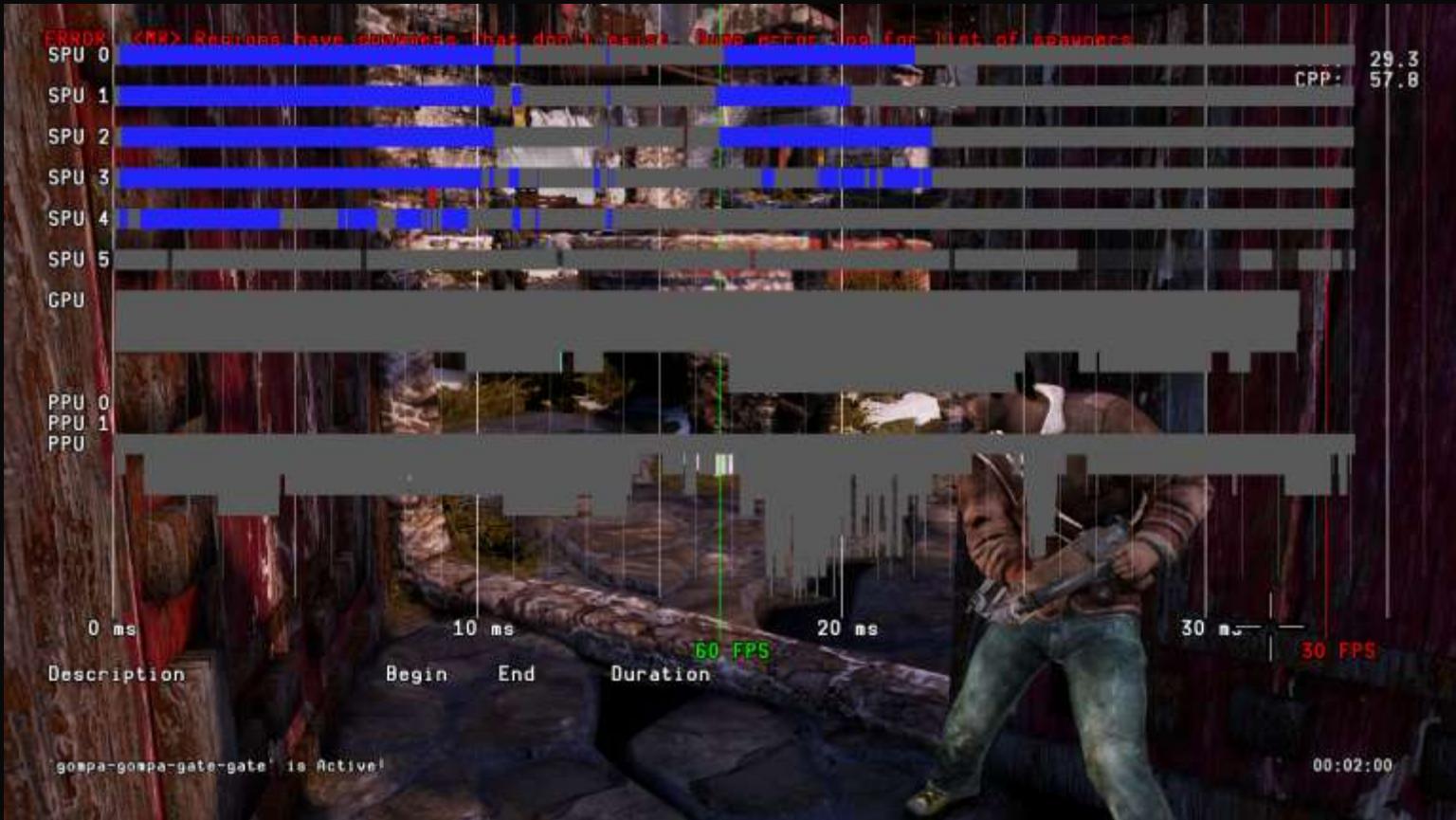
# Solution

---

- Gives about a 2x performance boost
- Sometimes more or less
- SSAO went from 2ms on all 6 SPUs to 1ms on all 6 SPUs
- With practice, can do about 1-2 loops per day.
  - SSAO has 6 loops, and took a little over a week.
- Same process for:
  - PostFX
  - Fullscreen Lighting
  - Ambient Cubemaps
  - Many more

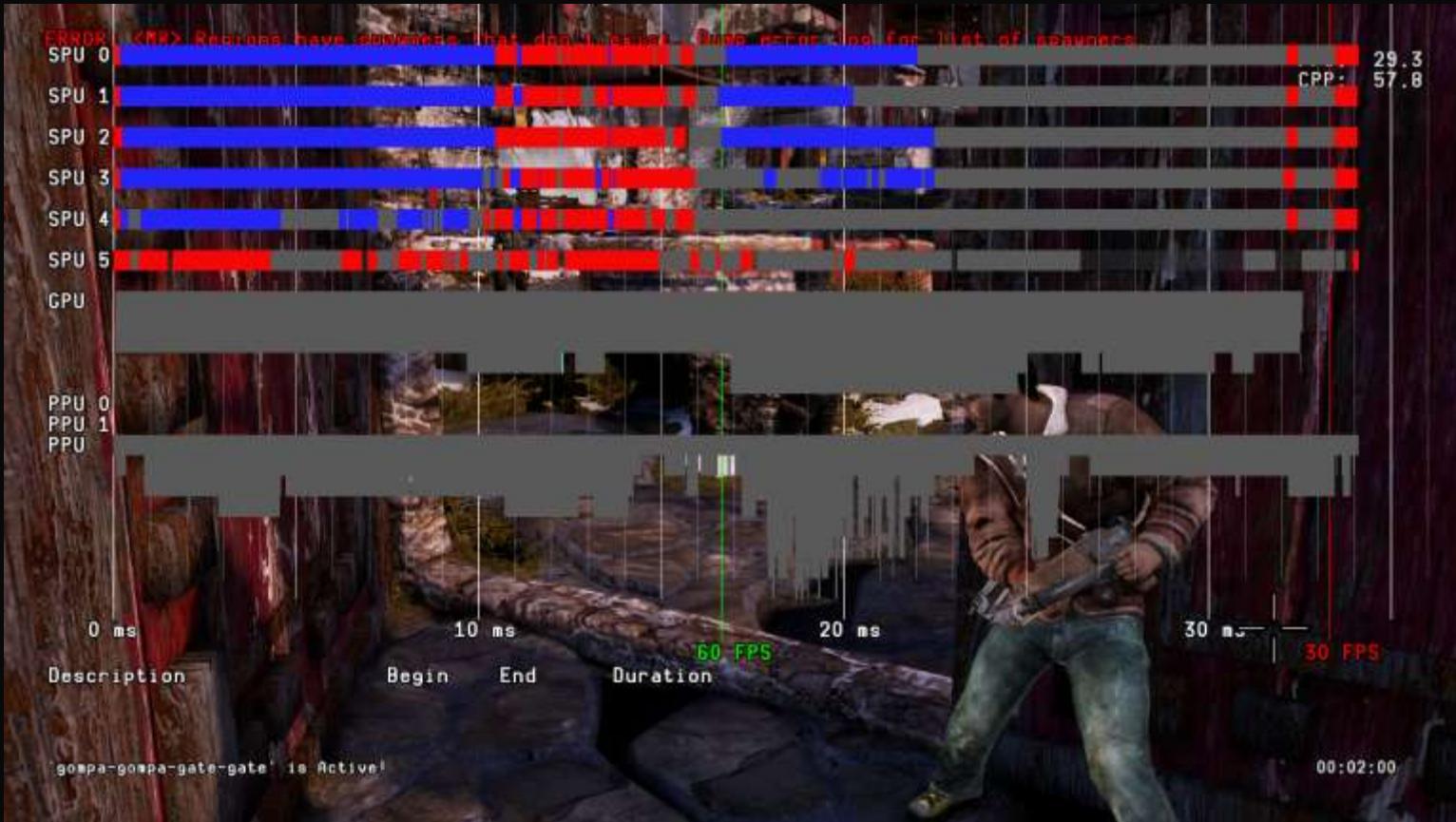
# Full Timeline

- Hand-Optimized by ICE Team



# Full Timeline

- Hand-Optimized by Uncharted 2 Team



# Performance

---

- Final Render



# Performance

---

- Without Fullscreen Lighting



# Performance

---

- Fullscreen Lighting Results



# Performance

- Fullscreen Lighting / SSAO Performance



# Performance

- Fullscreen Lighting / SSAO Performance



# SSAO

# Performance

---

- Fullscreen Lighting / SSAO Performance



# Performance

---

- Fullscreen Lighting / SSAO Performance



# Performance

- Fullscreen Lighting / SSAO Performance



# Performance

- Fullscreen Lighting / SSAO Performance



# Architecture Conclusions

---

“So, you are the scheduler that has been nipping at my heels...”



# Final Final Thoughts

---

- Gamma is super-duper important.
- Filmic Tonemapping changes your life.
- SSAO is cool, and keep it out of your sunlight.
- SPUs are awesome.
  - They don't optimize themselves.
  - For a tight for-loop, you can do about 2x better than the compiler.

# Btw...

---

- Naughty Dog is hiring!
- We're also looking for
  - Senior Lighting Artist.
  - Graphics Programmer

# That's it!

---

- Questions...

