SIGGRAPH2010

The People Behind the Pixels

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

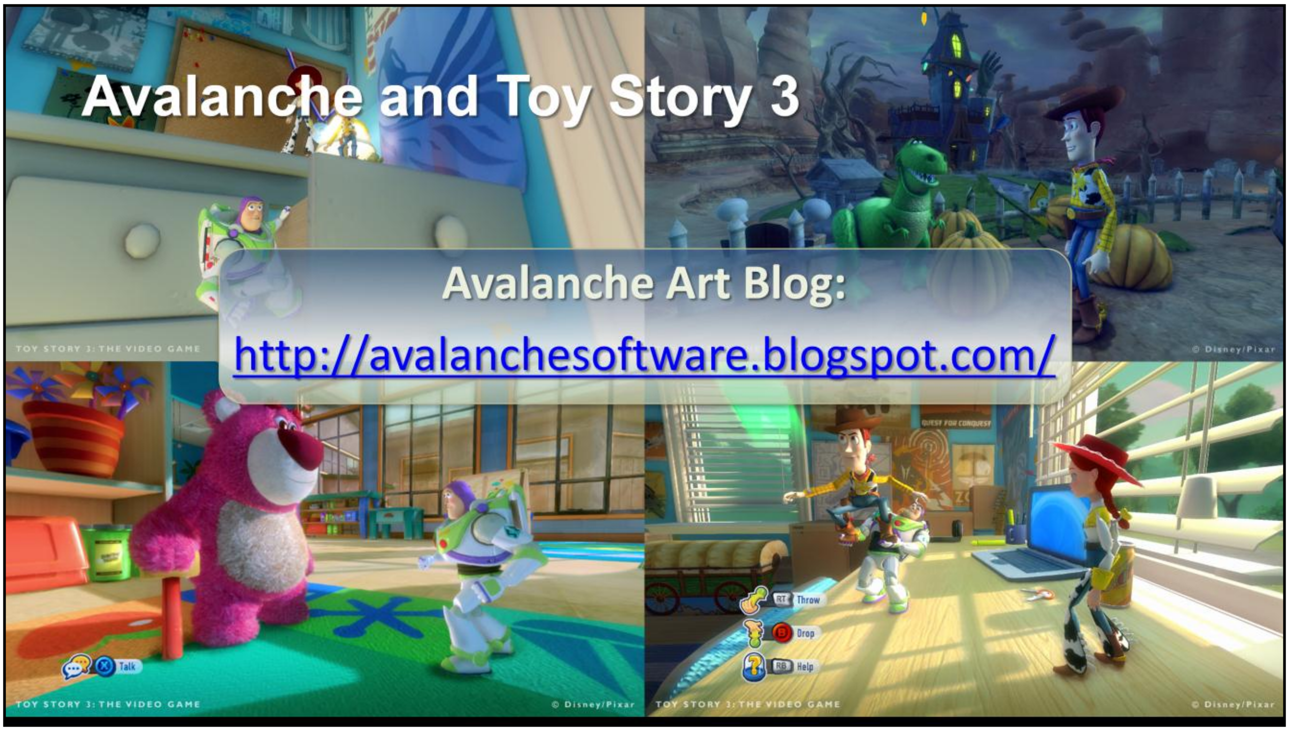
Toy Story 3: The Video Game Rendering Techniques

John-Paul Ownby

Chris Hall

Rob Hall

Avalanche Software



Avalanche and Toy Story 3

Avalanche Art Blog:

http://avalanchesoftware.blogspot.com/

Being able to work on the Toy Story 3 video game was a great experience, but we at Avalanche also felt a big responsibility to try and create a video game that would live up to the incredible film that Pixar was making. On the graphics side, for us this meant developing the technology our artists needed to realize their vision and create a look for our game that we would be proud of. The following scene is rendered in our engine, and all of the art assets and animation were done by our artists at Avalanche.

Please watch “RexVideoGame.avi” ☺

Outline

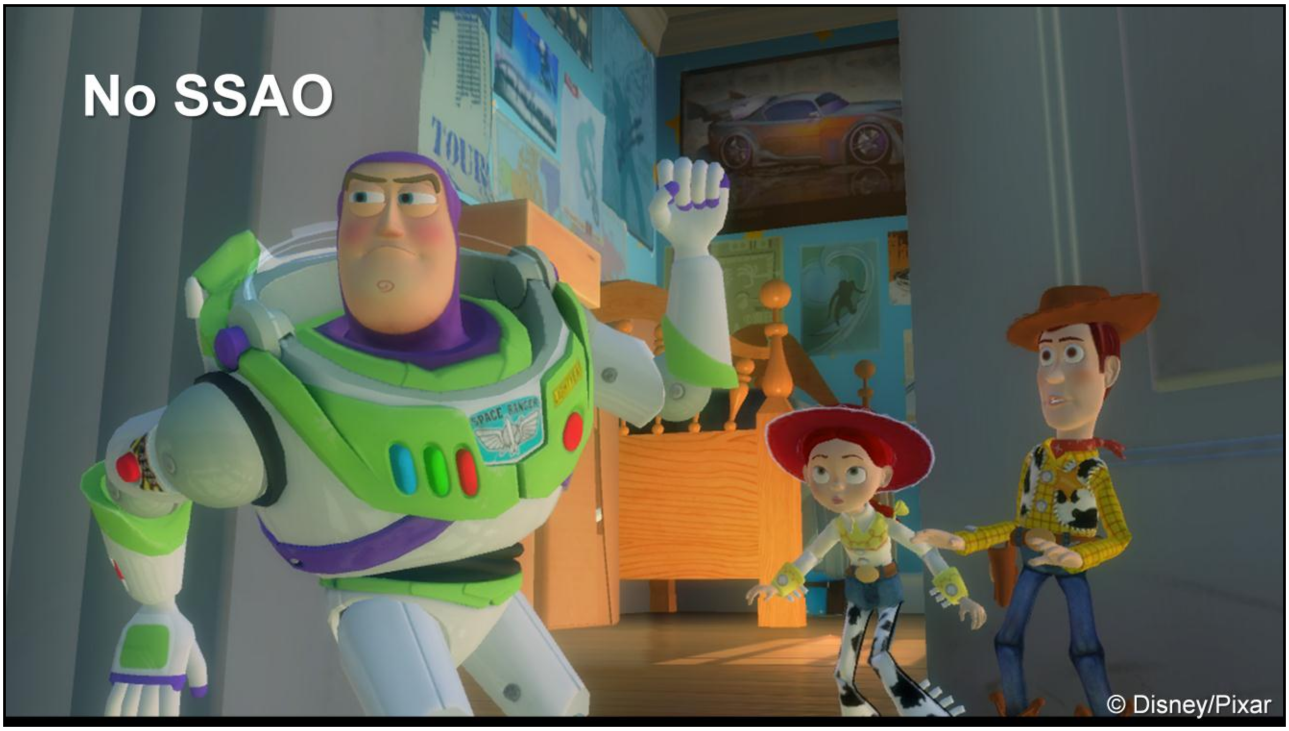
- Screen Space Ambient Occlusion
- Ambient Lighting
- Shadows

John-Paul Ownby
john.ownby@disney.com

SCREEN SPACE AMBIENT OCCLUSION

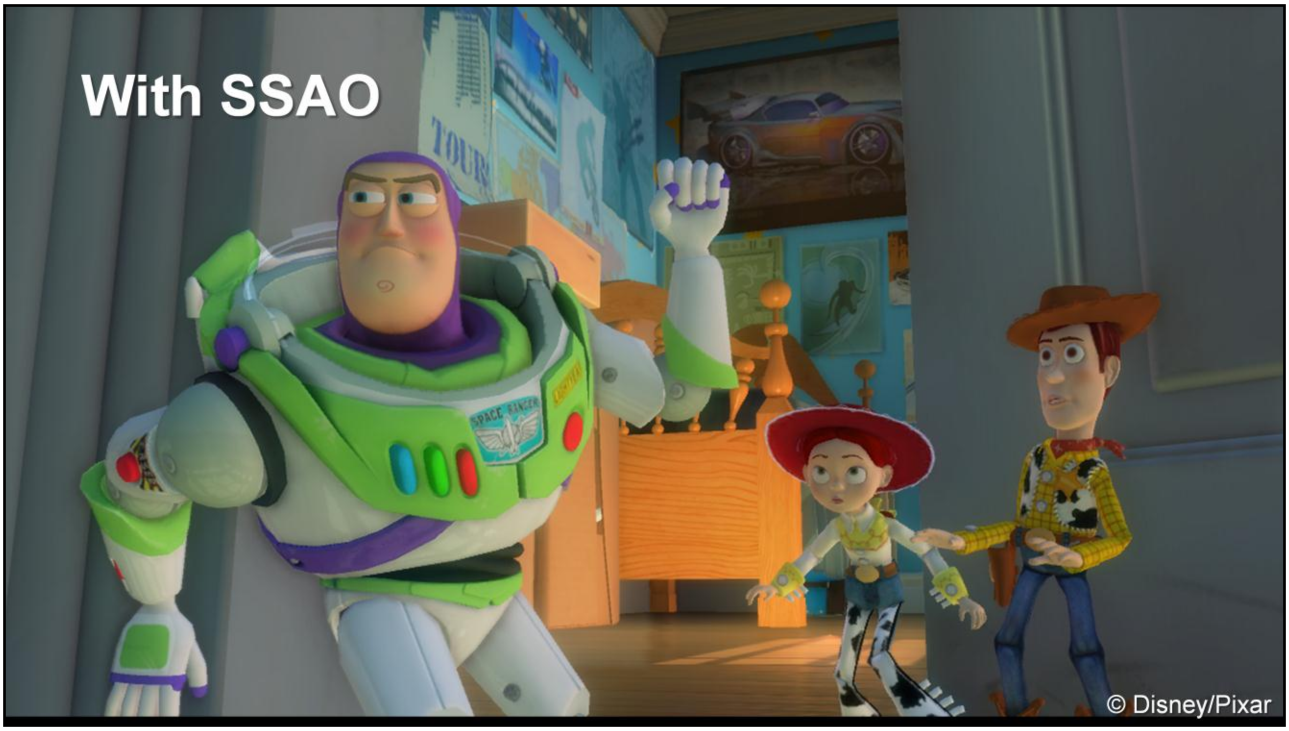
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

TOUR
SPACE RANGER
© Disney/Pixar



No SSAO

© Disney/Pixar

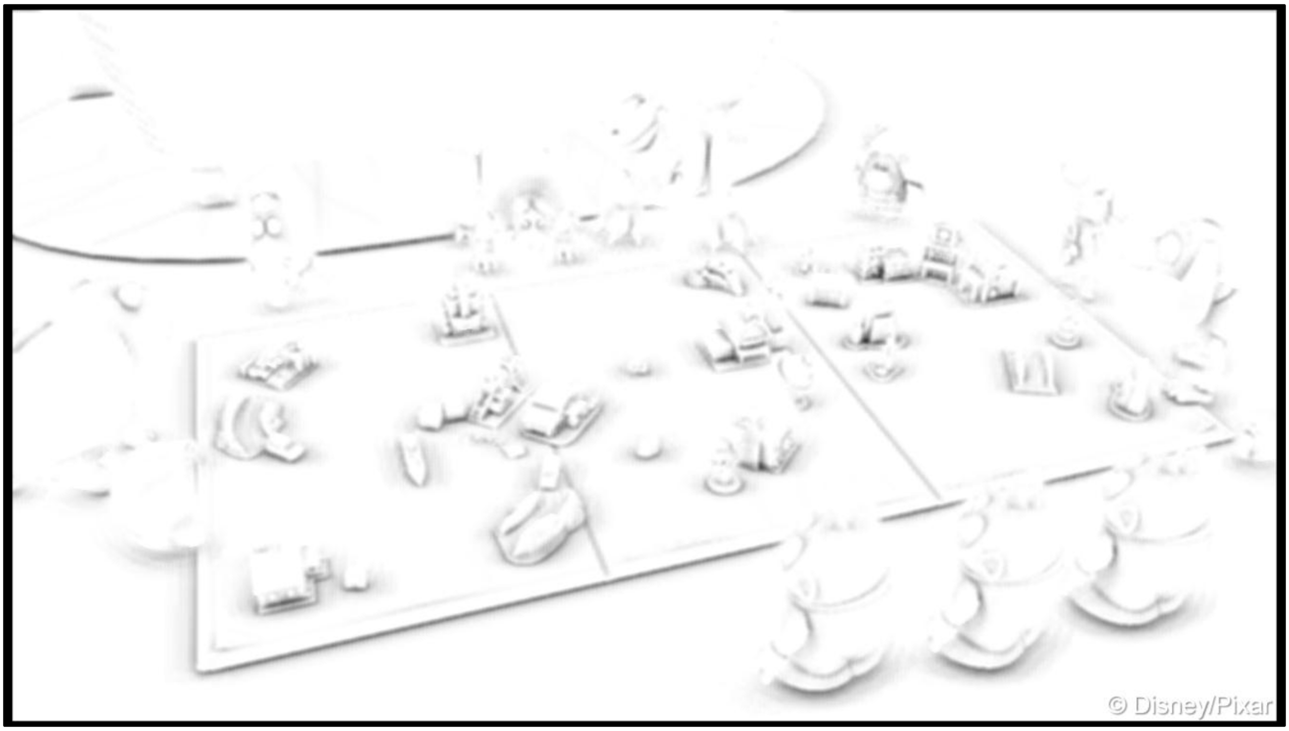


With SSAO

© Disney/Pixar

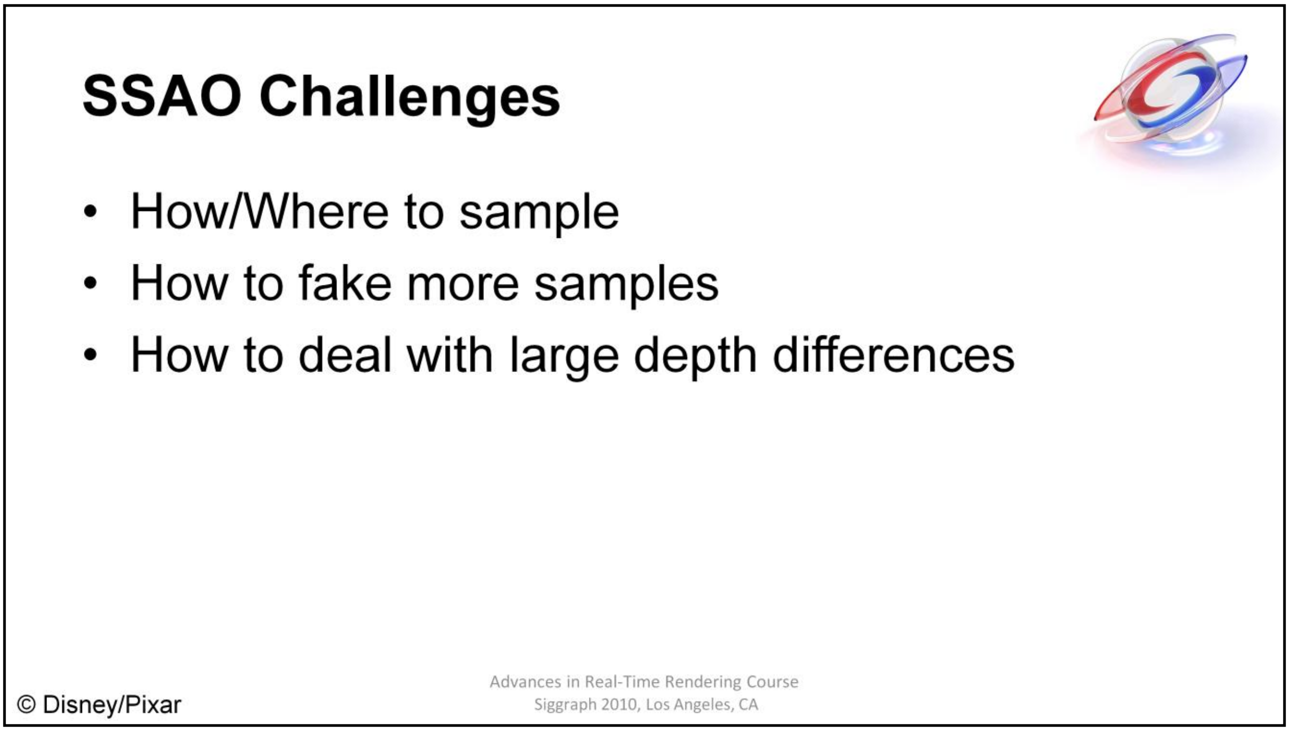
© Disney/Pixar

© Disney/Pixar



© Disney/Pixar

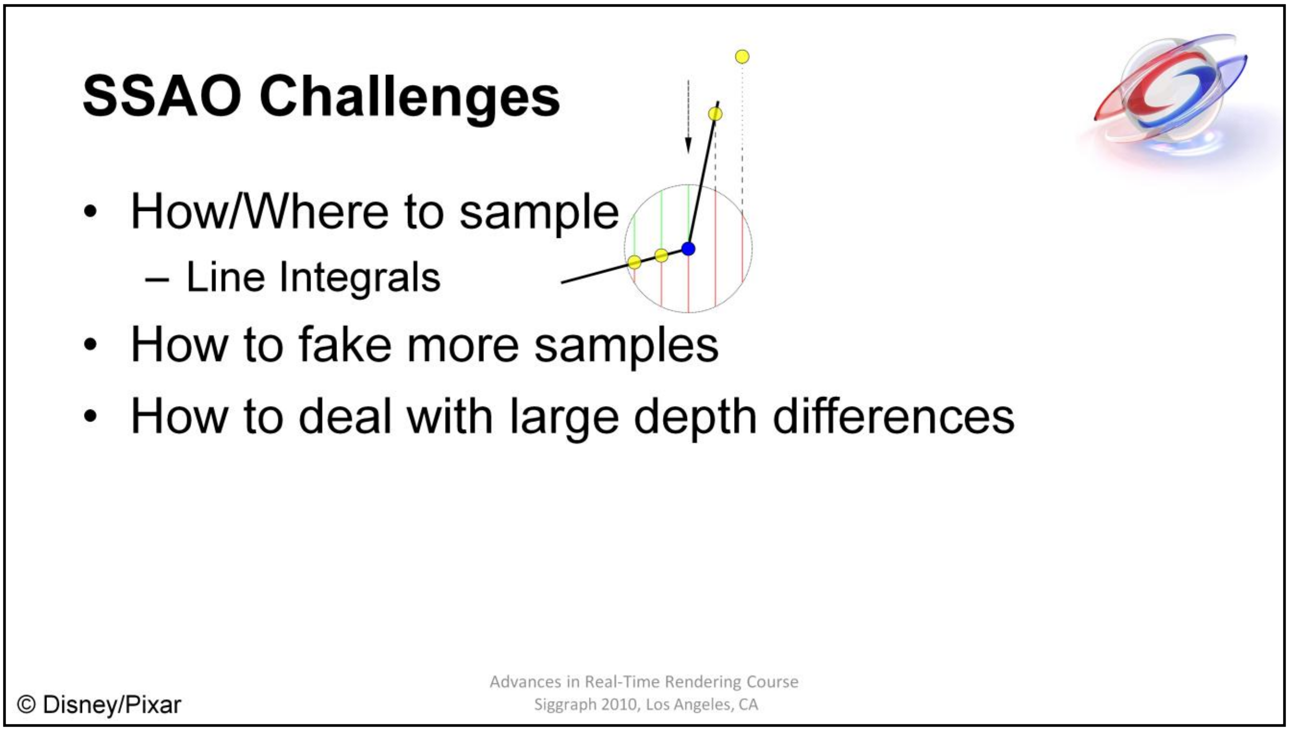
© Disney/Pixar



SSAO Challenges

- How/Where to sample
- How to fake more samples
- How to deal with large depth differences

There are basically three problems to solve when implementing SSAO. I will talk about what we do differently than others for each of the three.



SSAO Challenges

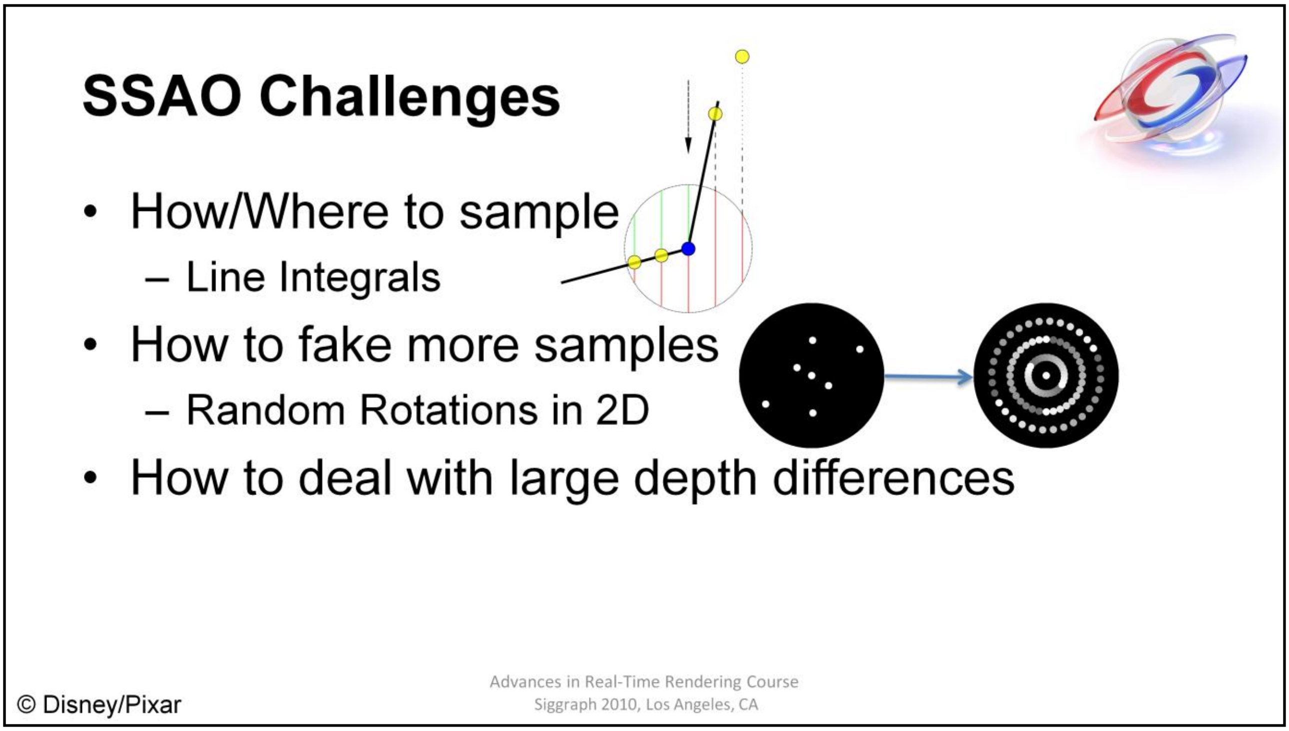
- How/Where to sample
  - – Line Integrals
- How to fake more samples
- How to deal with large depth differences

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Preview:

1) We use line integrals, which allows us to sample in 2D rather than 3D
2) This means we can rotate the sampling pattern in 2D
3) Since it’s in 2D, we can pair samples to better estimate invalid samples



SSAO Challenges

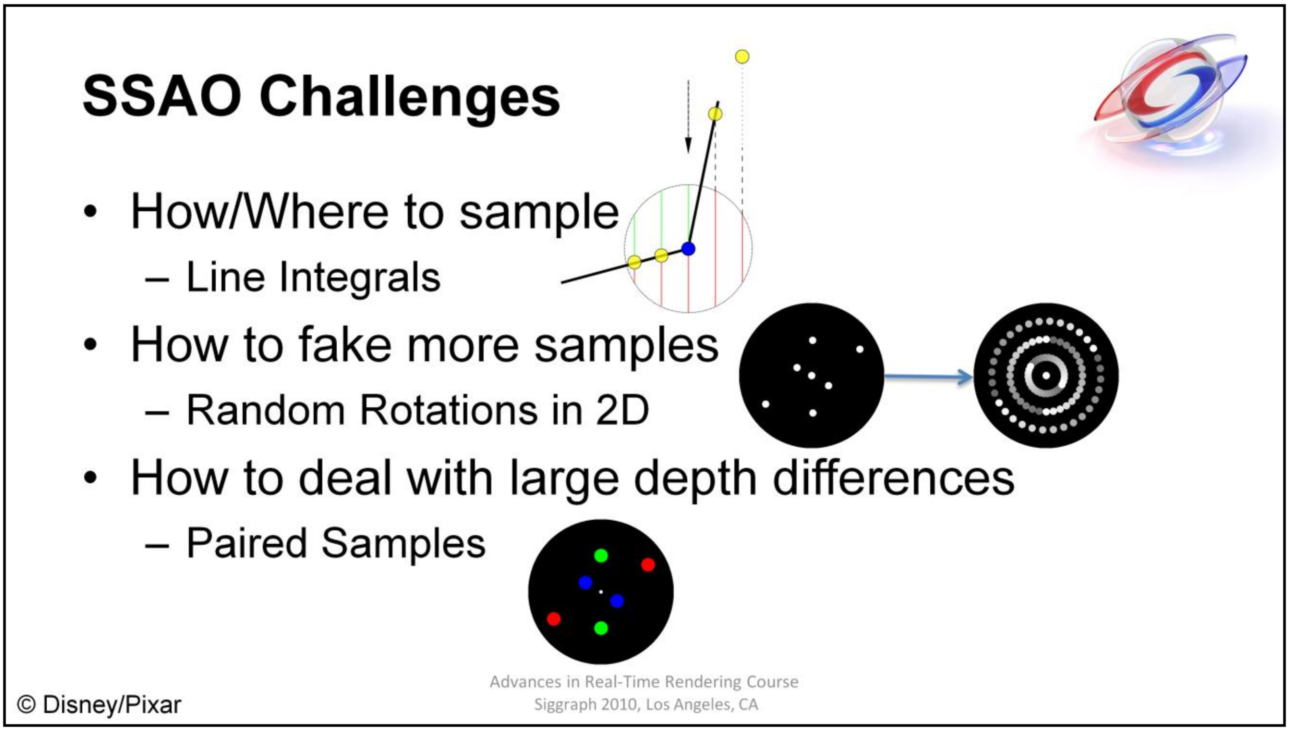
- How/Where to sample
  - Line Integrals
- How to fake more samples
  - Random Rotations in 2D
- How to deal with large depth differences

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Preview:

1) We use line integrals, which allows us to sample in 2D rather than 3D
2) This means we can rotate the sampling pattern in 2D
3) Since it's in 2D, we can pair samples to better estimate invalid samples



SSAO Challenges

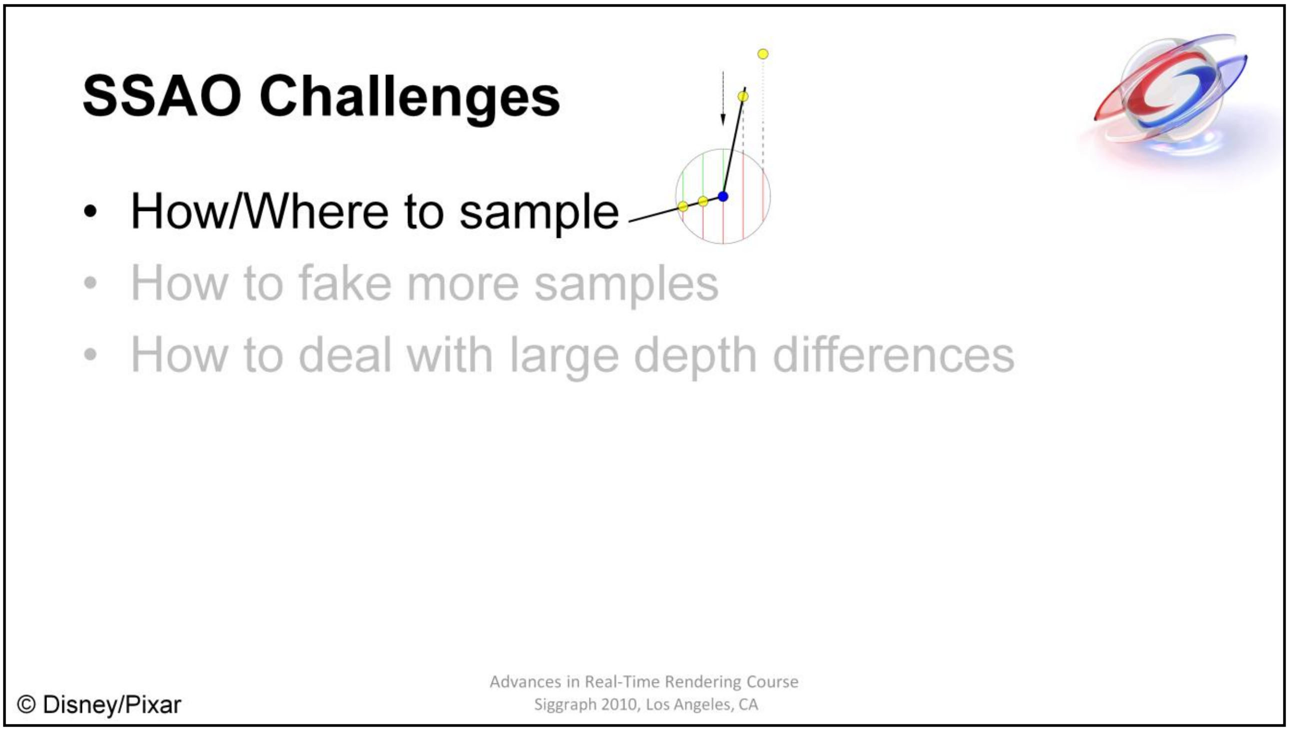
- How/Where to sample
  - – Line Integrals
- How to fake more samples
  - – Random Rotations in 2D
- How to deal with large depth differences
  - – Paired Samples

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

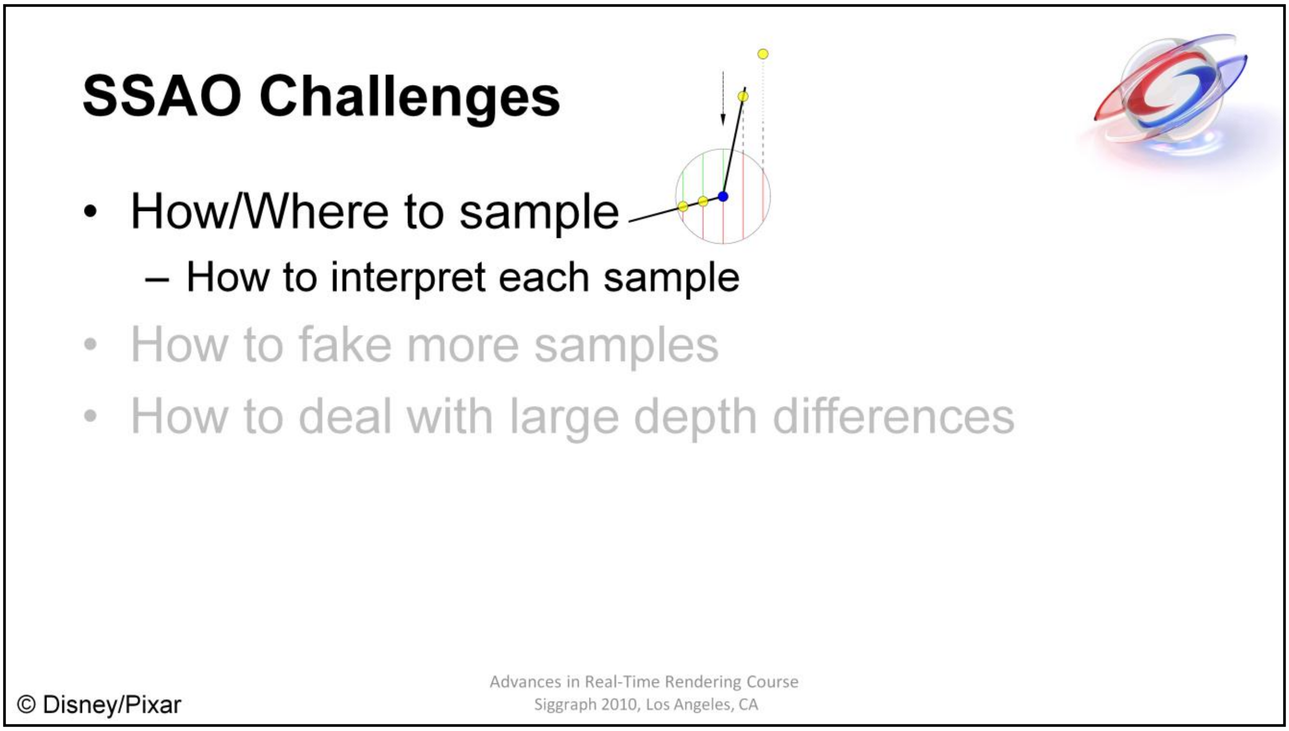
Preview:

1) We use line integrals, which allows us to sample in 2D rather than 3D
2) This means we can rotate the sampling pattern in 2D
3) Since it’s in 2D, we can pair samples to better estimate invalid samples



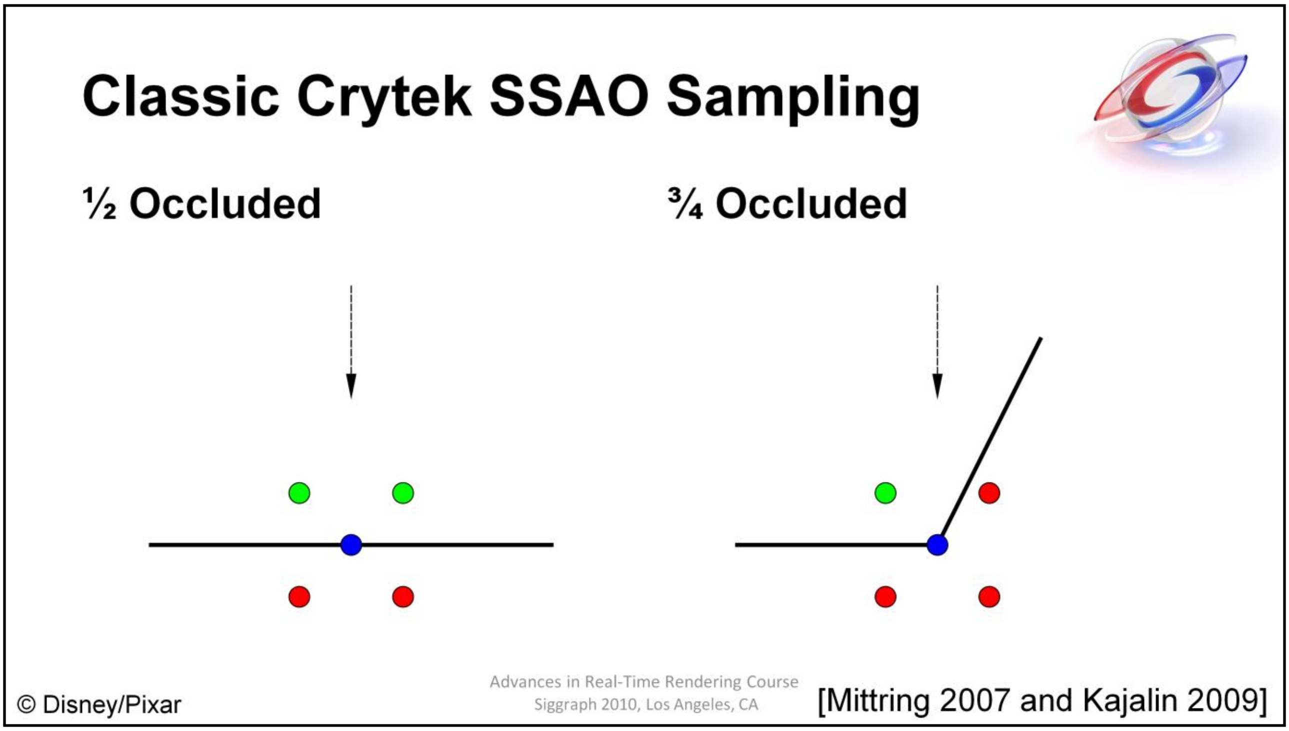
SSAO Challenges

- How/Where to sample
- How to fake more samples
- How to deal with large depth differences



SSAO Challenges

- How/Where to sample
  - How to interpret each sample
- How to fake more samples
- How to deal with large depth differences



Classic Crytek SSAO Sampling

½ Occluded

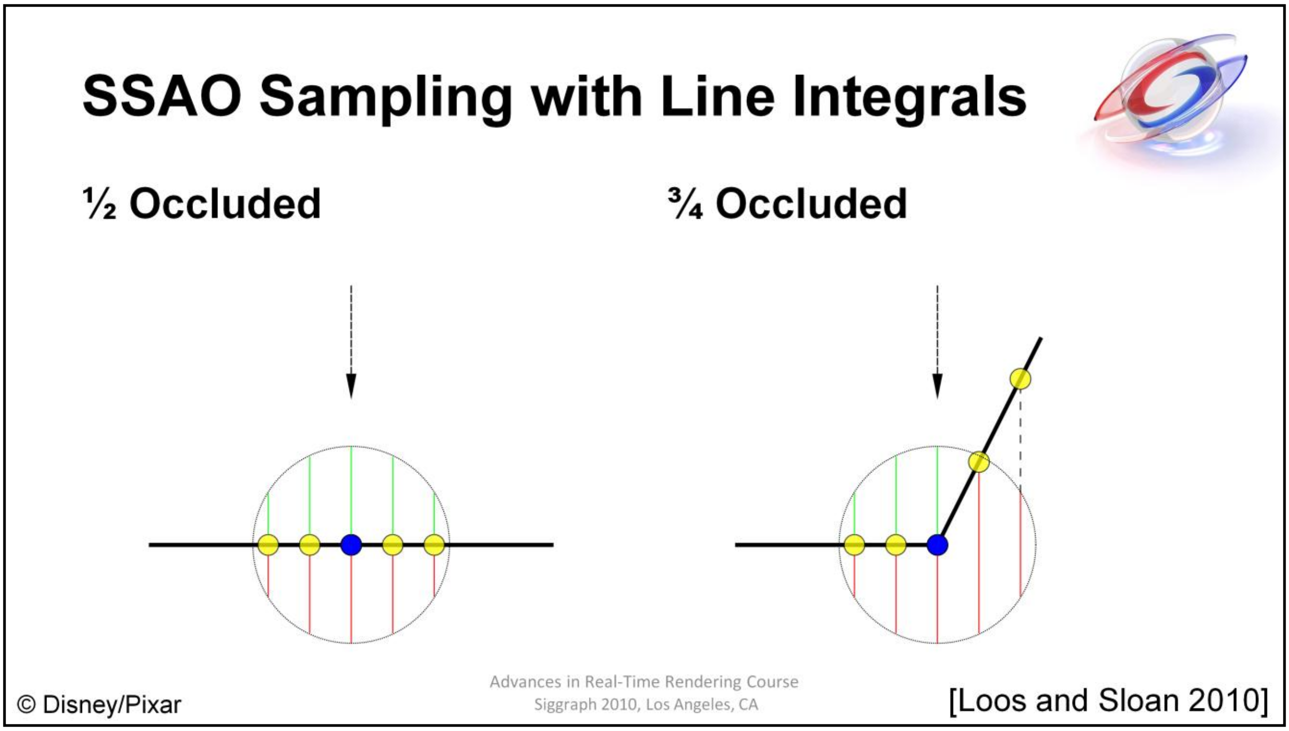
¾ Occluded

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

[Mittring 2007 and Kajalin 2009]

This is an overhead view (the arrow shows the direction the camera is facing). The blue dot is the pixel we are shading. The line going through the blue dot on the left can be thought of as a wall parallel with the view plane, and the line on the right can be thought of as a corner. The four dots are 3D point samples that are either occluded or unoccluded.



SSAO Sampling with Line Integrals

½ Occluded

¾ Occluded

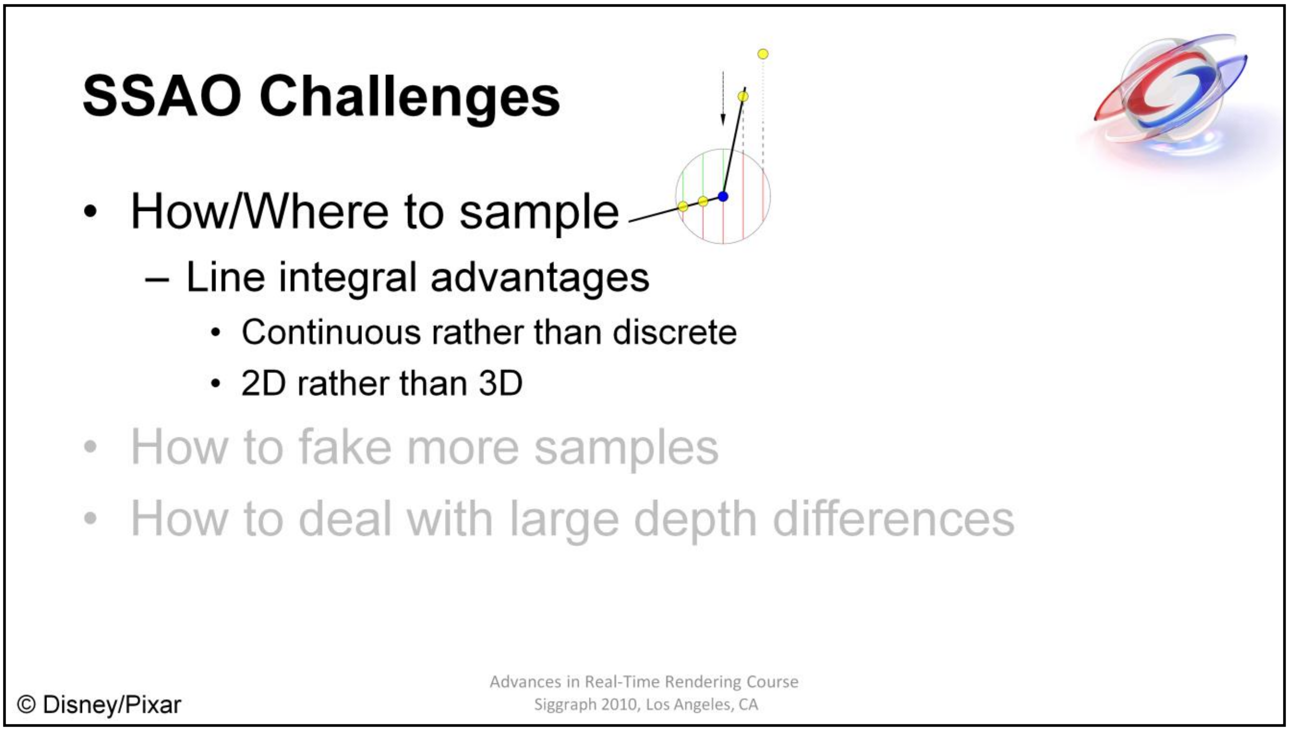
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

[Loos and Sloan 2010]

The blue dot is still the pixel we are sampling. We imagine a conceptual volume sphere surrounding it, and instead of sampling in 3D we sample in 2D. Each sample has a corresponding volume, and depending on the depth of the sample a fractional amount of that volume will be occluded.

While implementing SSAO I had many discussions with Peter-Pike Sloan, of Disney Interactive Research. He introduced me to this idea, and he and Brad Loos published a paper on it also.

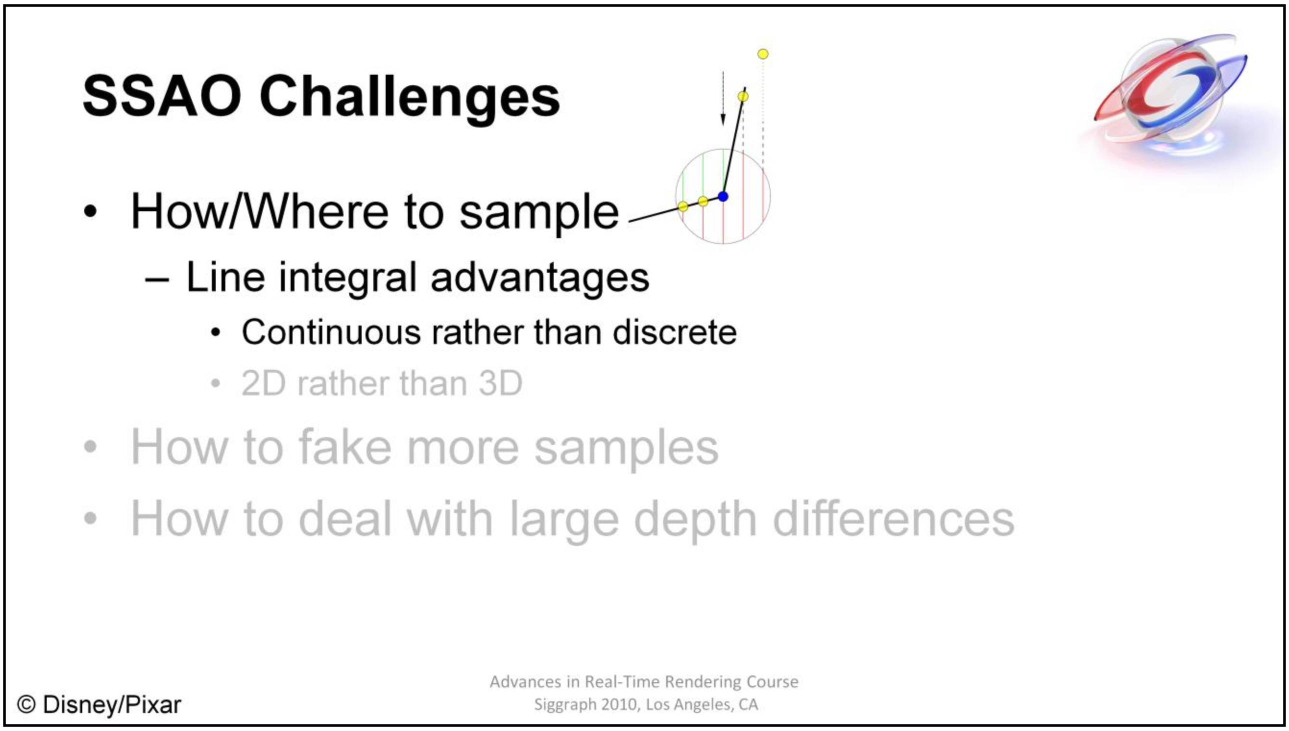


SSAO Challenges

- How/Where to sample
  - Line integral advantages
    - Continuous rather than discrete
    - 2D rather than 3D
- How to fake more samples
- How to deal with large depth differences

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

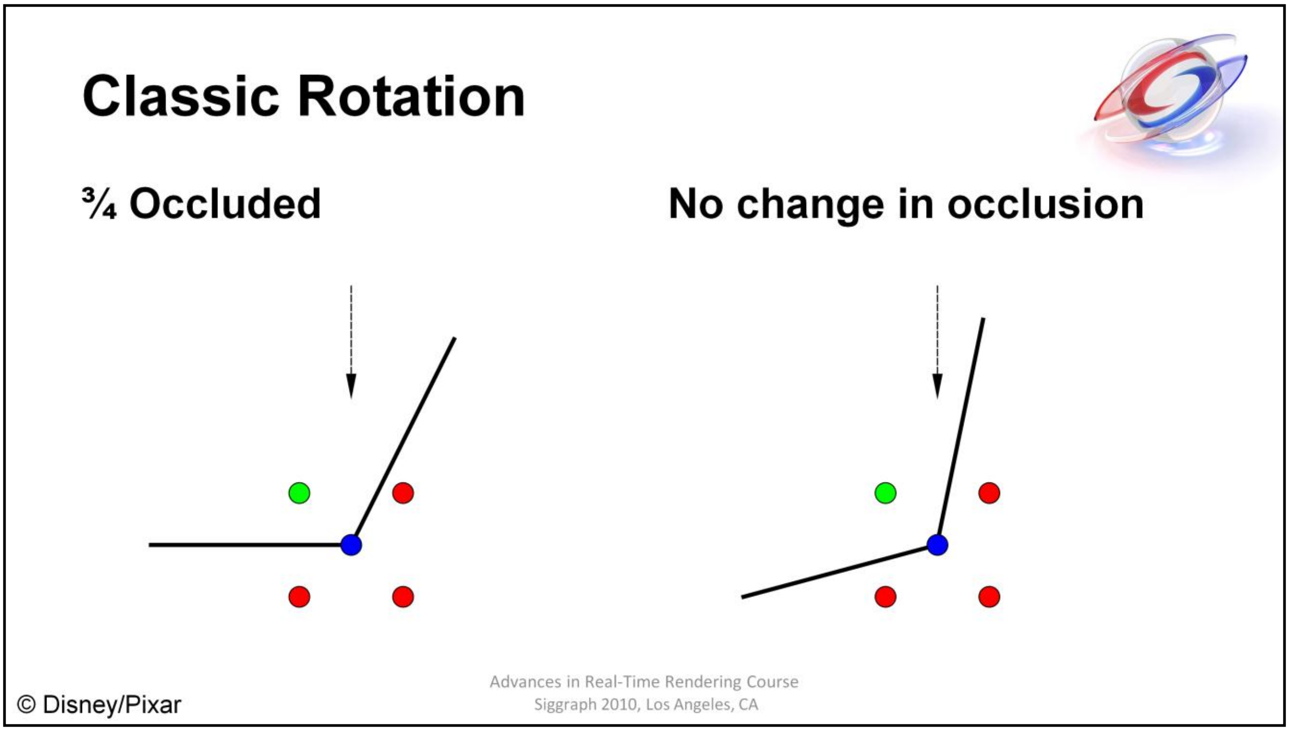


SSAO Challenges

- How/Where to sample
  - Line integral advantages
    - Continuous rather than discrete
    - 2D rather than 3D
- How to fake more samples
- How to deal with large depth differences

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA



Classic Rotation

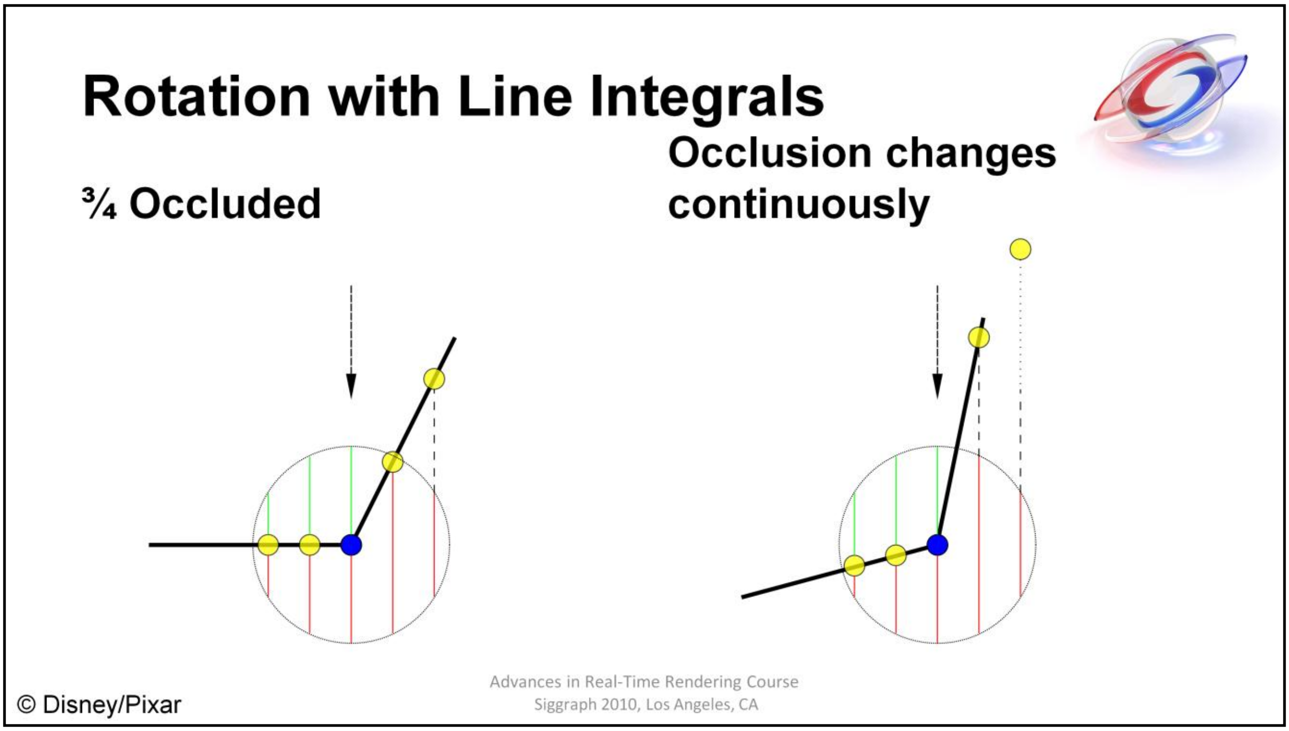
¾ Occluded

No change in occlusion

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The classic method treats every sample as either fully occluded or fully unoccluded, leading to discontinuities (like when the viewpoint rotates, for example).



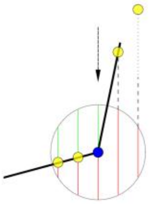
Rotation with Line Integrals

¾ Occluded

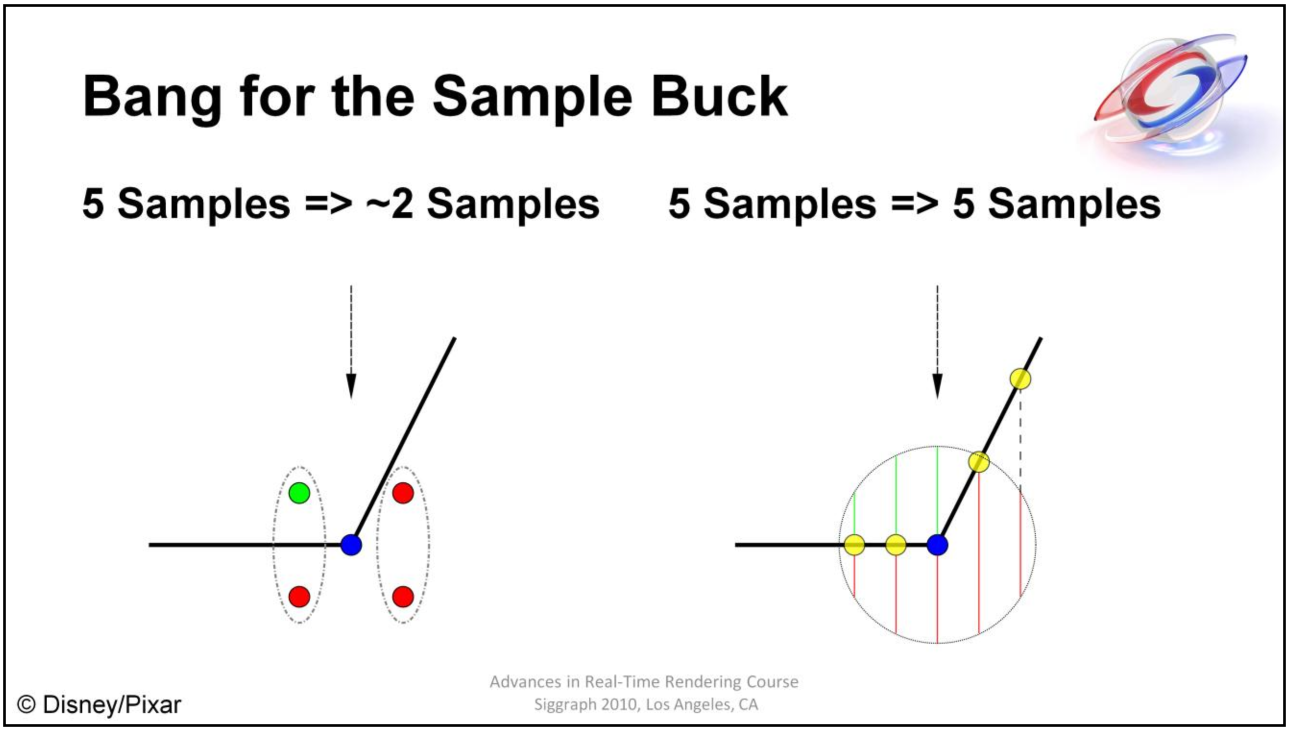
Occlusion changes continuously

With line integrals, every sample results in a fractional amount of occlusion vs. unocclusion, so the amount of occlusion changes smoothly.

SSAO Challenges



- How/Where to sample
  - Line integral advantages
    - Continuous rather than discrete
    - 2D rather than 3D
- How to fake more samples
- How to deal with large depth differences



Bang for the Sample Buck

5 Samples => ~2 Samples

5 Samples => 5 Samples

© Disney/Pixar

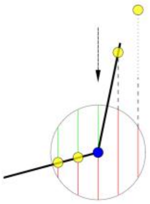
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

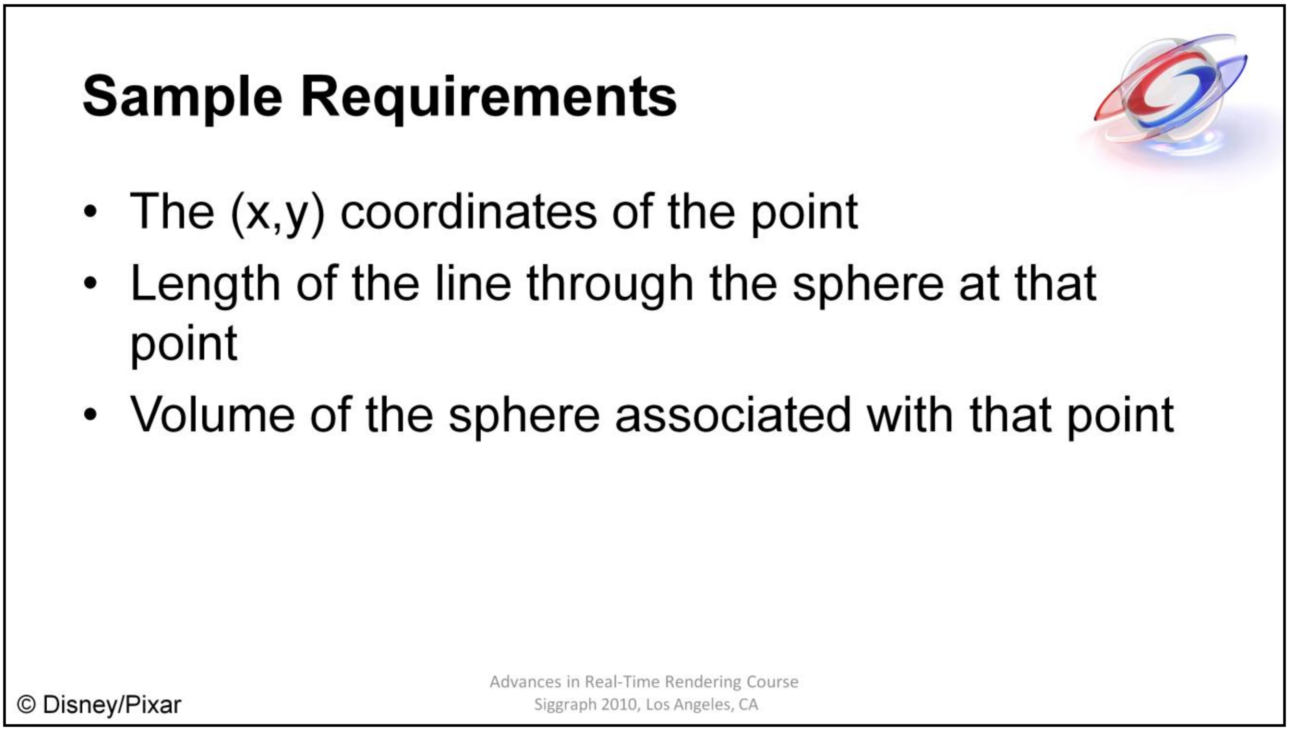
Because the samples in the classic method are chosen in 3D but sampled from a 2D texture, projection can cause the samples to not be well-distributed, which isn’t a problem when using line integrals.

Also, the original sample contributes to the occlusion in our method.

SSAO Challenges

- How/Where to sample
  - How to calculate line integrals
- How to fake more samples
- How to deal with large depth differences





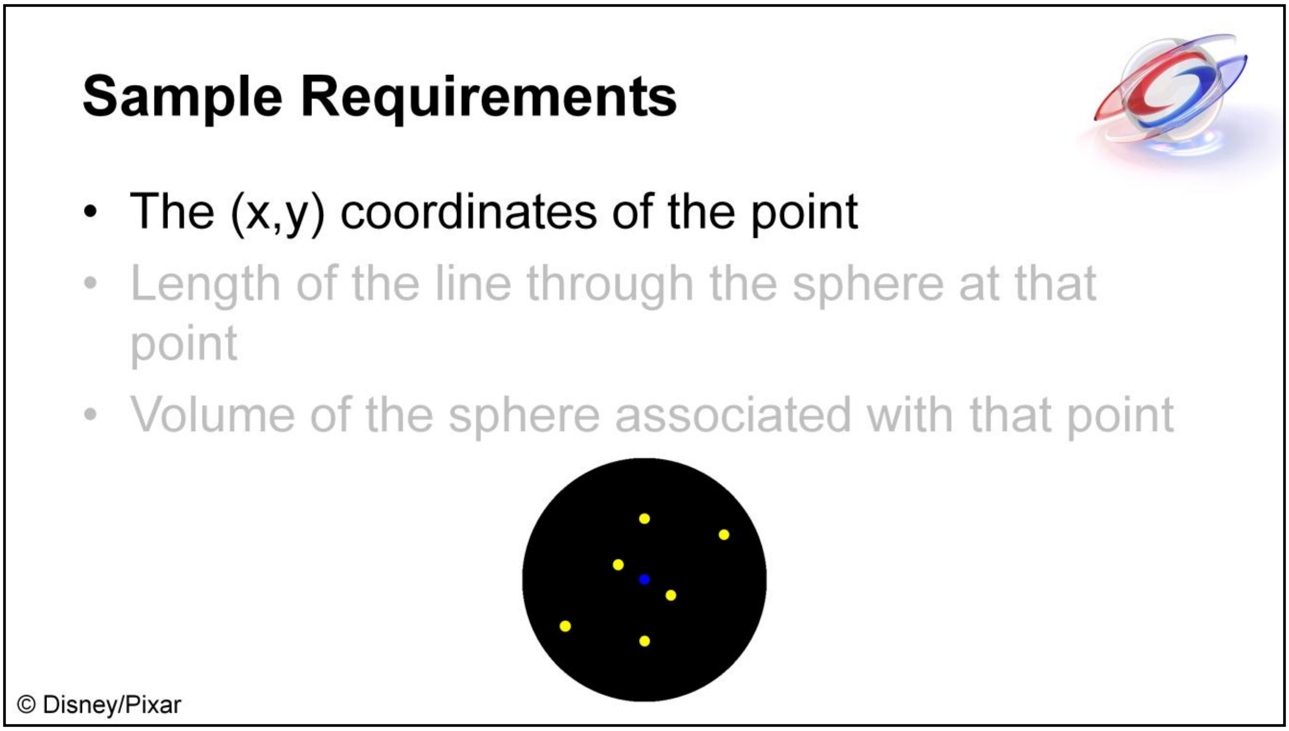
Sample Requirements

- The (x,y) coordinates of the point
- Length of the line through the sphere at that point
- Volume of the sphere associated with that point

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Three things are needed when calculating SSAO with line integrals:

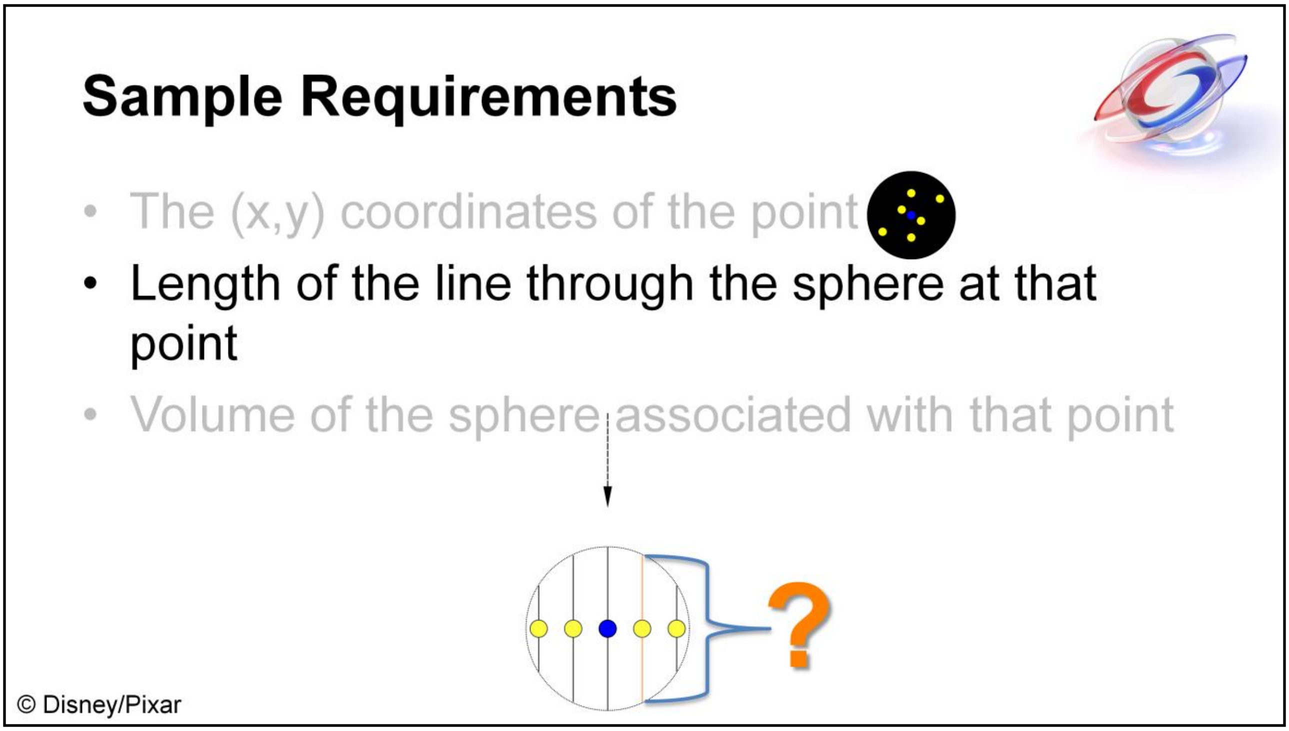


Sample Requirements

- The (x,y) coordinates of the point
- Length of the line through the sphere at that point
- Volume of the sphere associated with that point

© Disney/Pixar

You need to know the (x,y) coordinates of the sample point, relative to the unit sphere.

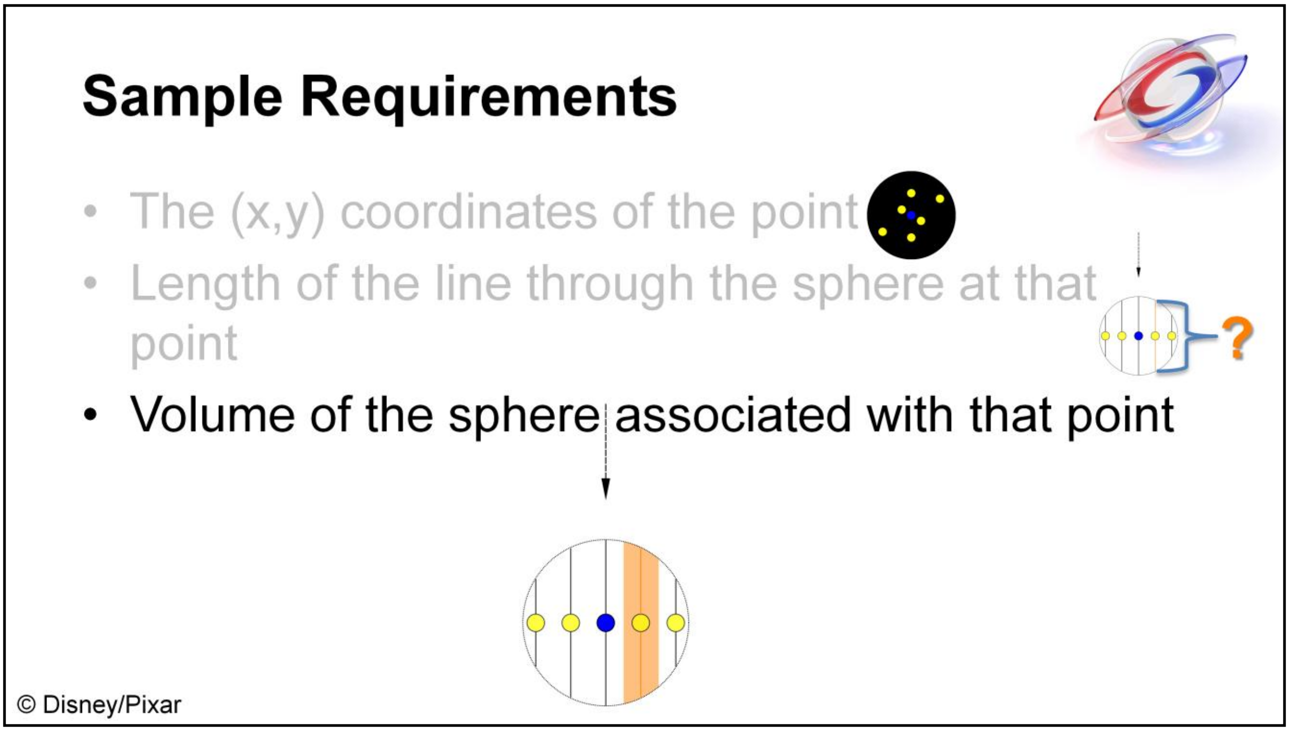


Sample Requirements

- The (x,y) coordinates of the point
- Length of the line through the sphere at that point
- Volume of the sphere associated with that point

© Disney/Pixar

You need to know the depth of the sphere at each (x,y) sample point.

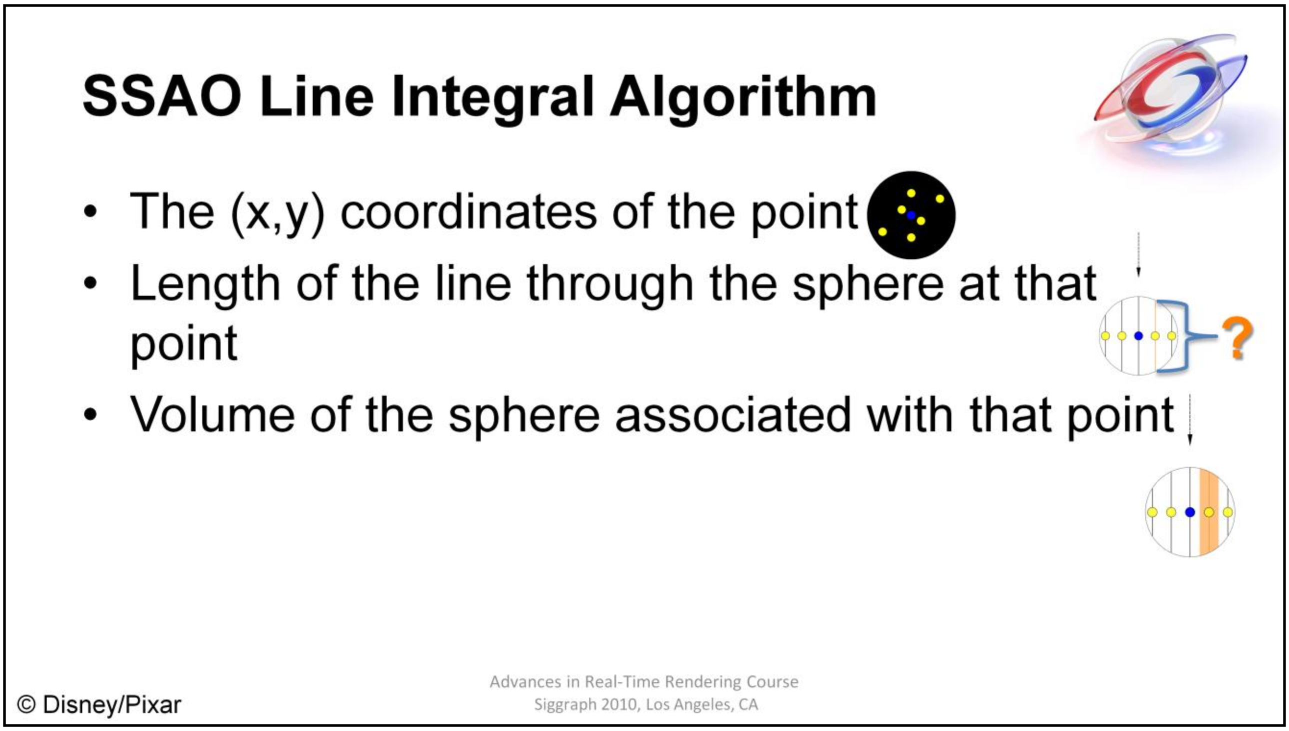


Sample Requirements

- The (x,y) coordinates of the point
- Length of the line through the sphere at that point
- Volume of the sphere associated with that point

© Disney/Pixar

You need to know the volume of the sphere associated with each sample point.



SSAO Line Integral Algorithm

- The (x,y) coordinates of the point
- Length of the line through the sphere at that point
- Volume of the sphere associated with that point

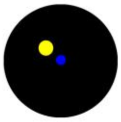
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

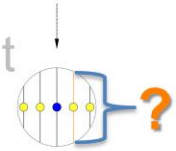
With these three pieces of information, the algorithm is straightforward:

SSAO Line Integral Algorithm

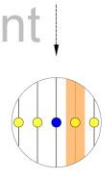
- Sample the (x,y) coordinates

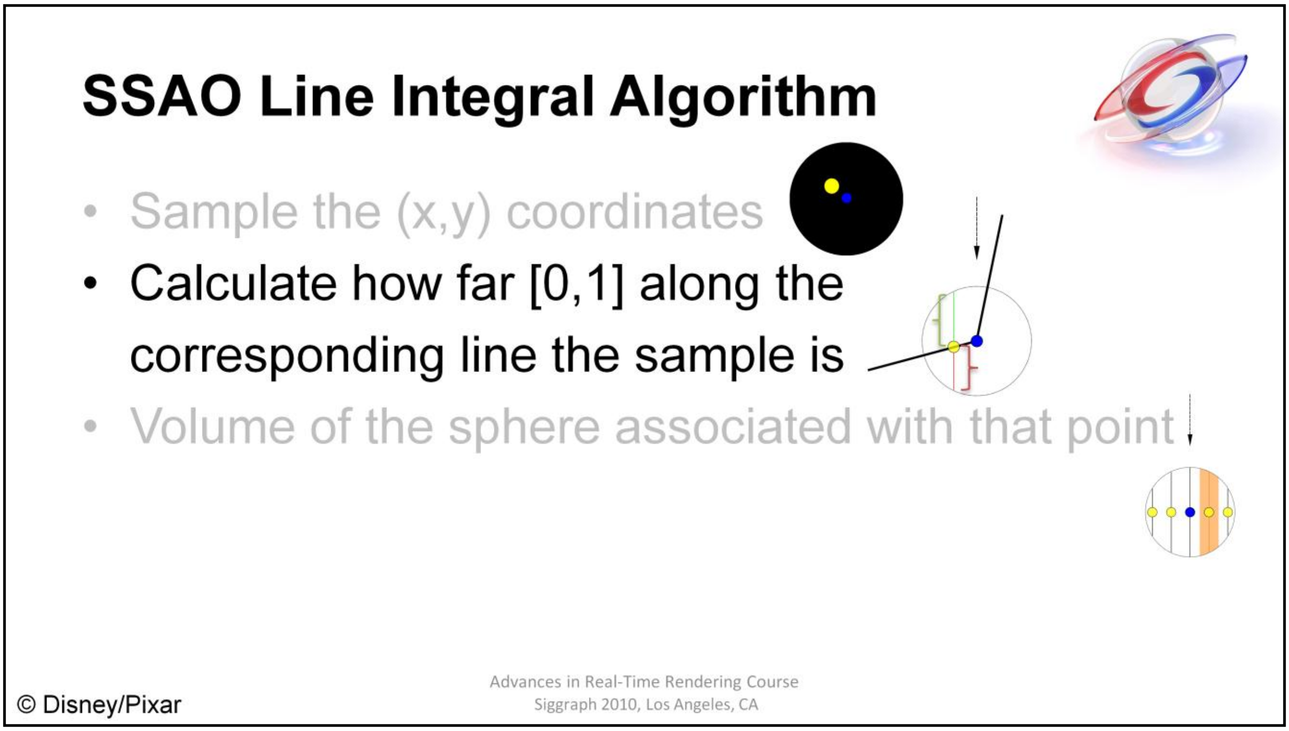


- Length of the line through the sphere at that point



- Volume of the sphere associated with that point





SSAO Line Integral Algorithm

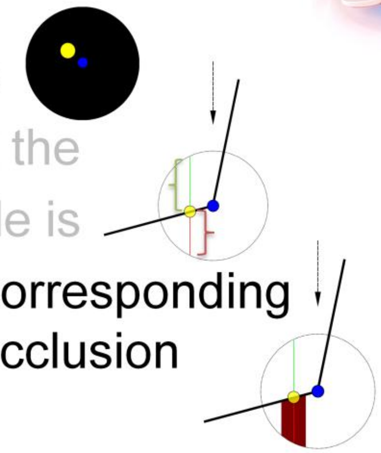
- Sample the (x,y) coordinates
- Calculate how far [0,1] along the corresponding line the sample is
- Volume of the sphere associated with that point

© Disney/Pixar

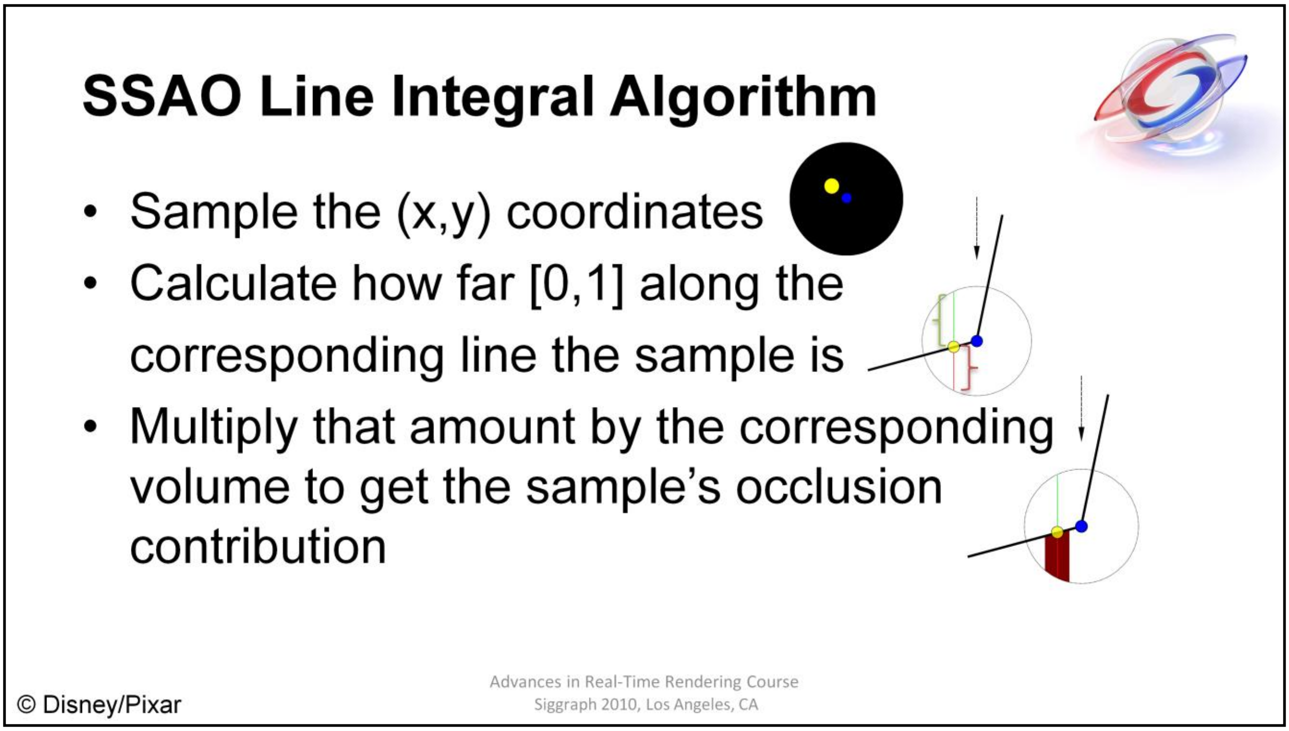
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

SSAO Line Integral Algorithm

- Sample the (x,y) coordinates
- Calculate how far [0,1] along the corresponding line the sample is
- Multiply that amount by the corresponding volume to get the sample's occlusion contribution



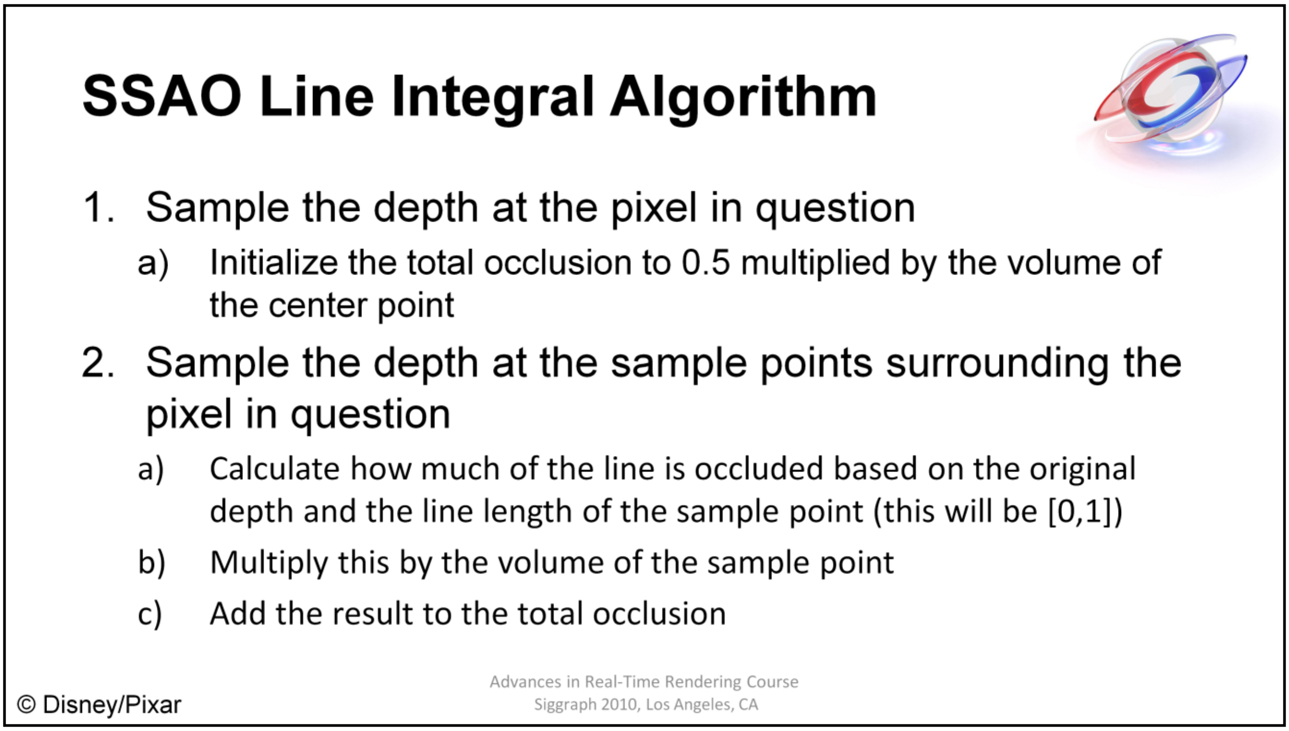
© Disney/Pixar



SSAO Line Integral Algorithm

- Sample the (x,y) coordinates
- Calculate how far [0,1] along the corresponding line the sample is
- Multiply that amount by the corresponding volume to get the sample's occlusion contribution

Do this for each sample to find the total occlusion.

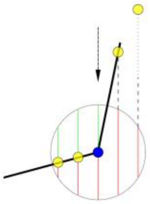


SSAO Line Integral Algorithm

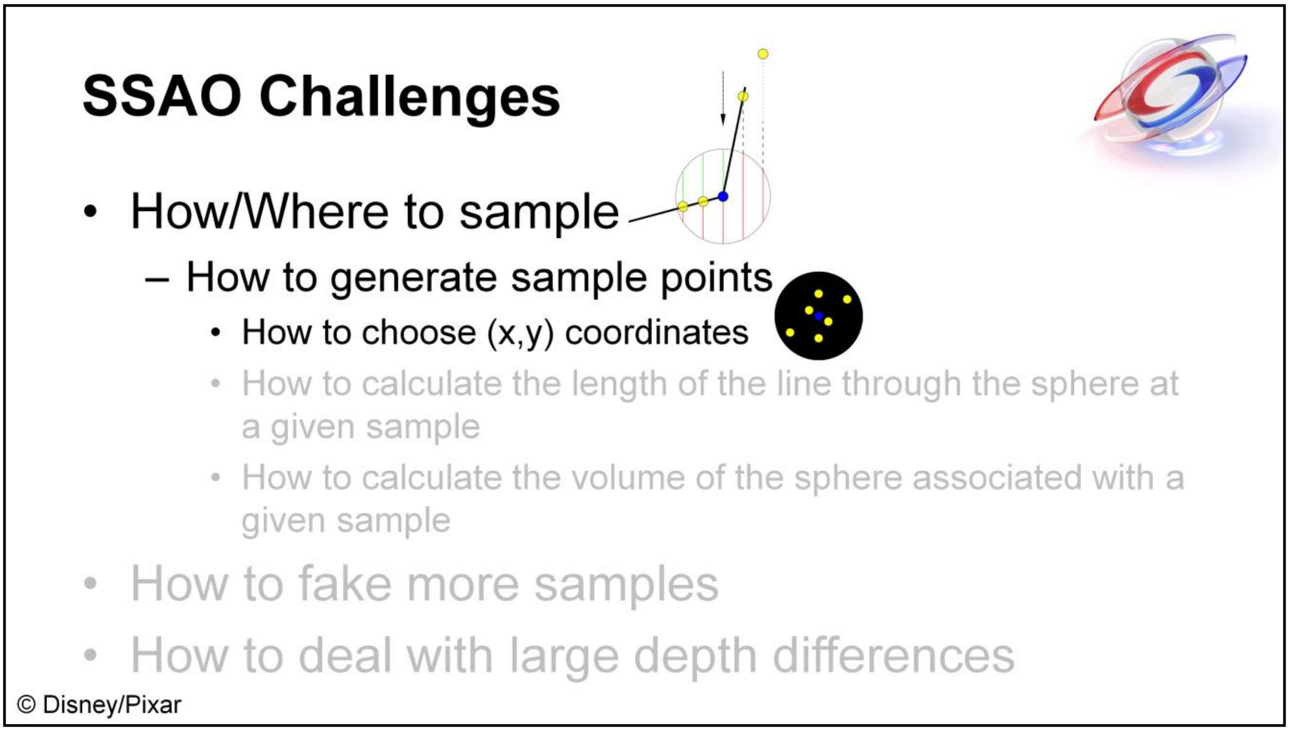
1. Sample the depth at the pixel in question
   a) Initialize the total occlusion to 0.5 multiplied by the volume of the center point
2. Sample the depth at the sample points surrounding the pixel in question
   a) Calculate how much of the line is occluded based on the original depth and the line length of the sample point (this will be [0,1])
   b) Multiply this by the volume of the sample point
   c) Add the result to the total occlusion

SSAO Challenges

- How/Where to sample
  - How to generate sample points
    - How to choose (x,y) coordinates
    - How to calculate the length of the line through the sphere at a given sample
    - How to calculate the volume of the sphere associated with a given sample

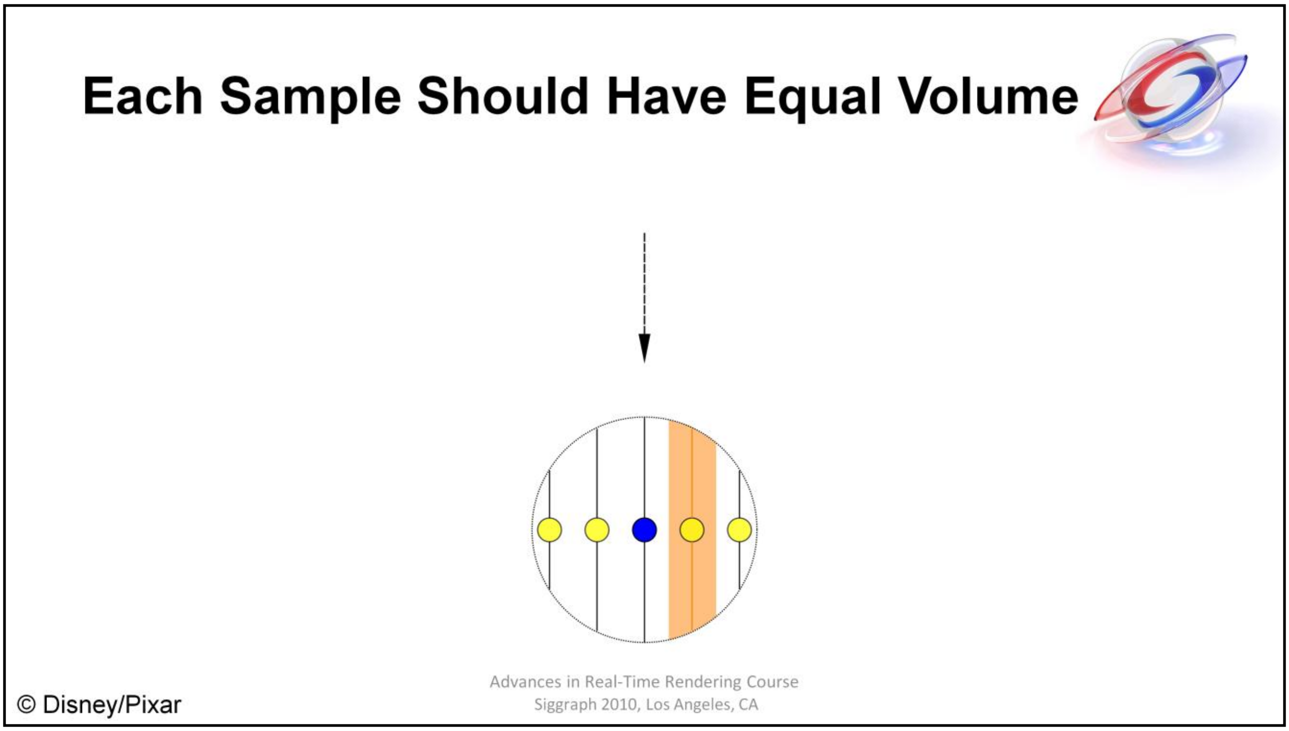


- How to fake more samples
- How to deal with large depth differences



SSAO Challenges

- How/Where to sample
  - How to generate sample points
    - How to choose (x,y) coordinates
    - How to calculate the length of the line through the sphere at a given sample
    - How to calculate the volume of the sphere associated with a given sample
- How to fake more samples
- How to deal with large depth differences

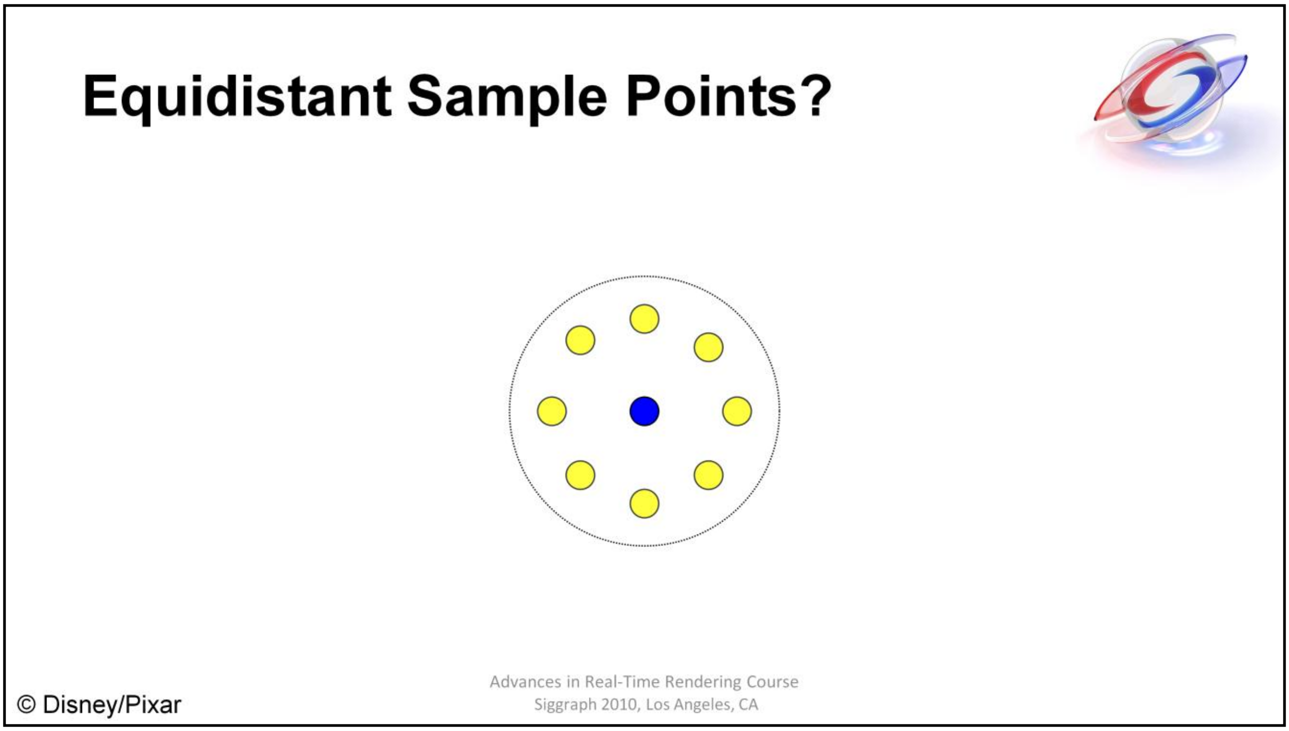


Each Sample Should Have Equal Volume

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Remember that each sample represents some volume of the unit sphere. No matter which sample pattern is chosen, the goal should be to have all of the volumes be as close to equal as possible, so that each sample has equal importance.

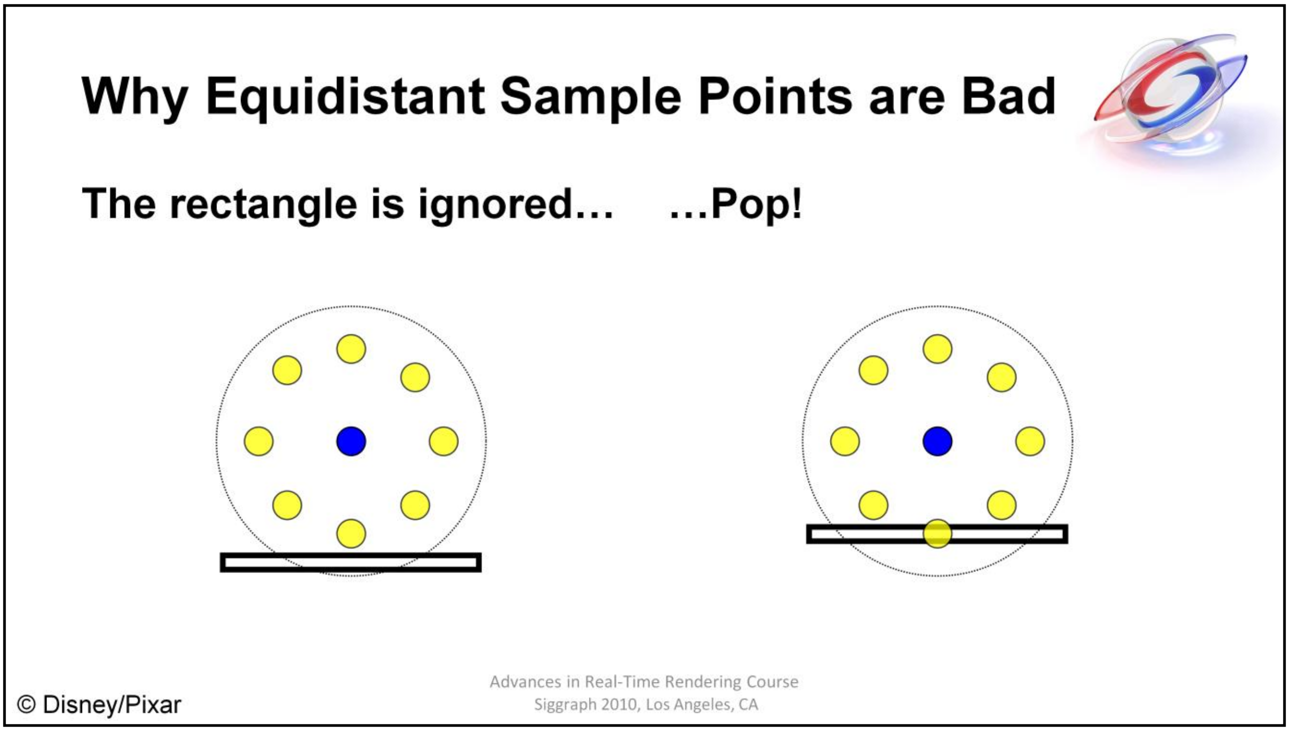


Equidistant Sample Points?

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The most obvious pattern is to have all of the points equidistant from the center (with the radius chosen to maintain equal volumes).



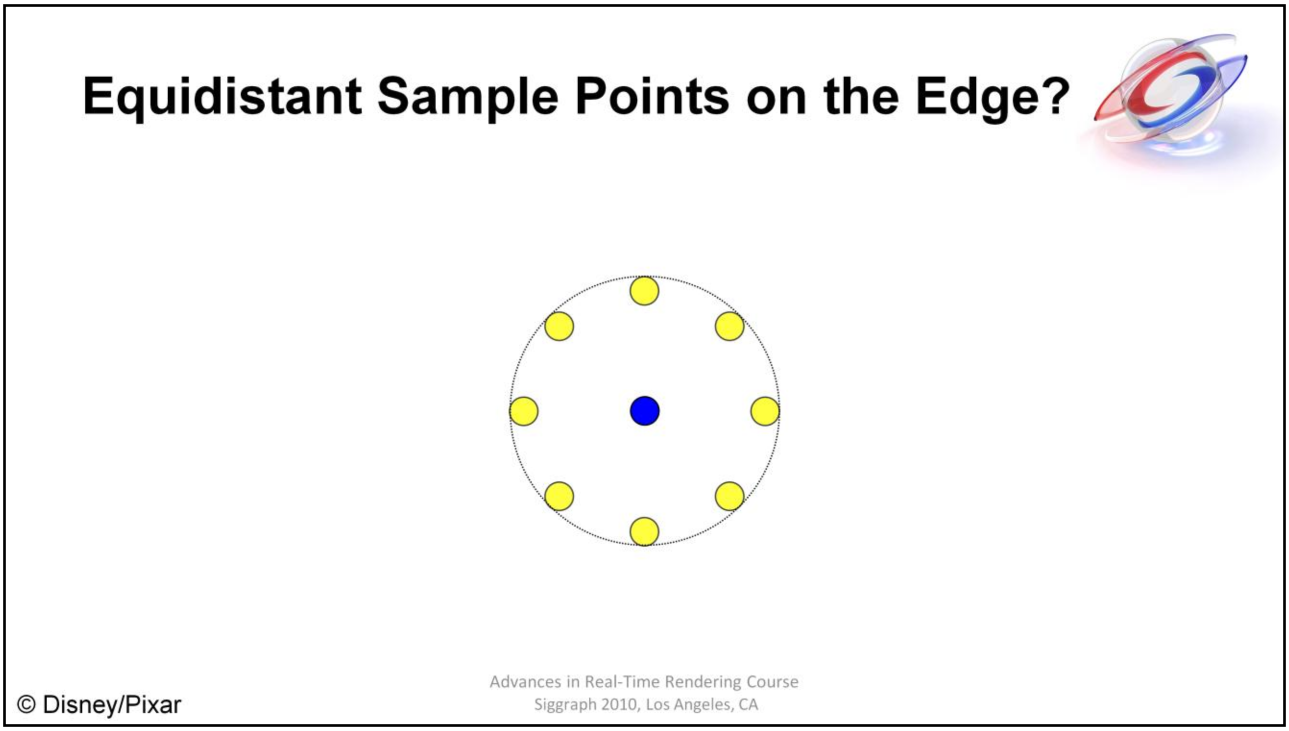
Why Equidistant Sample Points are Bad

The rectangle is ignored… …Pop!

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Occluders will be ignored until they are within the sample point radius, even if they are within the conceptual sphere that the line integrals apply to. This results in objectionable popping once occluders *are* inside the sample point radius.

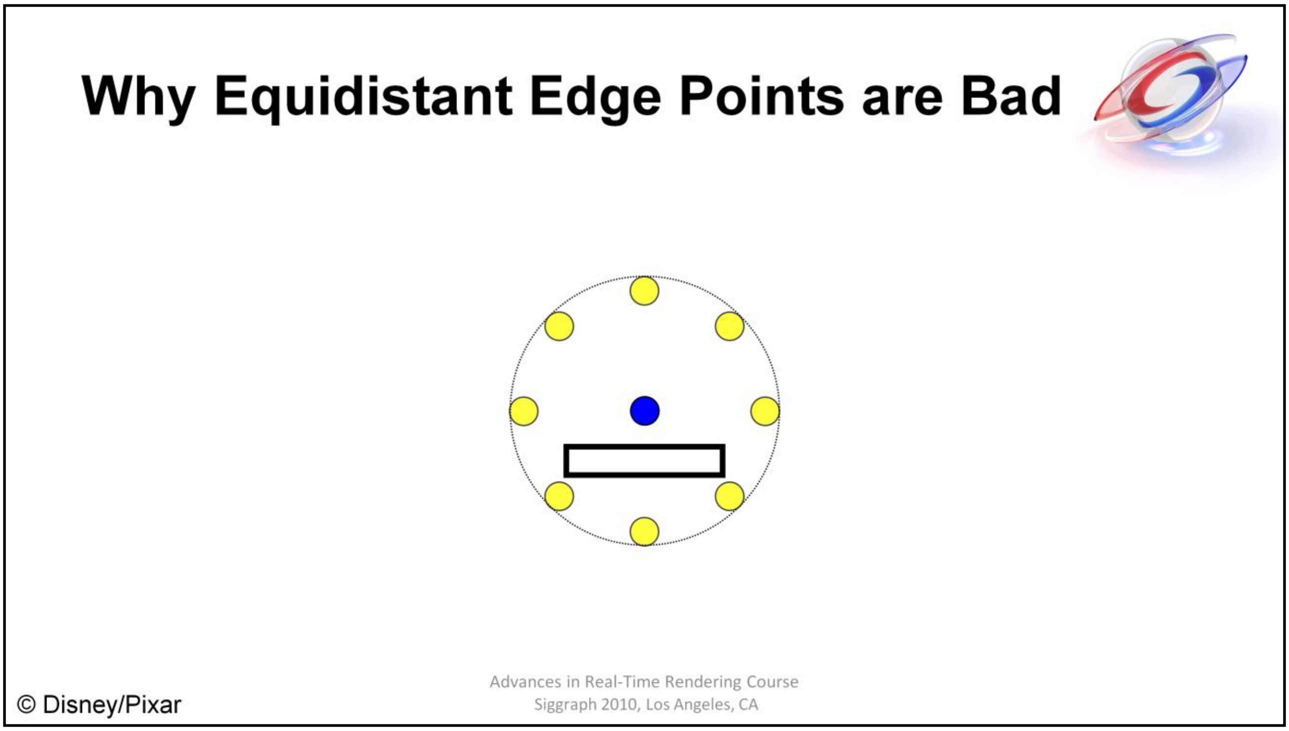


Equidistant Sample Points on the Edge?

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

One potential solution is to place the sample points on the sphere itself, so that there will never be popping.



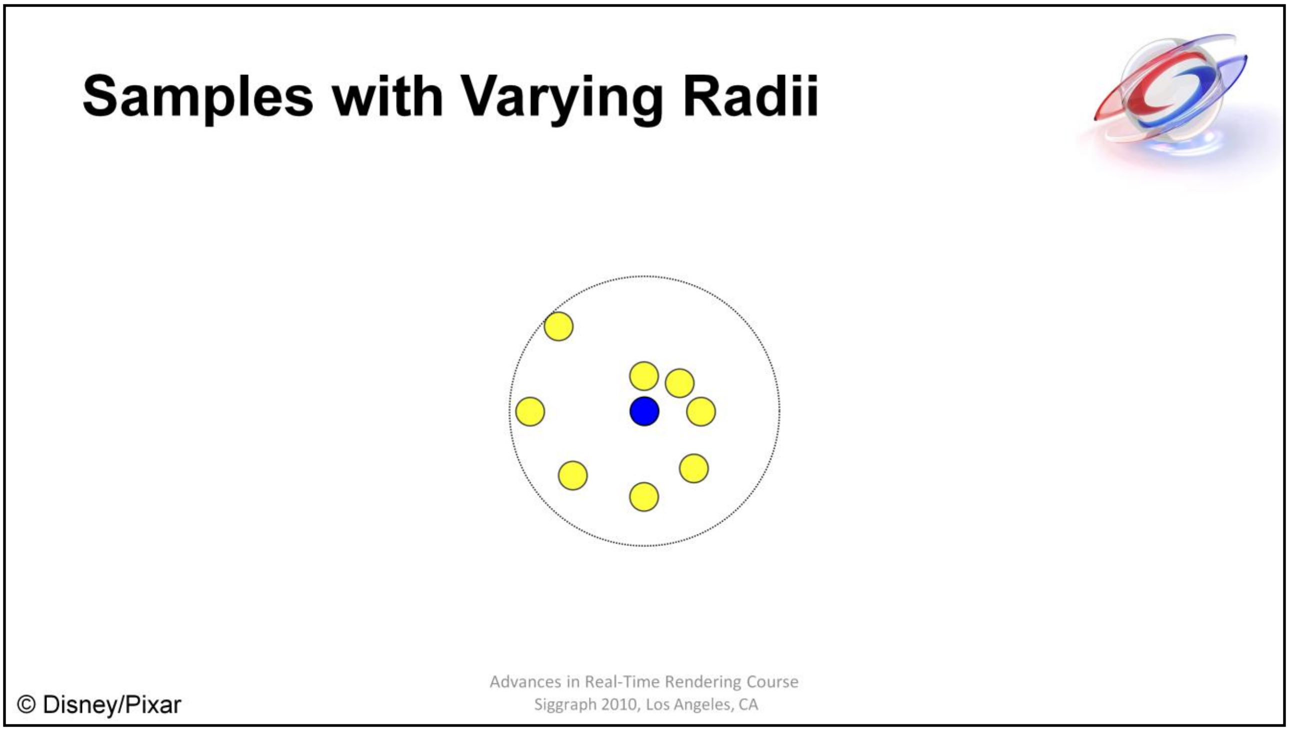
Why Equidistant Edge Points are Bad

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The problem is with this is that occluders smaller than the sphere’s radius will be missed.

(An additional problem is that there is less accuracy at the edges, since the line lengths are close to zero.)

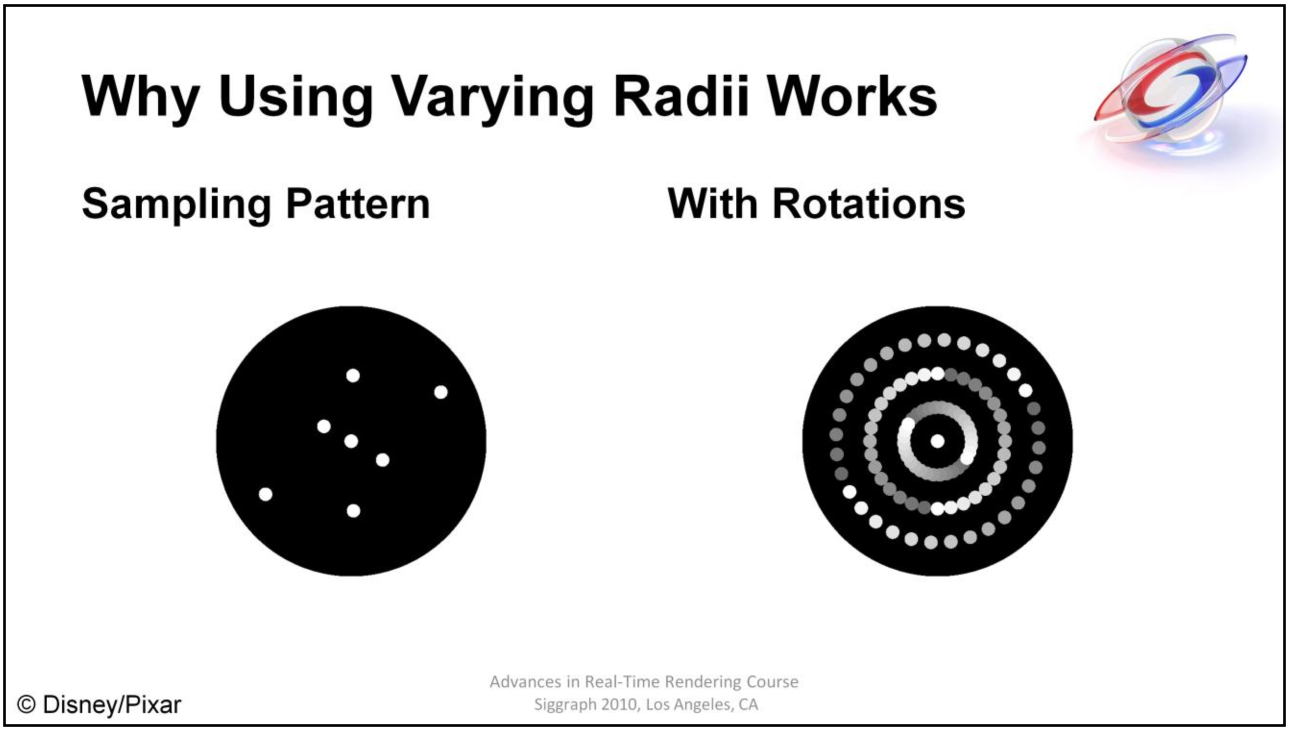


Samples with Varying Radii

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

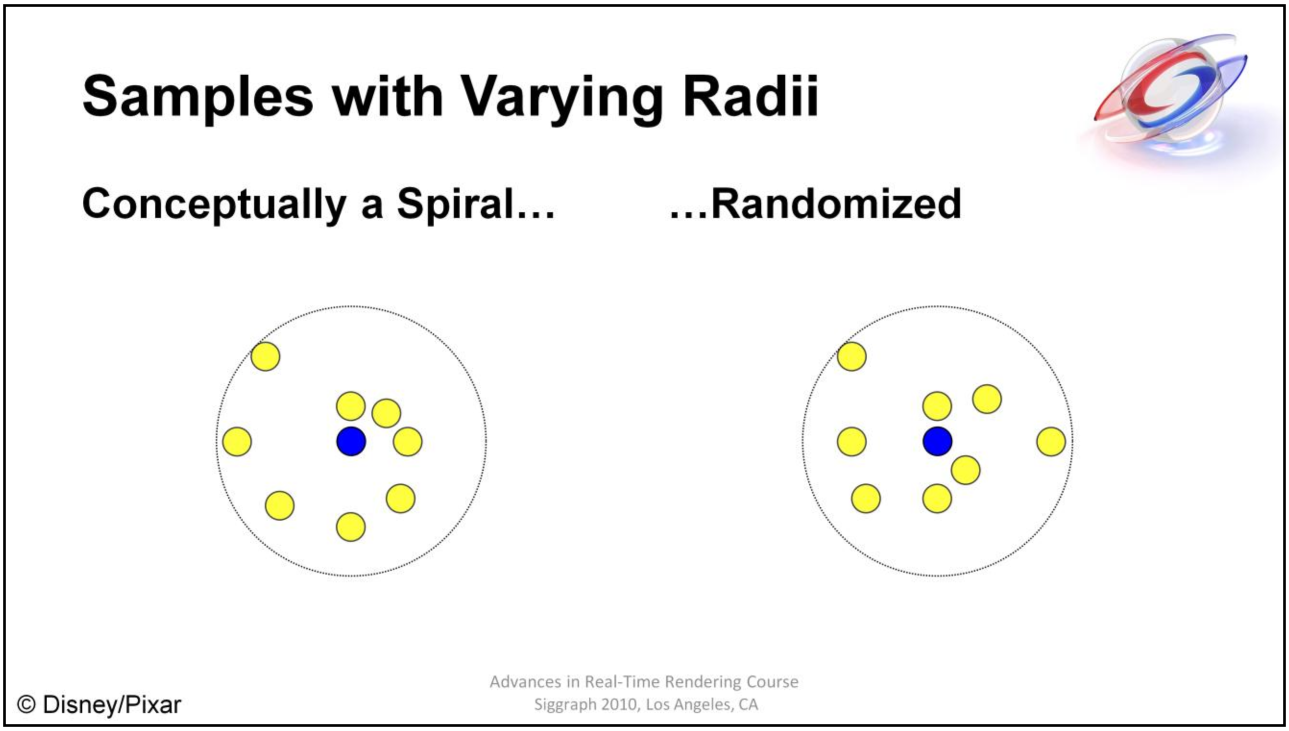
The best solution is to use varying radii so there is good coverage everywhere.



Why Using Varying Radii Works

Sampling Pattern **With Rotations**

Remember that we are going to rotate the sampling pattern to try and fake more samples.



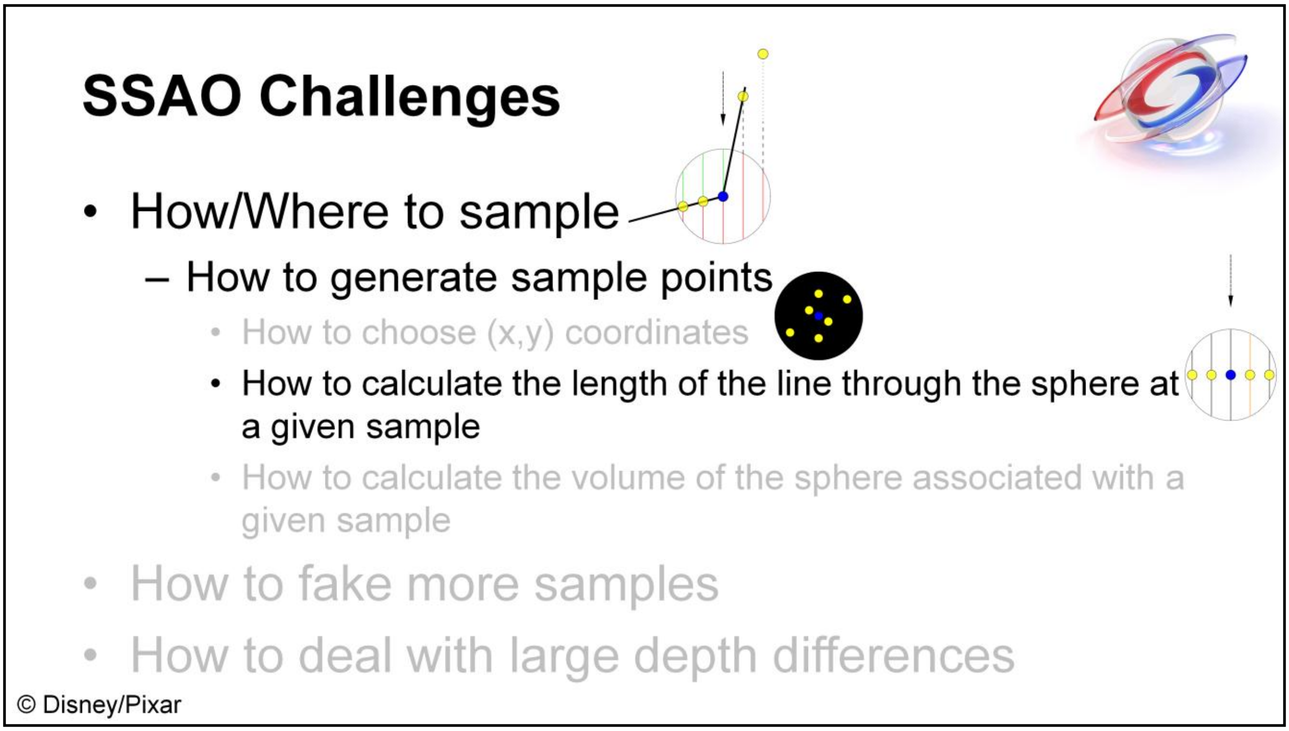
Samples with Varying Radii

Conceptually a Spiral… …Randomized

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

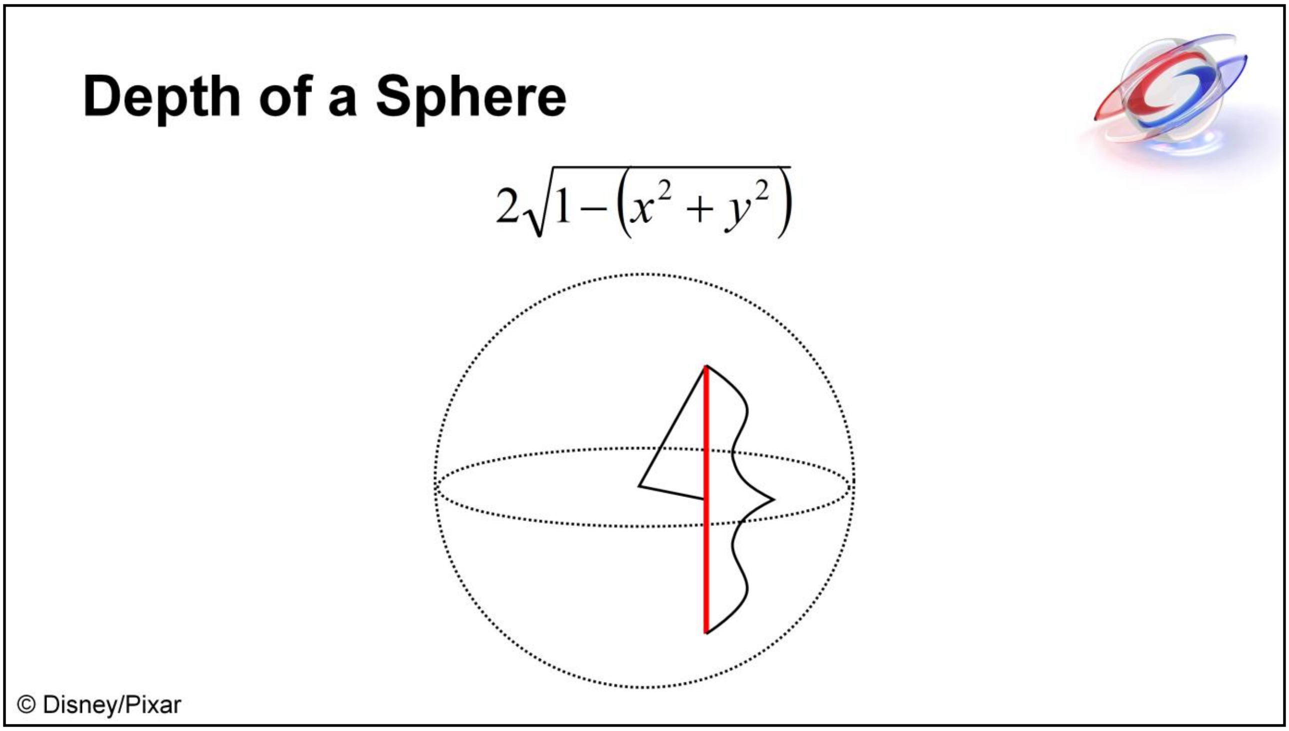
You can start with equally-spaced radii, but there should be some iterations to try and get equal volumes. The different radii should also be randomized to avoid predictable patterns. (I say “randomized”, but we actually chose ours to try and get the most equal volumes we could.)



SSAO Challenges

- How/Where to sample
  - How to generate sample points
    - How to choose (x,y) coordinates
    - How to calculate the length of the line through the sphere at a given sample
    - How to calculate the volume of the sphere associated with a given sample
- How to fake more samples
- How to deal with large depth differences

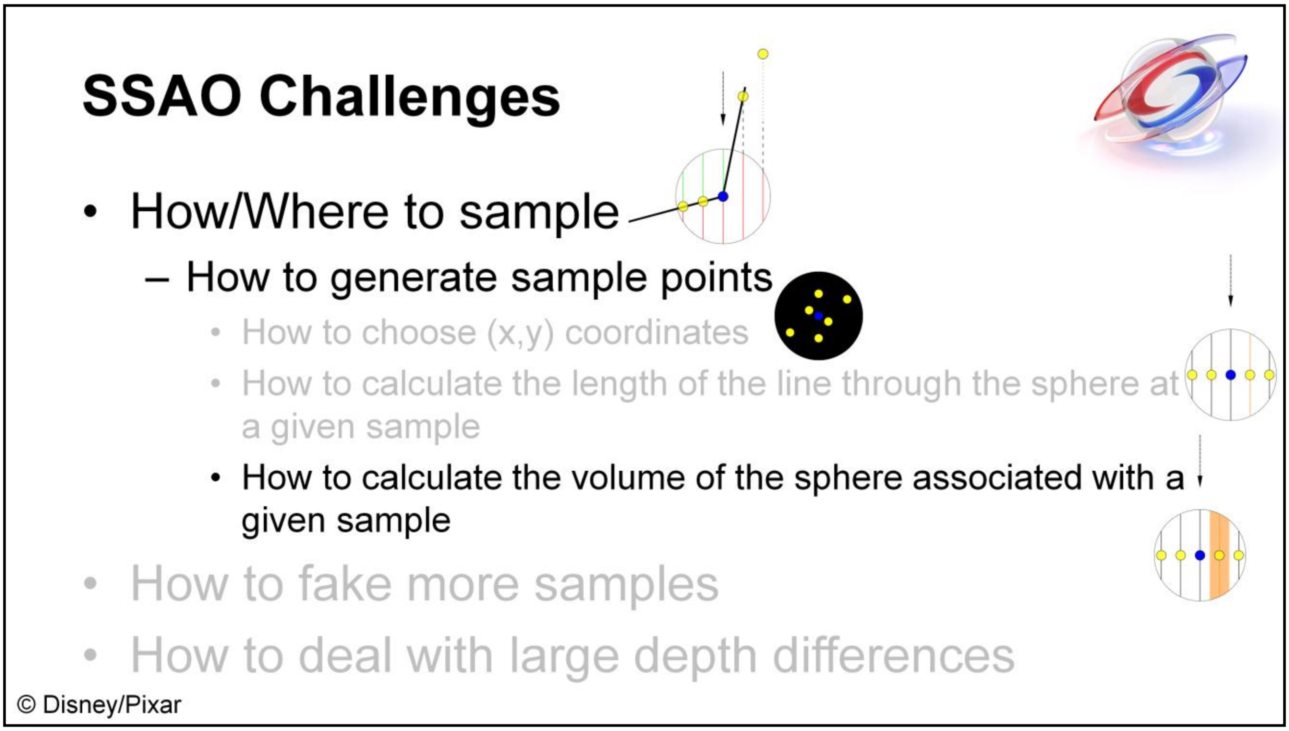
© Disney/Pixar



Depth of a Sphere

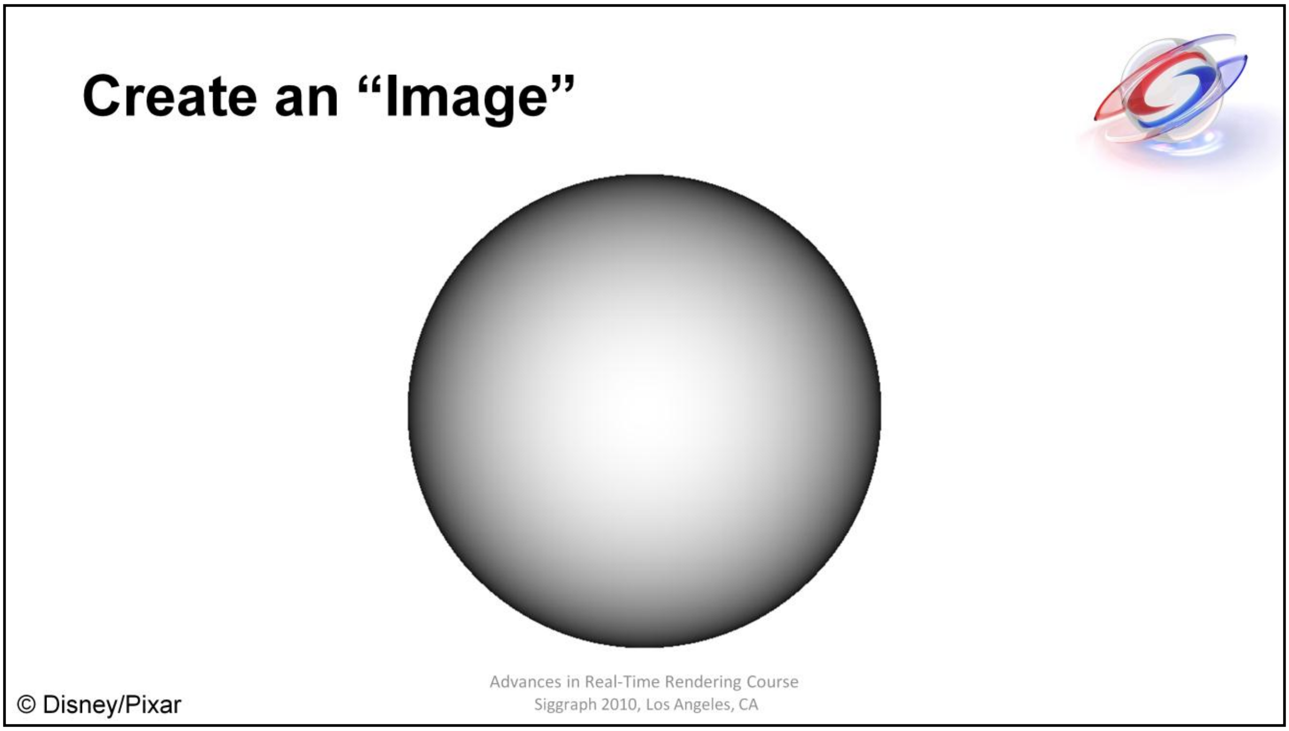
$$2\sqrt{1-\left(x^{2}+y^{2}\right)}$$

Luckily, given an (x,y) coordinate, this is easy ☺



SSAO Challenges

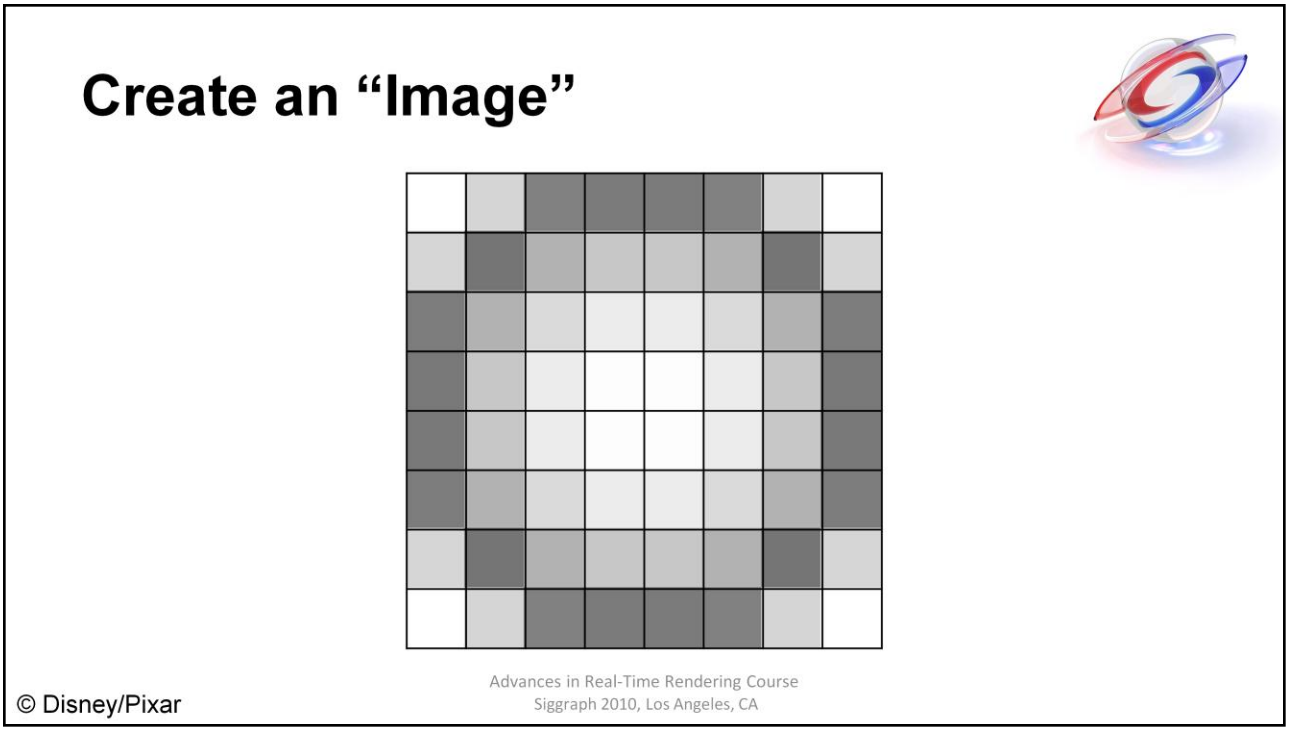
- How/Where to sample
  - How to generate sample points
    - How to choose (x,y) coordinates
    - How to calculate the length of the line through the sphere at a given sample
    - How to calculate the volume of the sphere associated with a given sample
- How to fake more samples
- How to deal with large depth differences



Create an "Image"

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Rather than calculating this analytically, there is a pretty simple "brute force" method. Conceptually, create a 2D image of the circle. The higher resolution, the more accurate the final results will be.

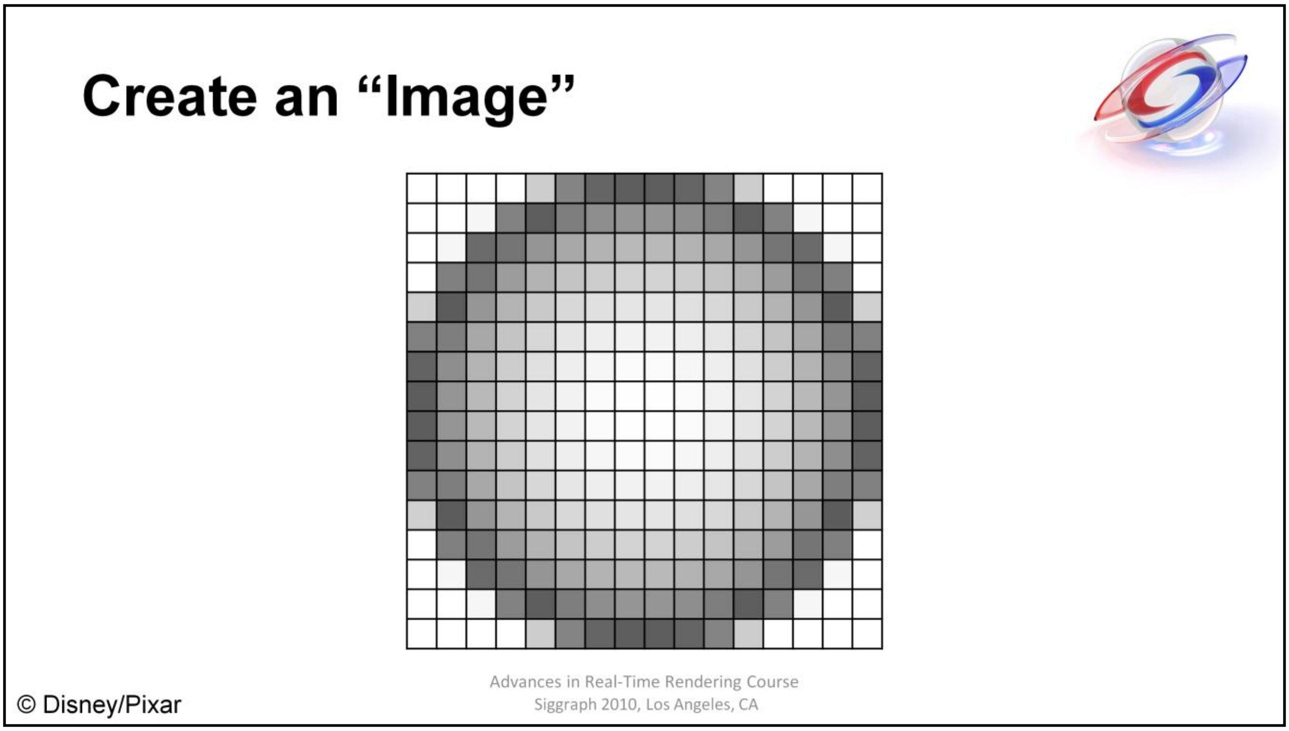


Create an "Image"

© Disney/Pixar

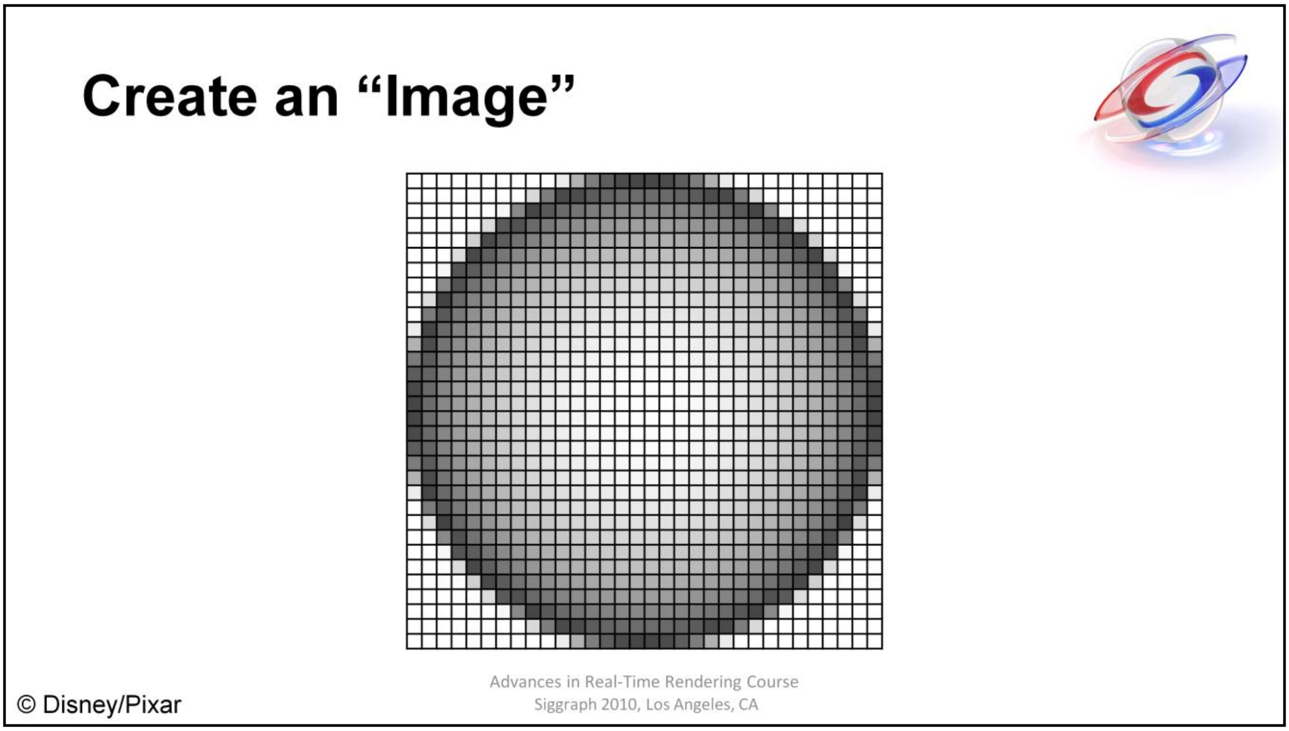
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Rather than calculating this analytically, there is a pretty simple "brute force" method. Conceptually, create a 2D image of the circle. The higher resolution, the more accurate the final results will be.



Create an "Image"

Rather than calculating this analytically, there is a pretty simple "brute force" method. Conceptually, create a 2D image of the circle. The higher resolution, the more accurate the final results will be.



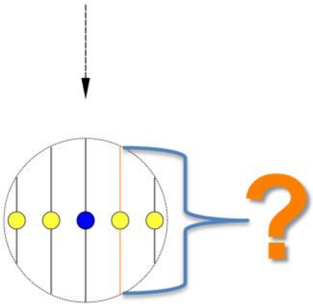
Create an “Image”

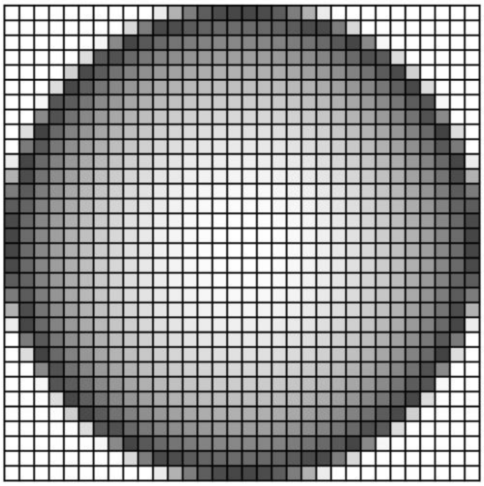
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

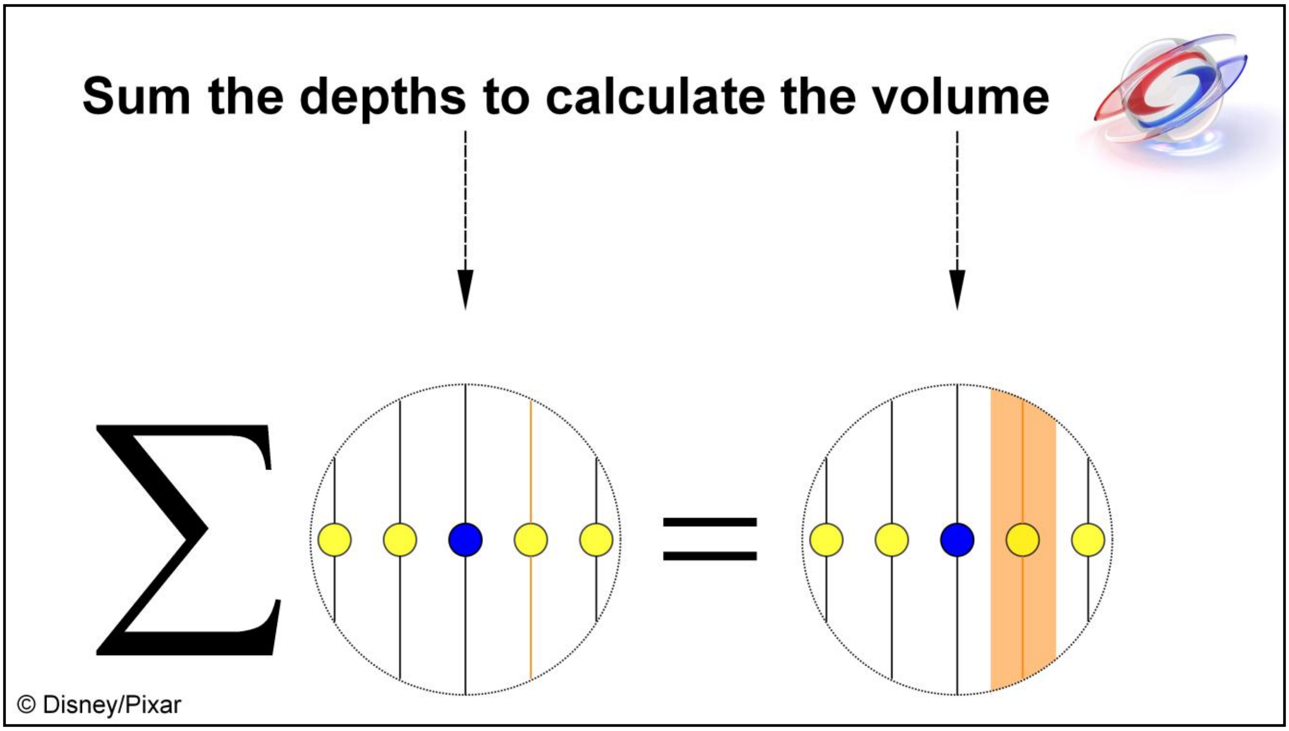
Rather than calculating this analytically, there is a pretty simple “brute force” method. Conceptually, create a 2D image of the circle. The higher resolution, the more accurate the final results will be.

Find the depth of the sphere at each pixel



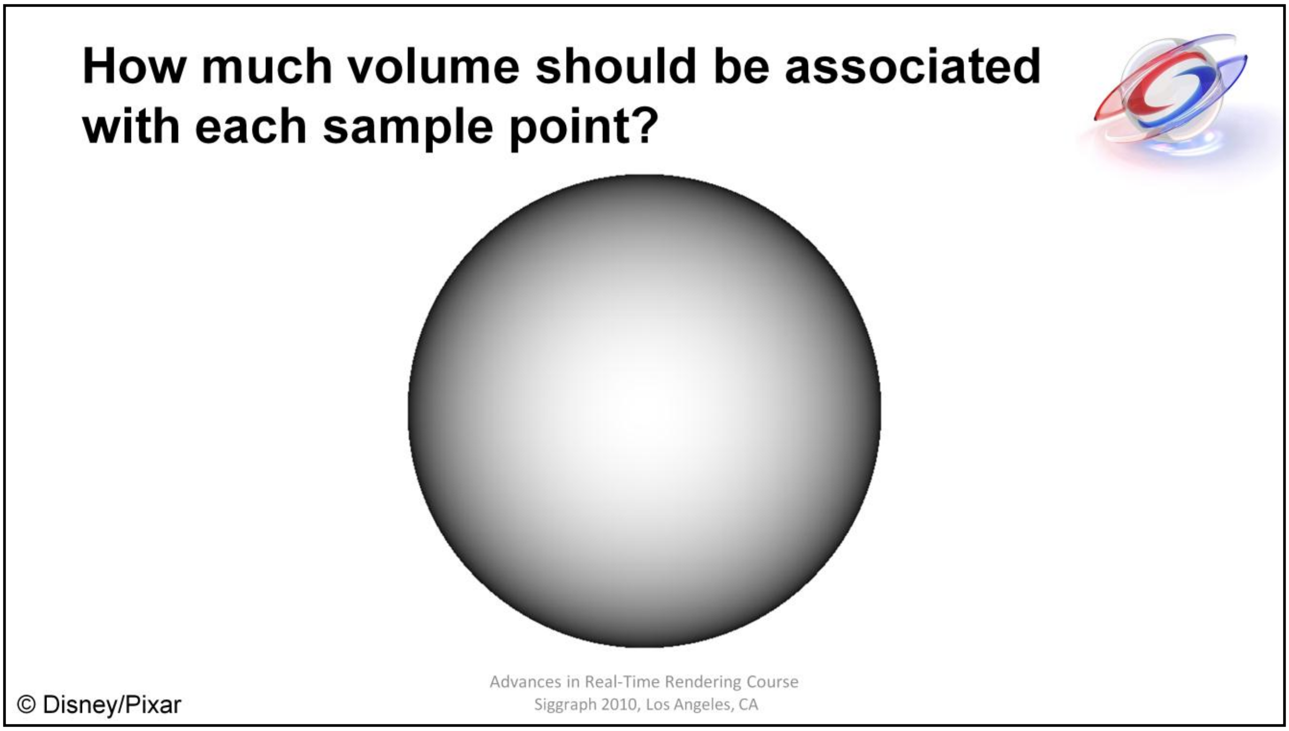


$$2\sqrt{1-\left(x^2+y^2\right)}$$



Sum the depths to calculate the volume

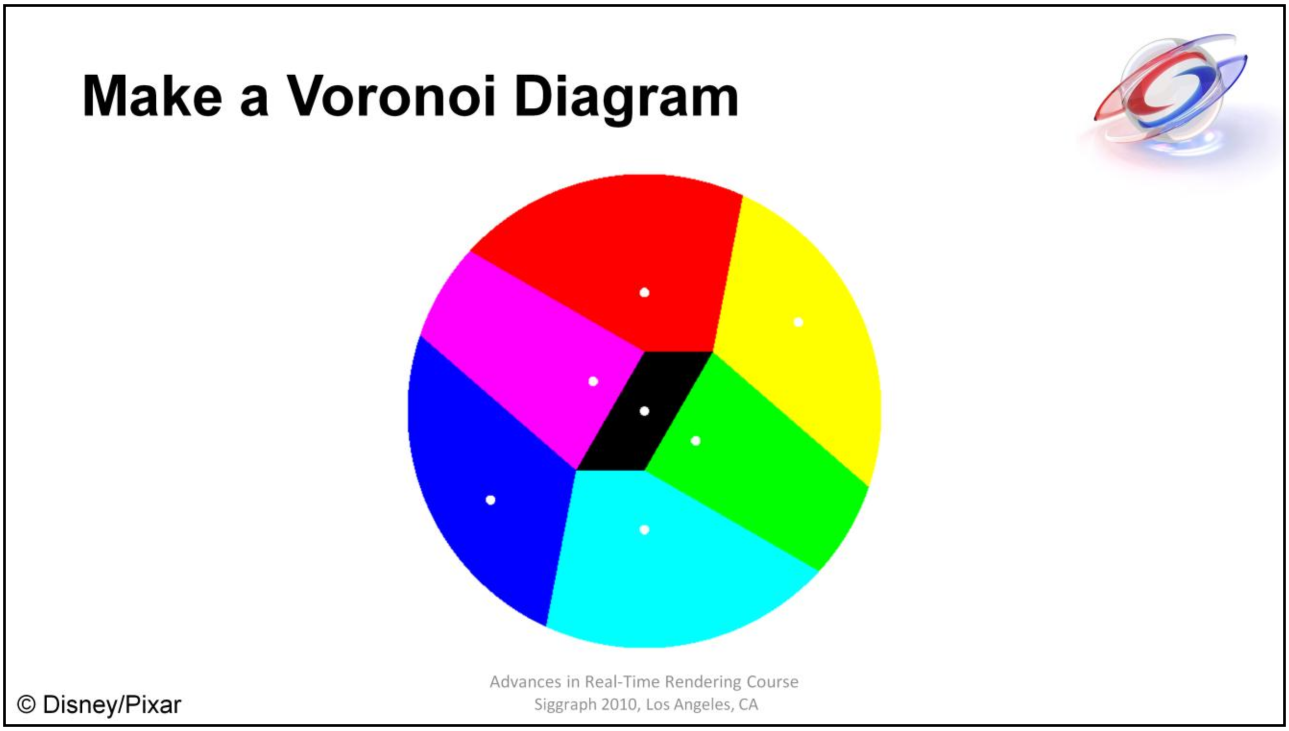
© Disney/Pixar



How much volume should be associated with each sample point?

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA



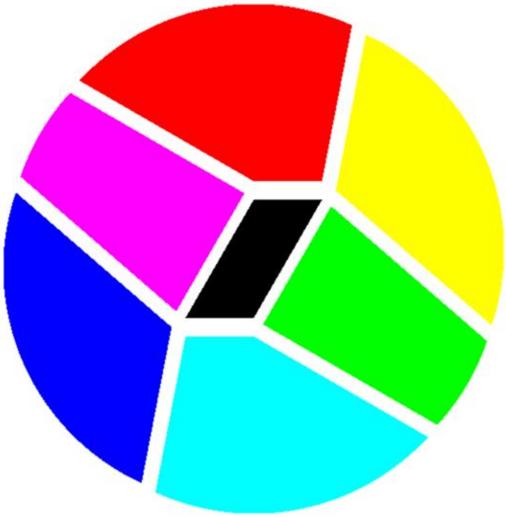
Make a Voronoi Diagram

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

For each pixel, decide which sample is the closest.

Each pixel is associated with a sample point

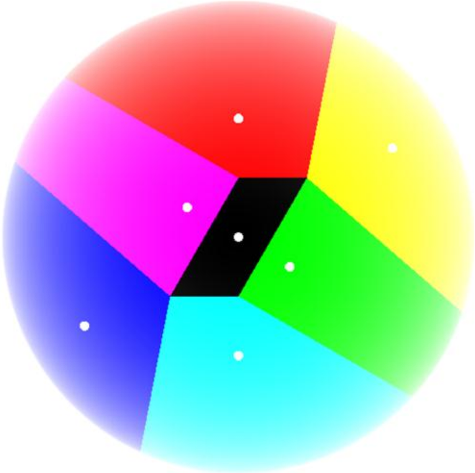


© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

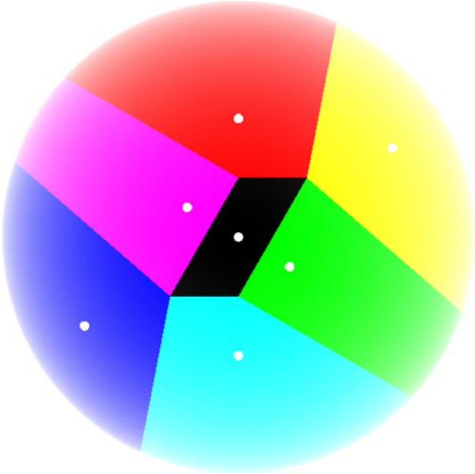
Add the depth of each pixel to the running total of the appropriate sample point

$$MagentaVolume = \sum_{MagentaPixels} 2\sqrt{1-\left(x^2+y^2\right)}$$



Add the depth of each pixel to the running total of the appropriate sample point

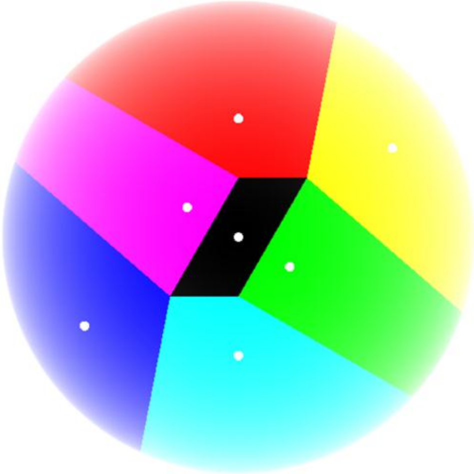
$$MagentaVolume = \sum_{MagentaPixels} 2\sqrt{1-\left(x^2+y^2\right)}$$



$$BlueVolume = \sum_{BluePixels} 2\sqrt{1-\left(x^2+y^2\right)}$$

Add the depth of each pixel to the running total of the appropriate sample point

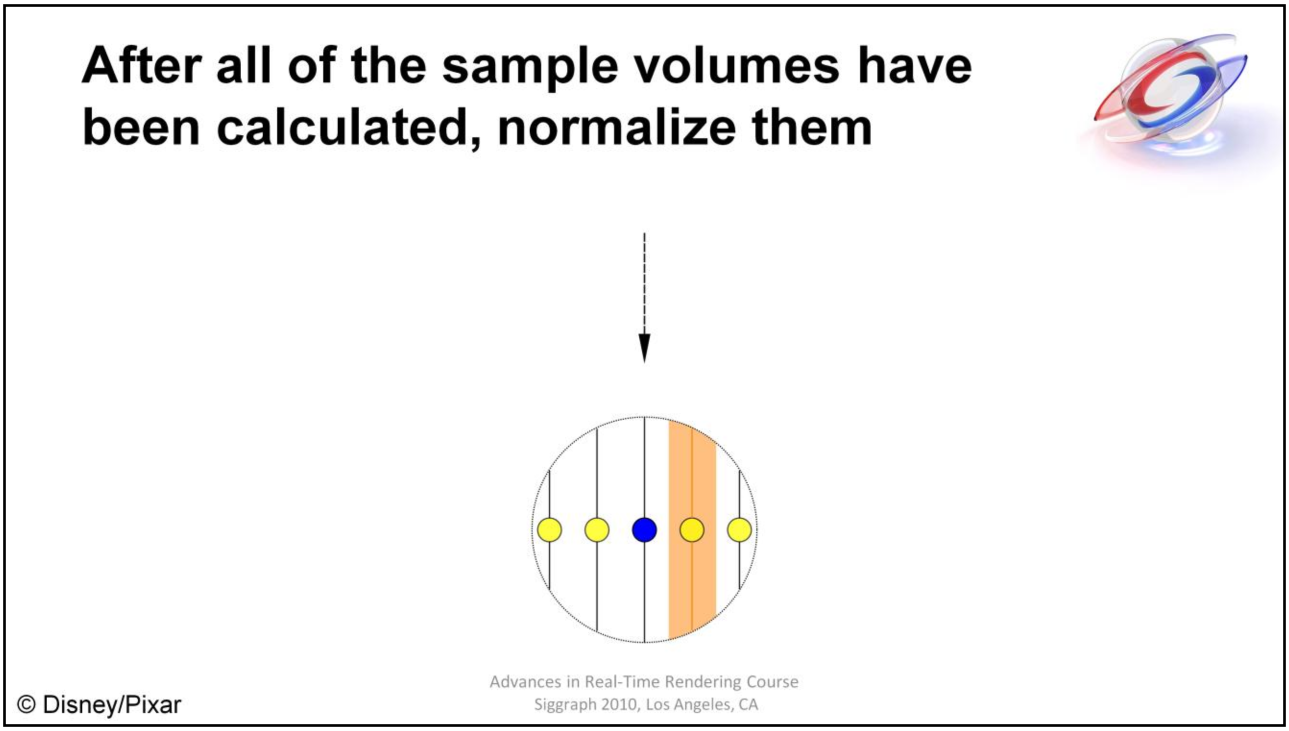
$$MagentaVolume = \sum_{MagentaPixels} 2\sqrt{1-\left(x^2+y^2\right)}$$



$$BlueVolume = \sum_{BluePixels} 2\sqrt{1-\left(x^2+y^2\right)}$$

$$CyanVolume = \sum_{CyanPixels} 2\sqrt{1-\left(x^2+y^2\right)}$$

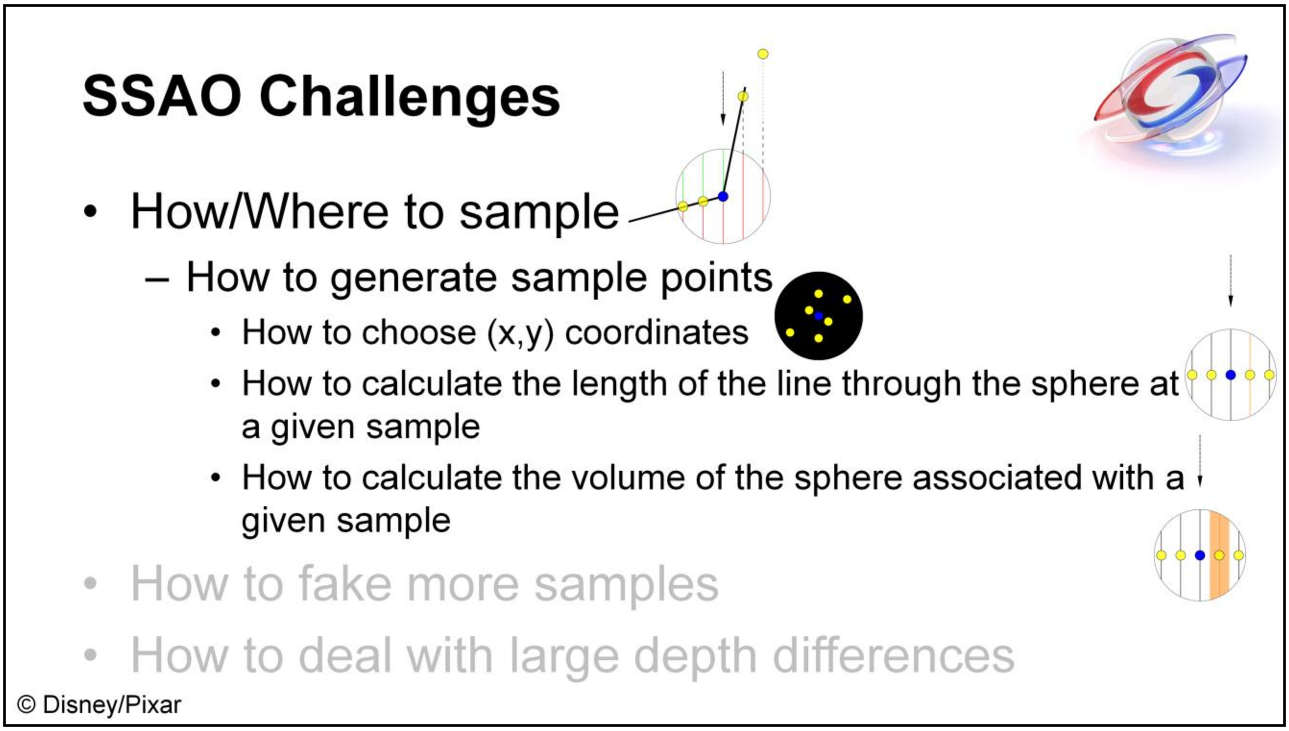
© Disney/Pixar



After all of the sample volumes have been calculated, normalize them

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

© Disney/Pixar

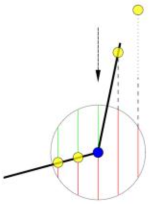


SSAO Challenges

- How/Where to sample
  - How to generate sample points
    - How to choose (x,y) coordinates
    - How to calculate the length of the line through the sphere at a given sample
    - How to calculate the volume of the sphere associated with a given sample
- How to fake more samples
- How to deal with large depth differences

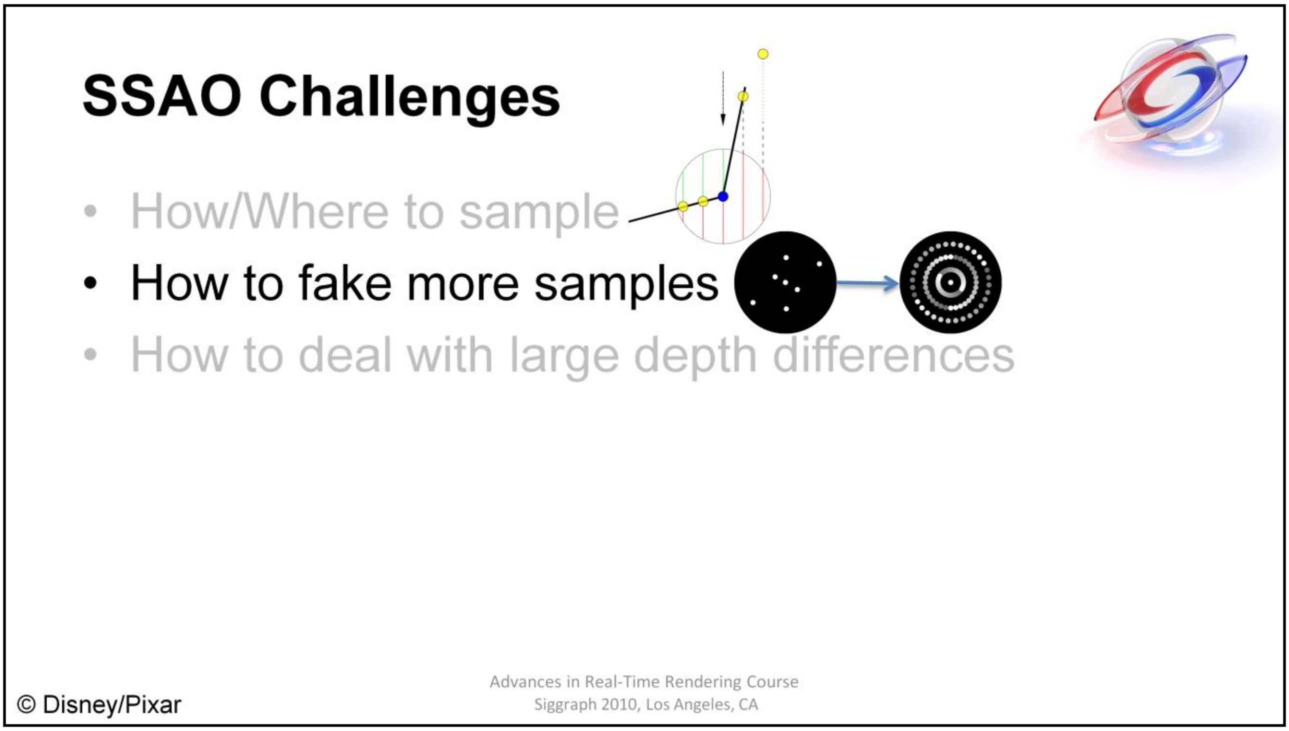
SSAO Challenges

- How/Where to sample
- How to fake more samples
- How to deal with large depth differences



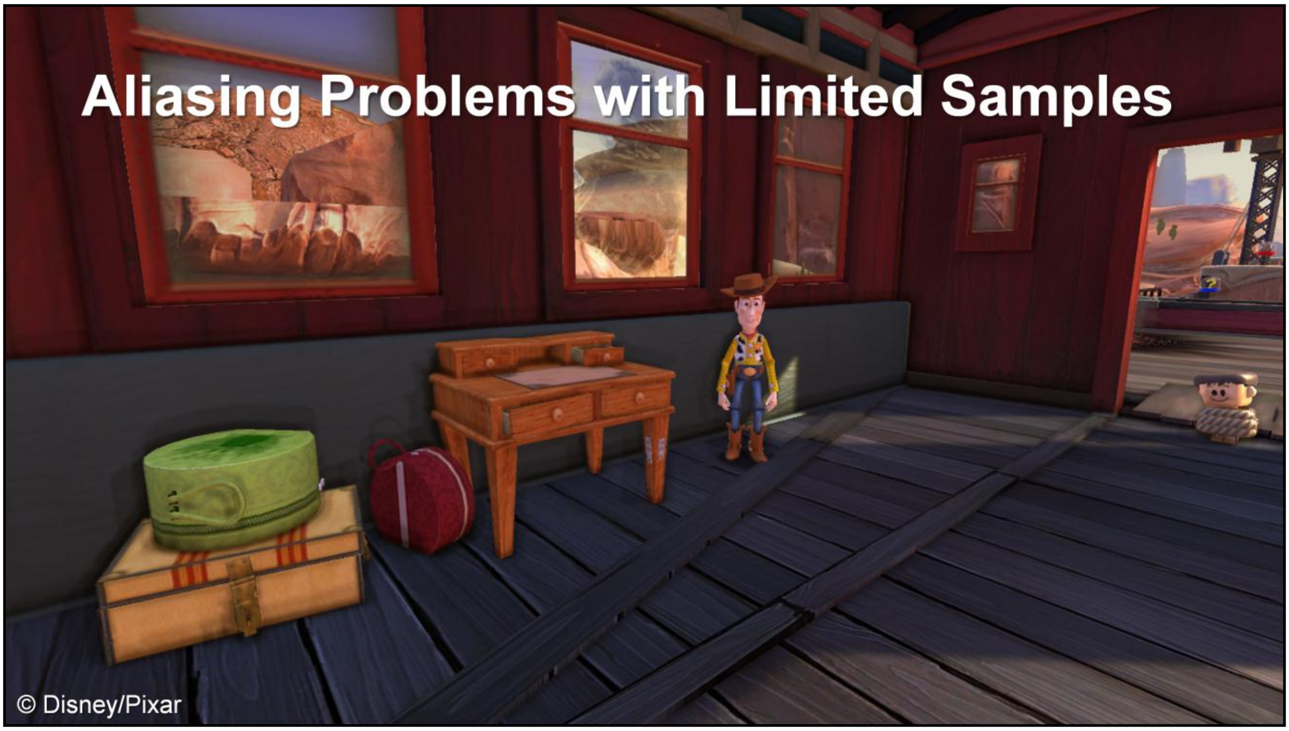
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA



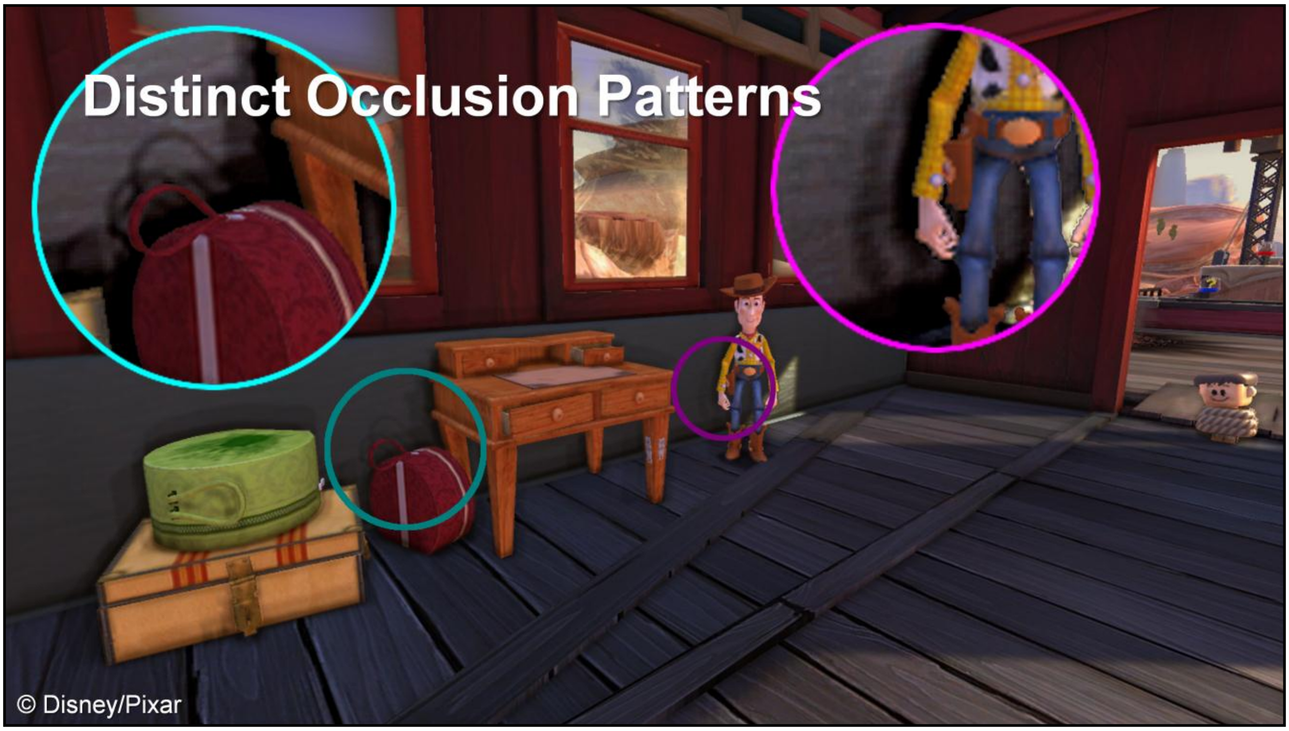
SSAO Challenges

- How/Where to sample
- How to fake more samples
- How to deal with large depth differences



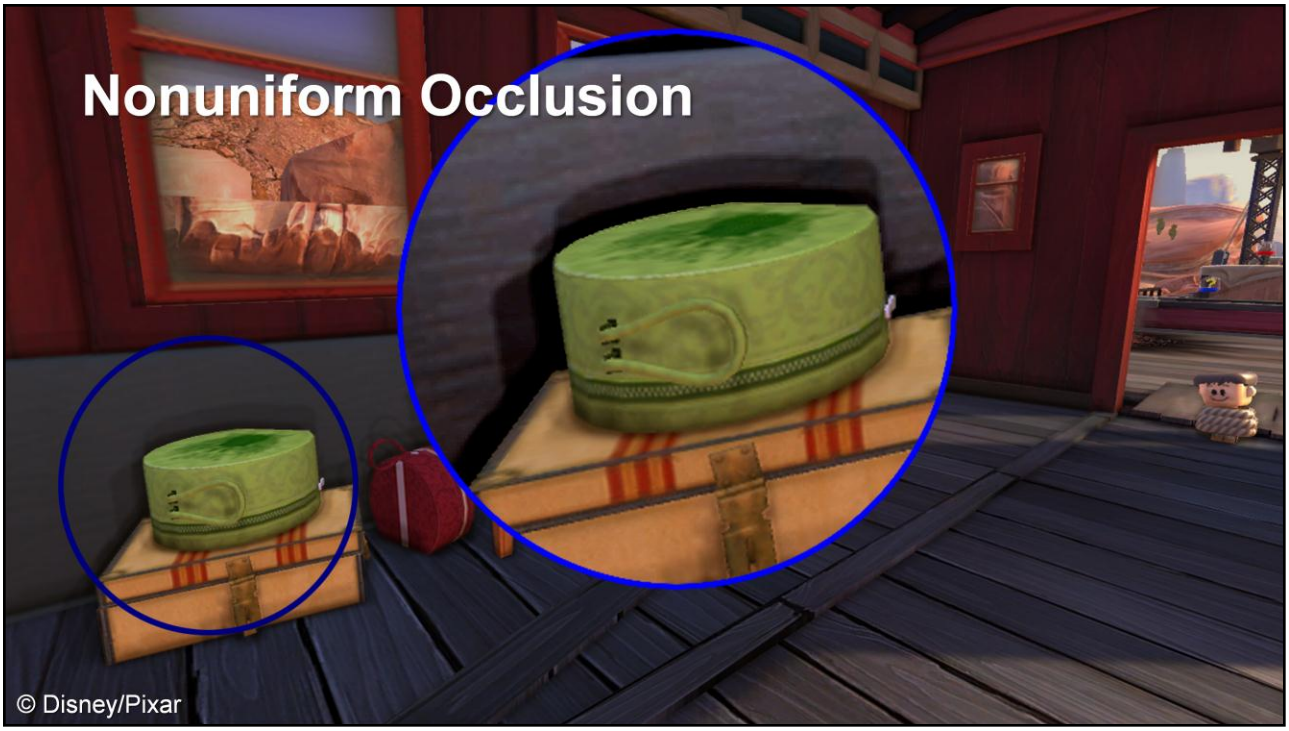
Aliasing Problems with Limited Samples

© Disney/Pixar

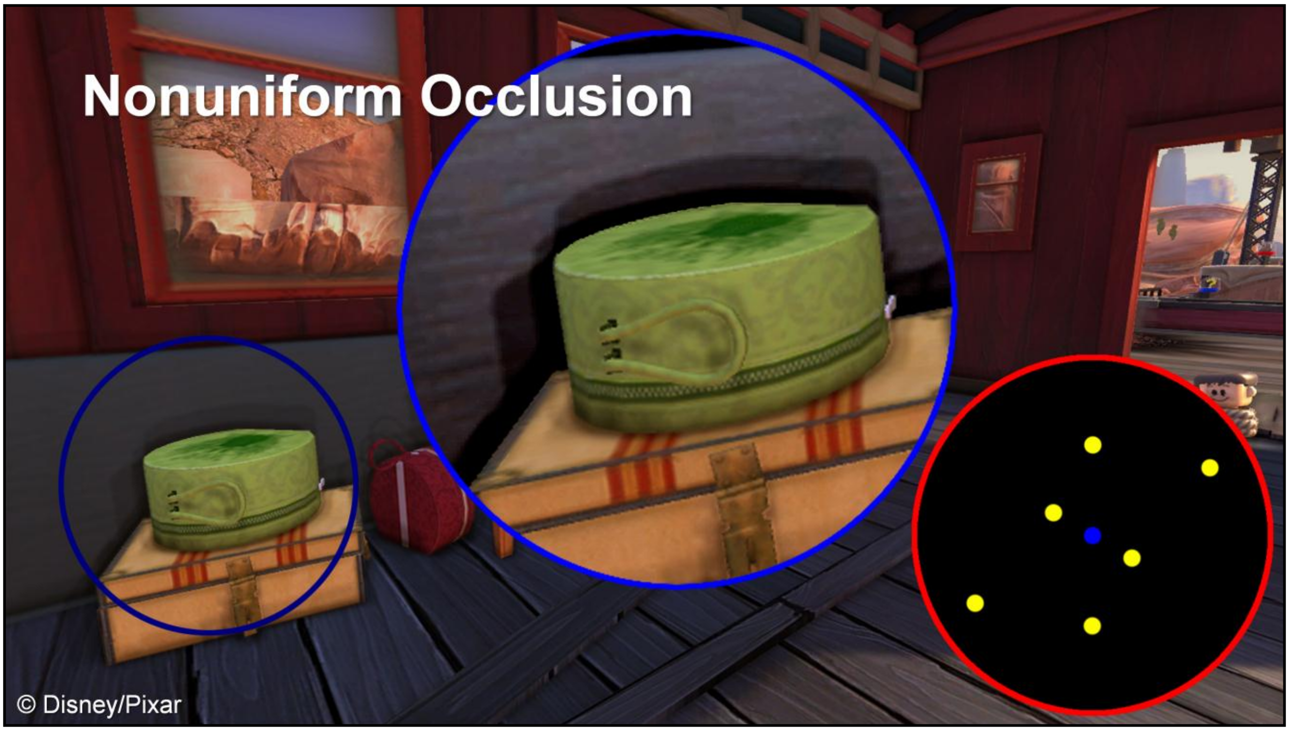


Distinct Occlusion Patterns

© Disney/Pixar



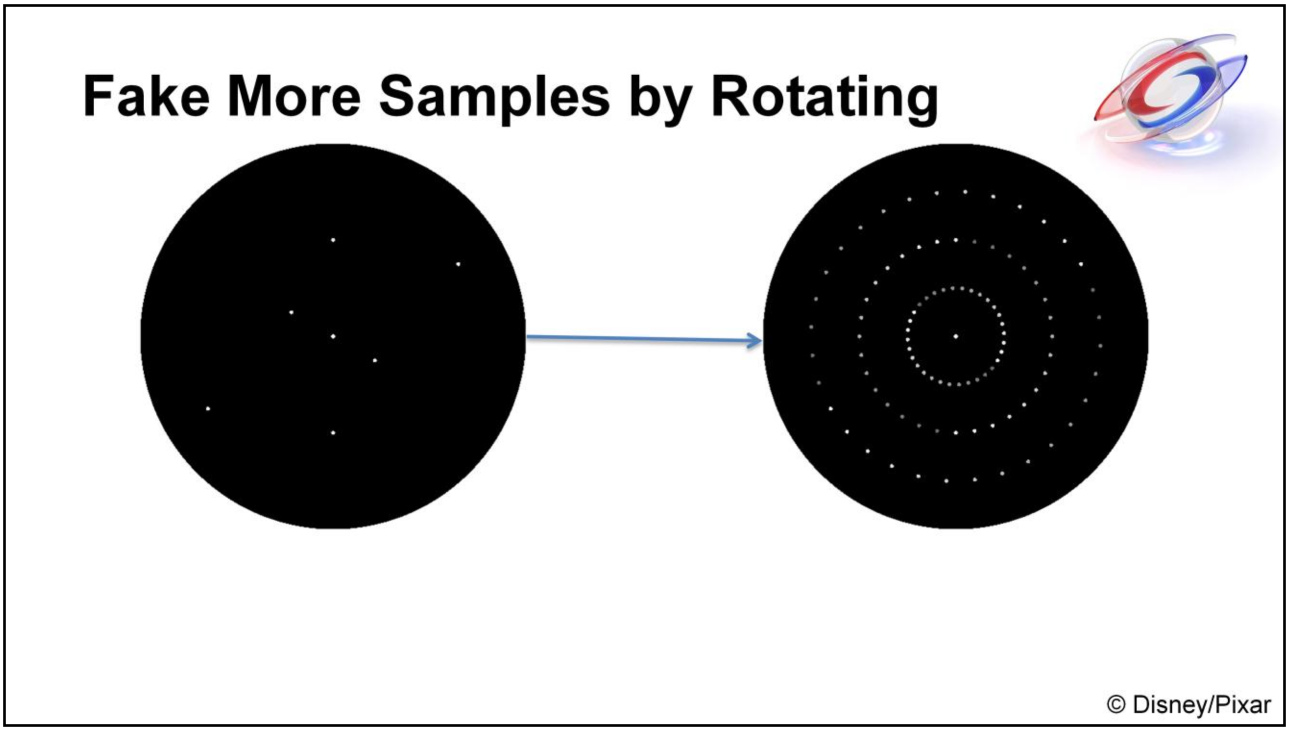
Nonuniform Occlusion



Nonuniform Occlusion

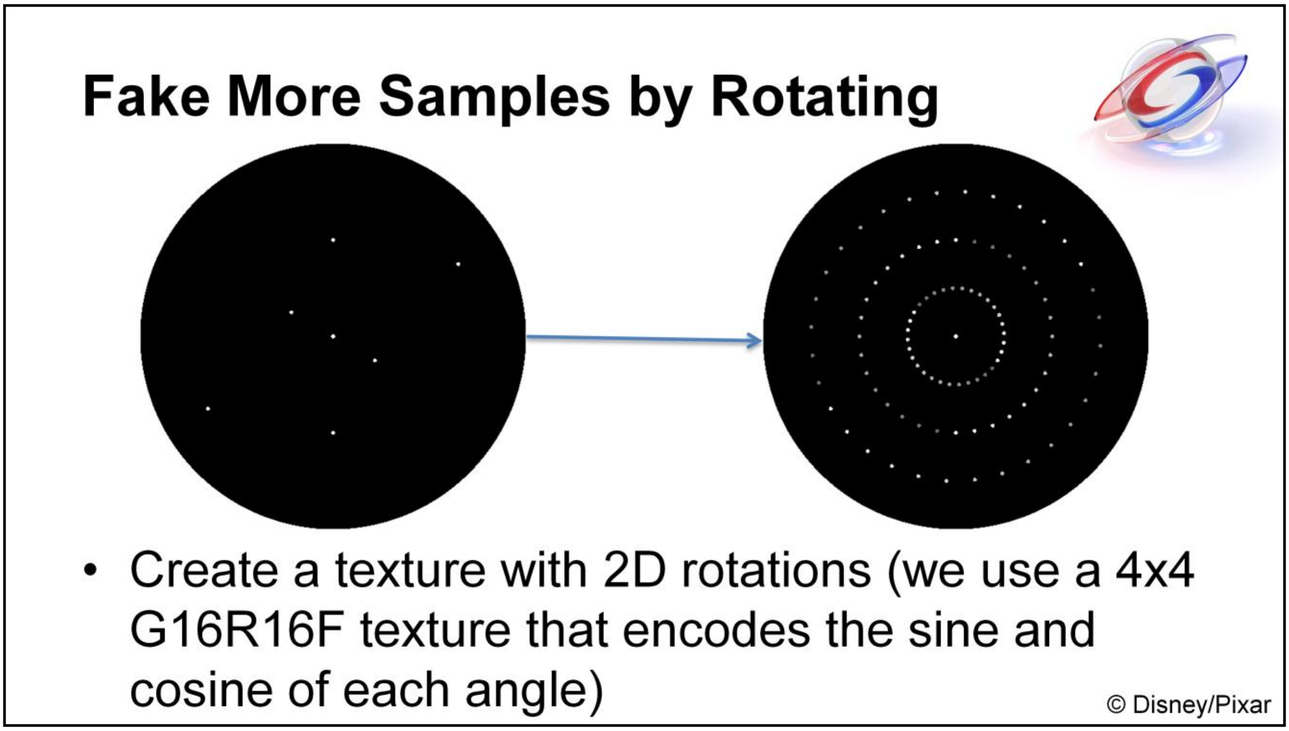
© Disney/Pixar

Our particular paired-sample pattern leads to this strange nonuniformity.



Fake More Samples by Rotating

If each pixel uses the same sampling pattern with a different rotation, it approximates using more samples than is possible in real time.



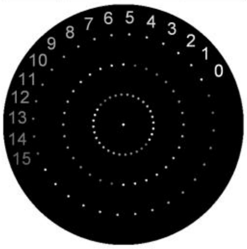
Fake More Samples by Rotating

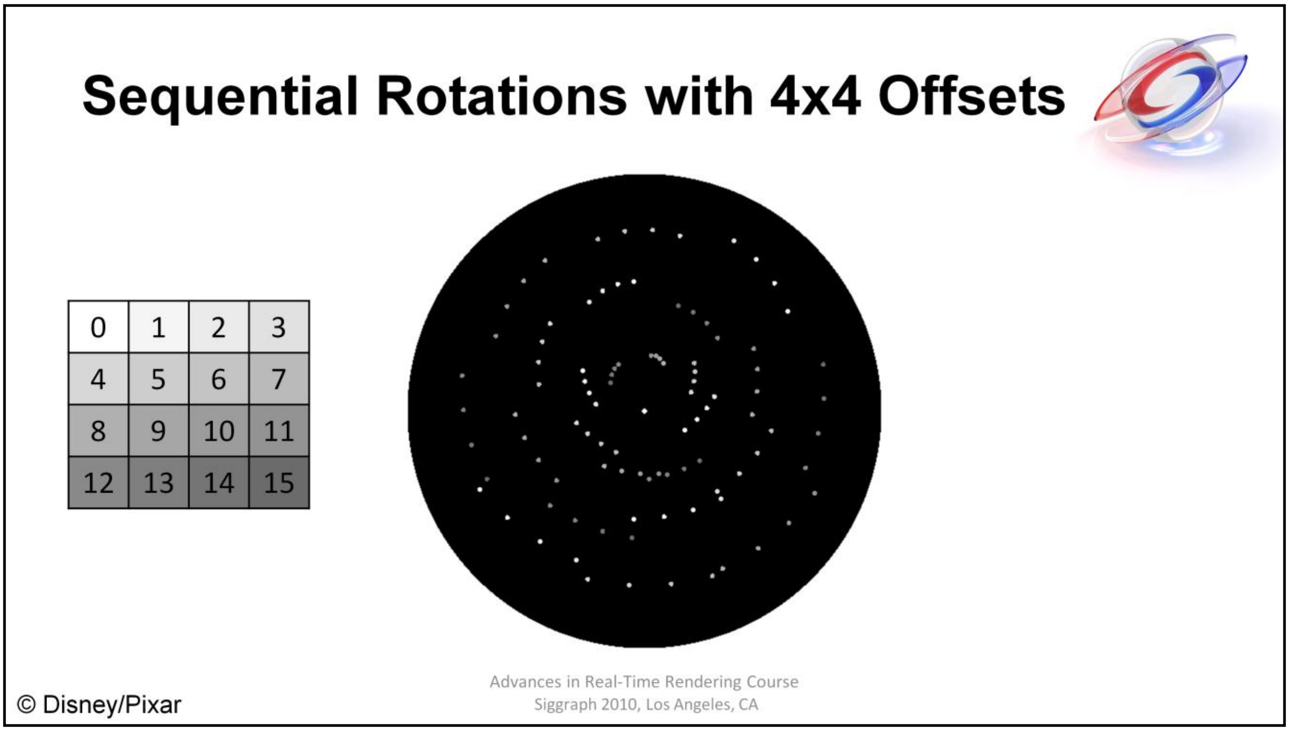
- Create a texture with 2D rotations (we use a 4x4 G16R16F texture that encodes the sine and cosine of each angle)

Rotating samples is easier in 2D than in 3D.

Rotations Encoded in a 4x4 Texture

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15





Sequential Rotations with 4x4 Offsets

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

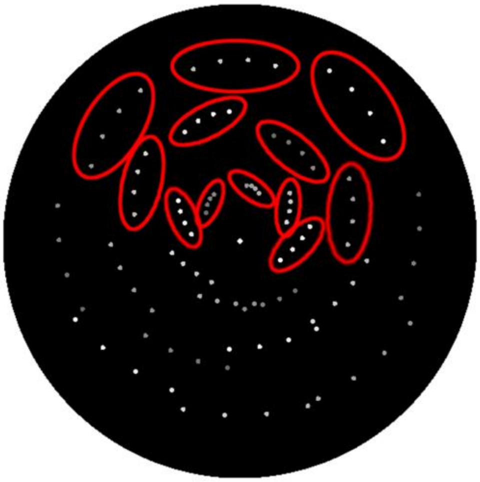
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

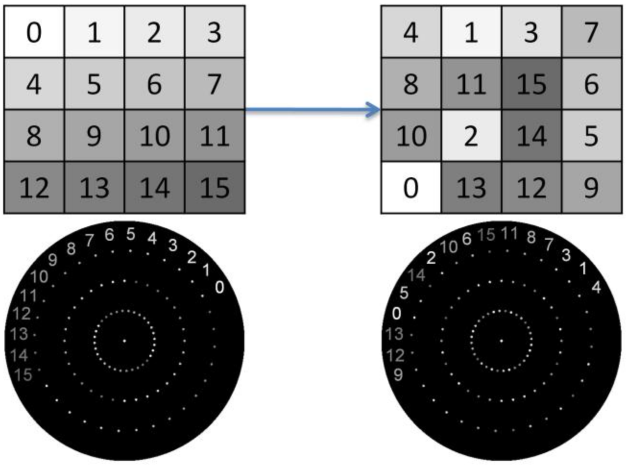
If you encode the rotations sequentially, it leads to clumps of samples.

Sequential Rotations with 4x4 Offsets

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



Randomized Rotations

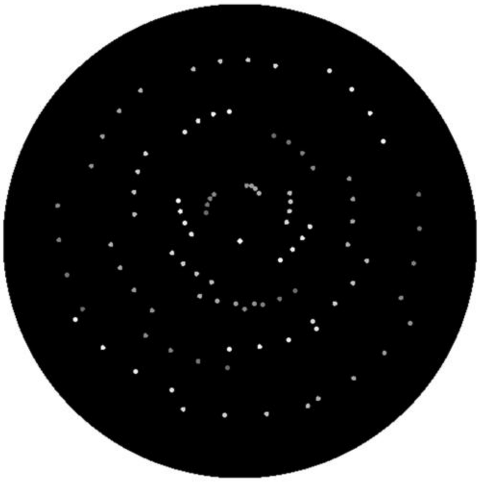


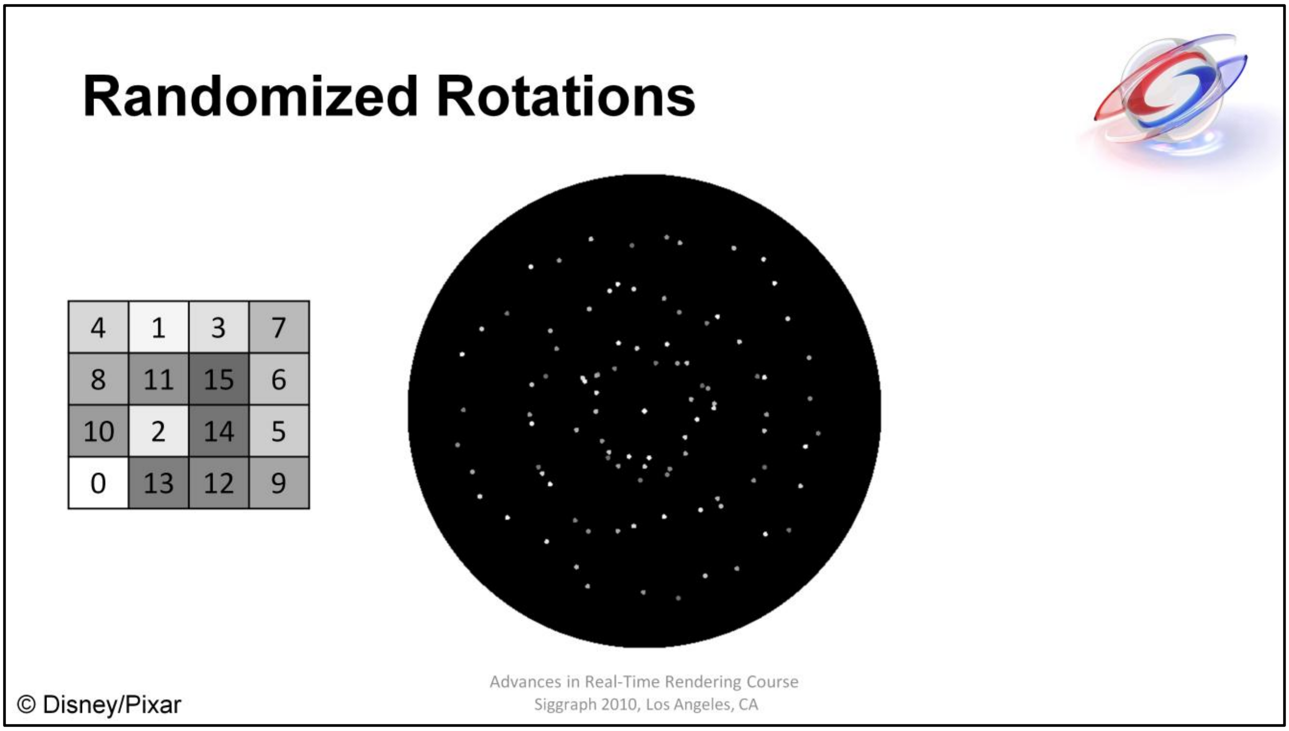
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

4	1	3	7
8	11	15	6
10	2	14	5
0	13	12	9

Sequential Rotations with 4x4 Offsets

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



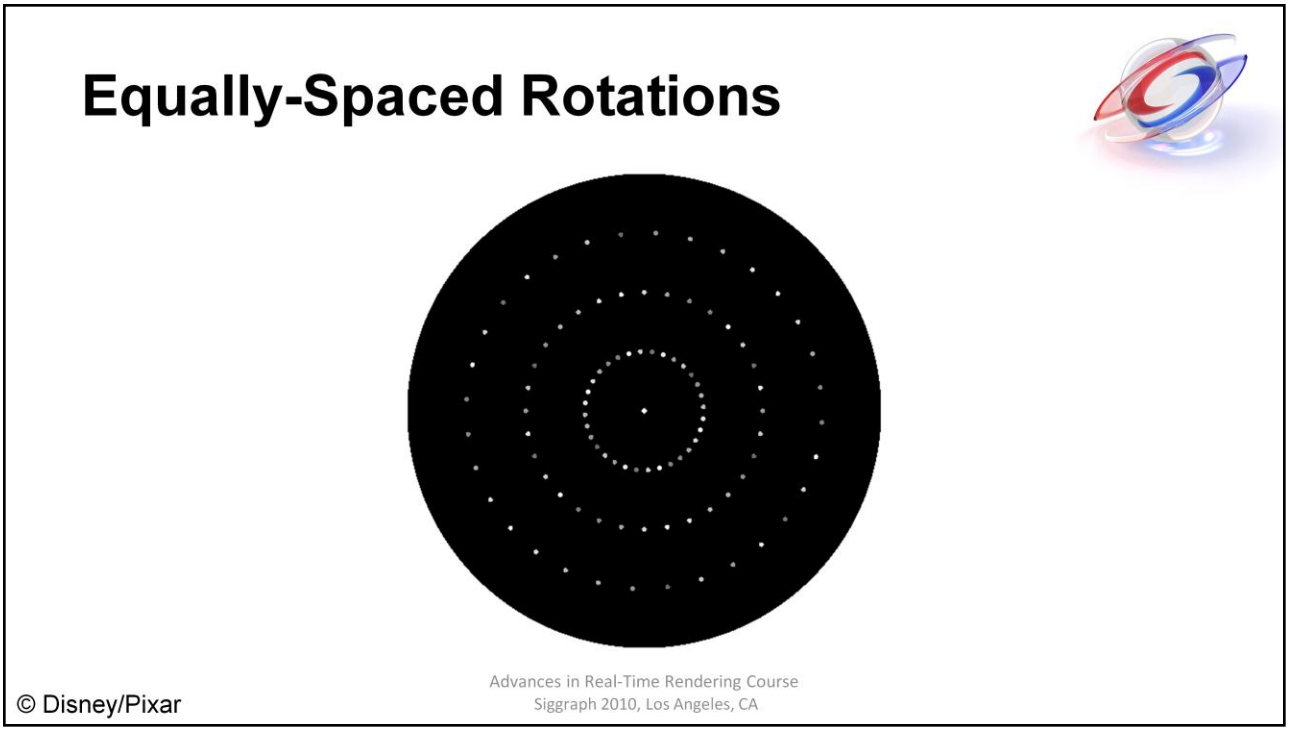


Randomized Rotations

4	1	3	7
8	11	15	6
10	2	14	5
0	13	12	9

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

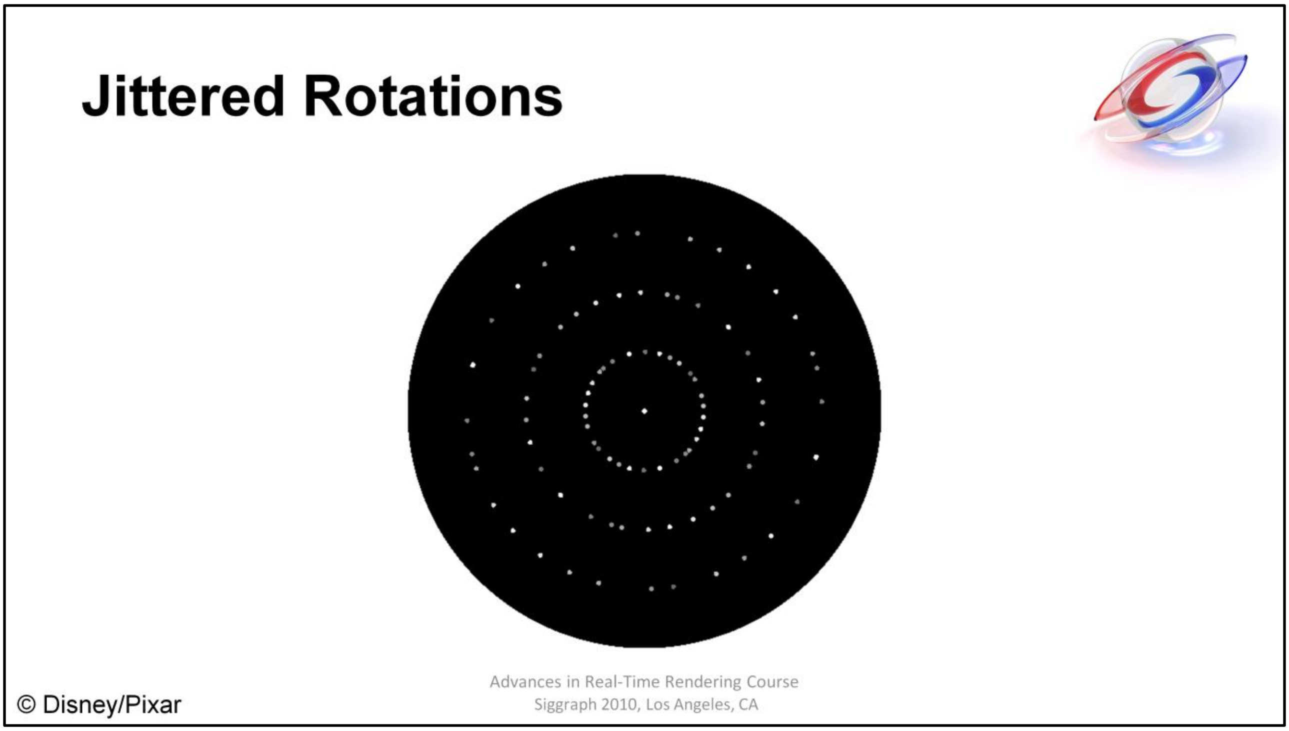


Equally-Spaced Rotations

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

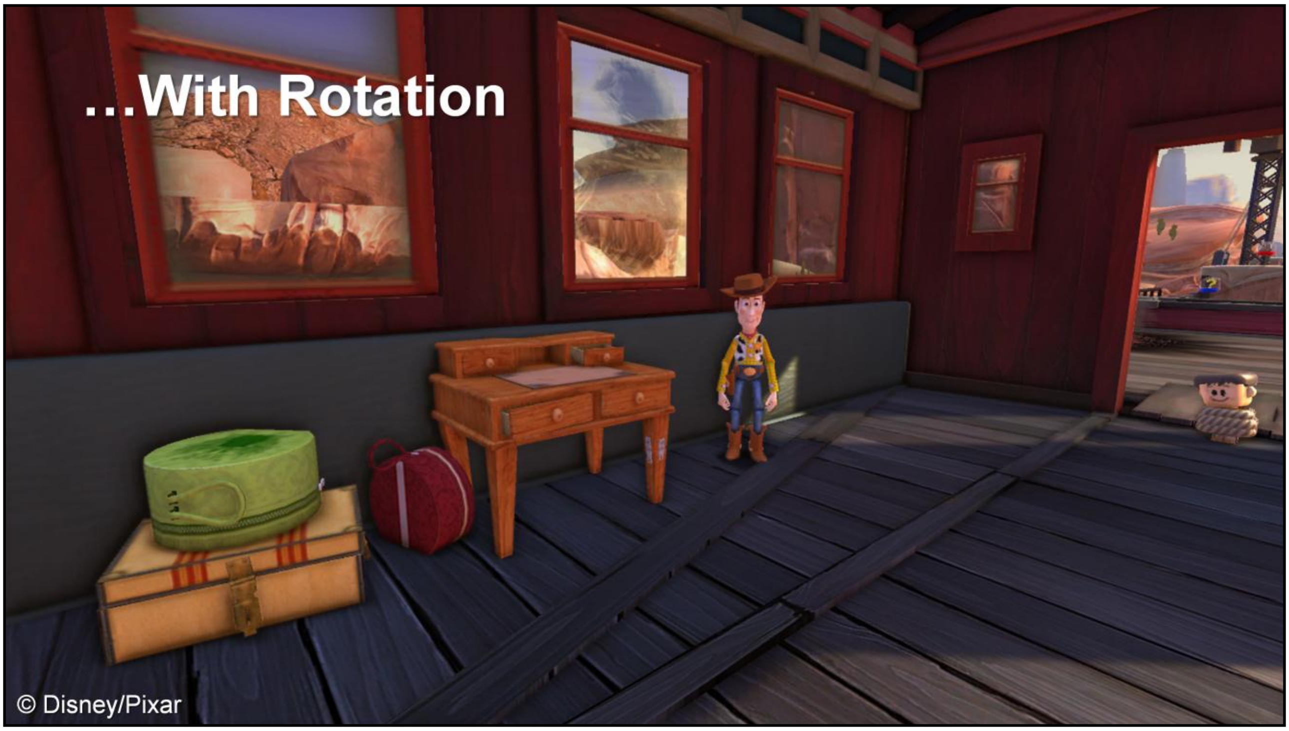
Equally-spaced rotations are fine, but a bit of jitter isn't a bad idea.



Jittered Rotations

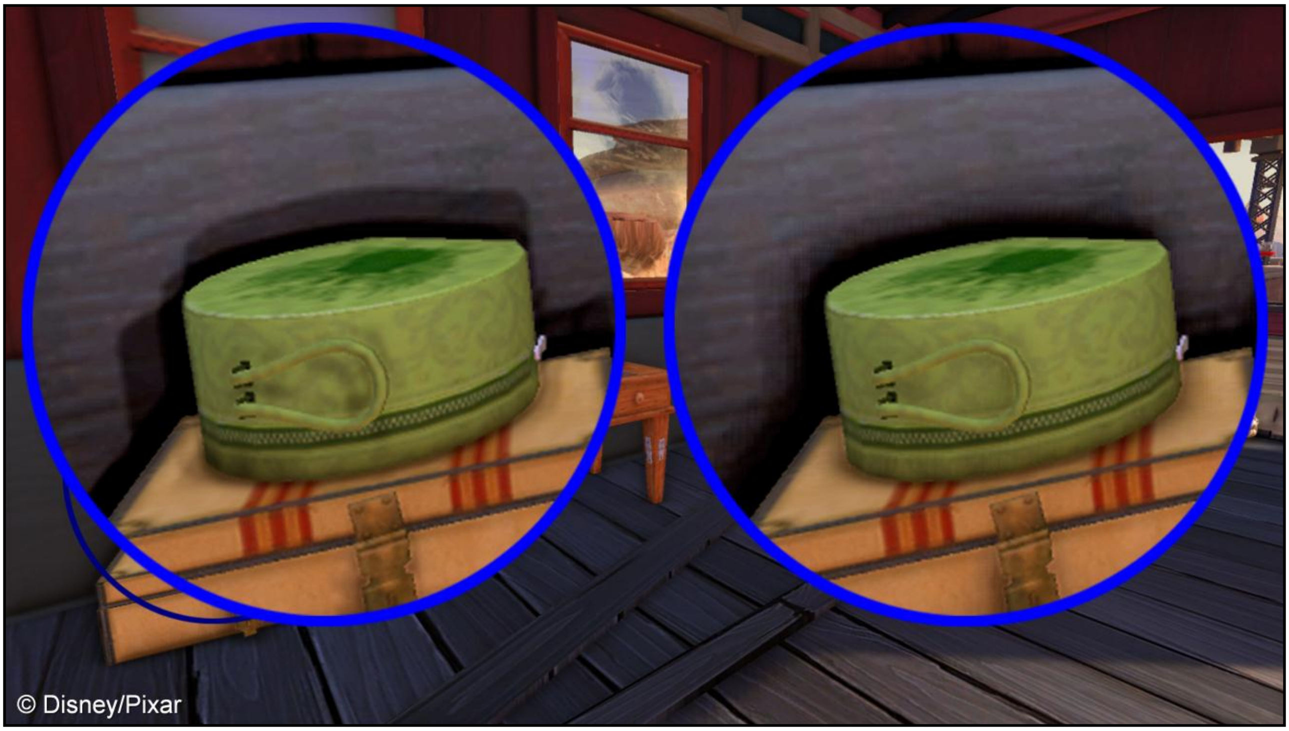
No Rotation…

© Disney/Pixar

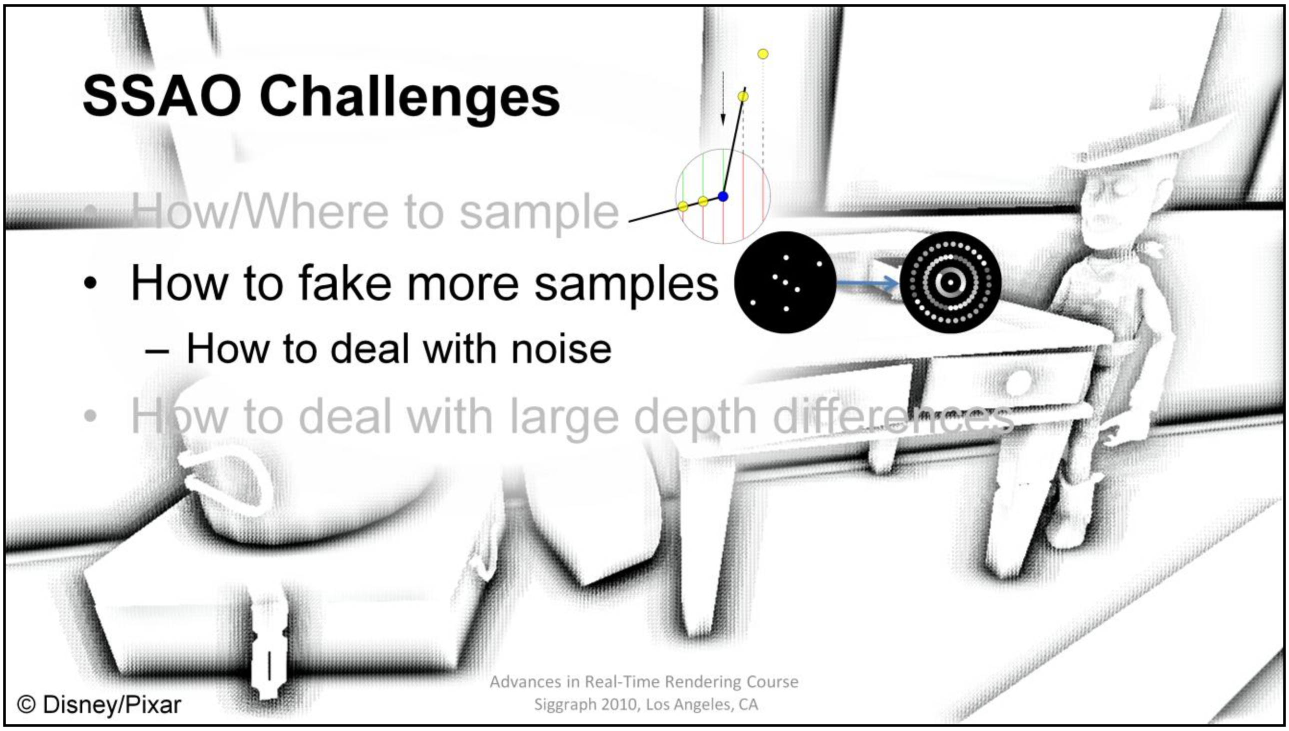


…With Rotation

© Disney/Pixar



© Disney/Pixar



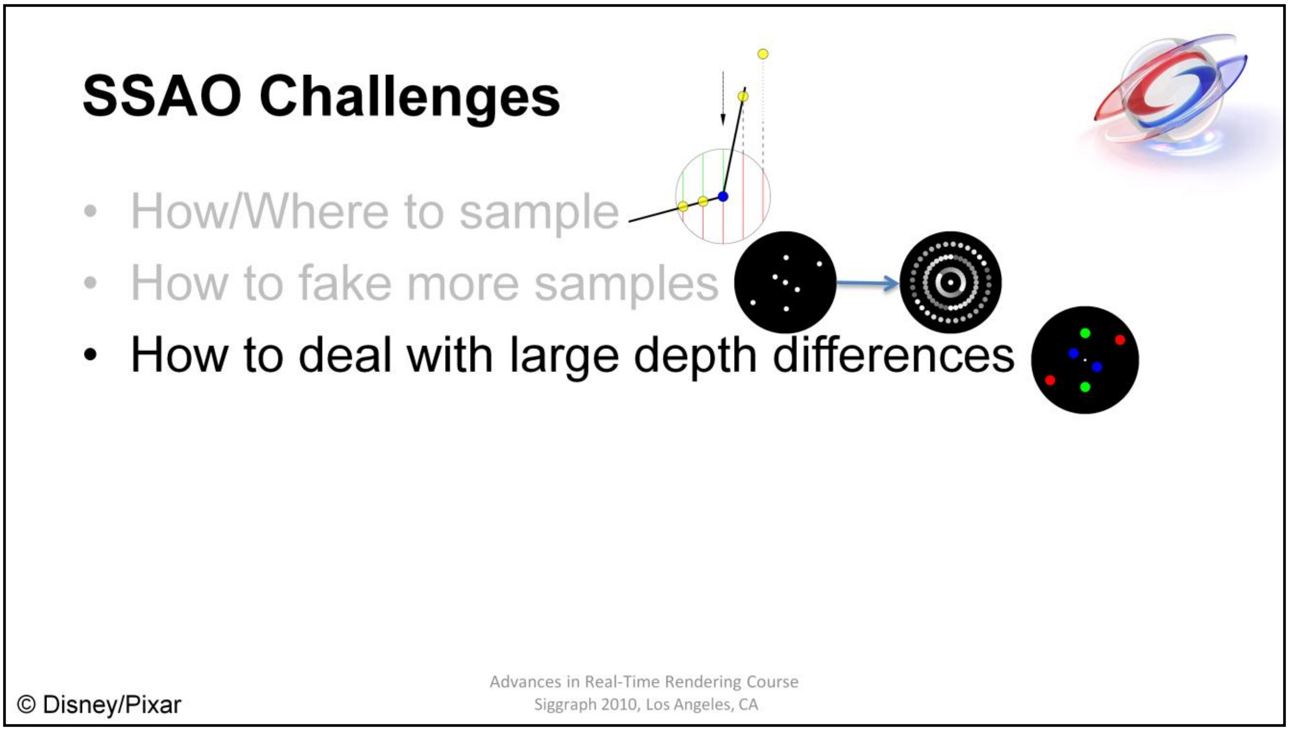
SSAO Challenges

- How/Where to sample
- How to fake more samples
  - How to deal with noise
- How to deal with large depth differences

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Faking extra samples results in noise, and can be dealt with using an edge-aware blur. This isn't different from any of the other SSAO methods, though, and so I won't be covering it.

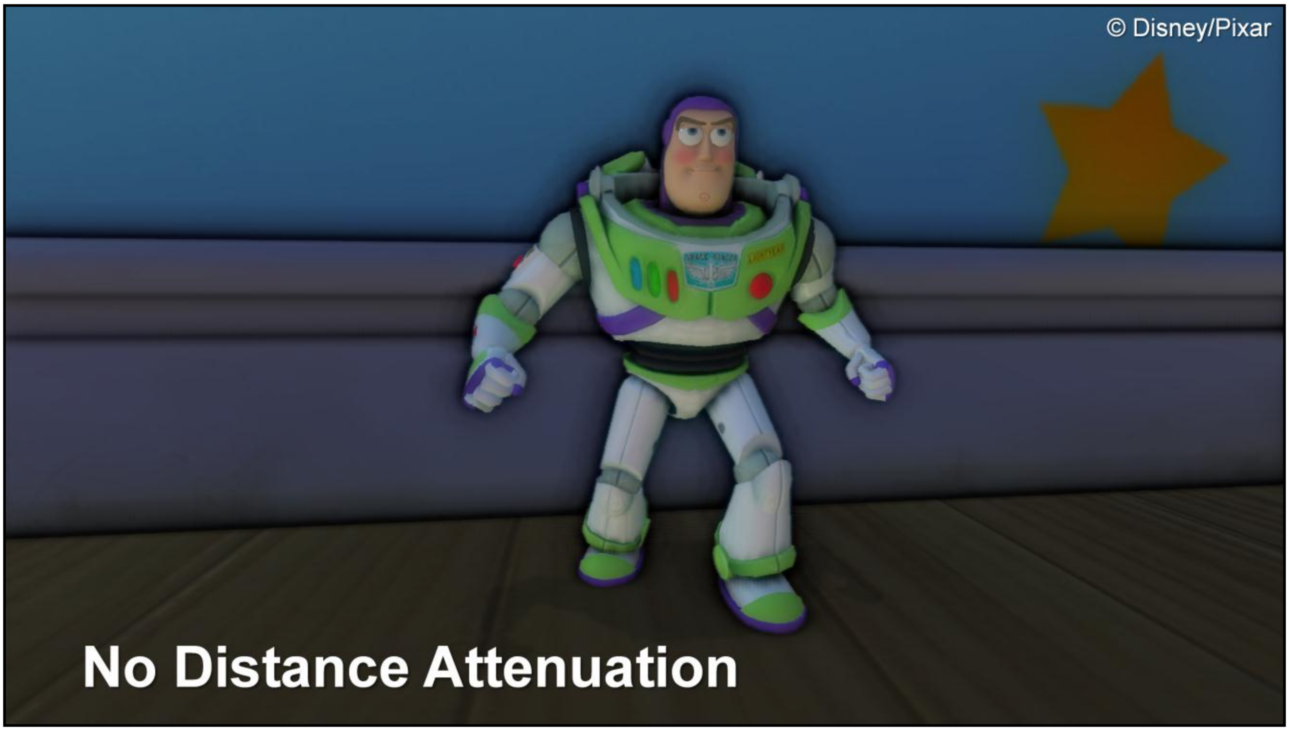


SSAO Challenges

- How/Where to sample
- How to fake more samples
- How to deal with large depth differences

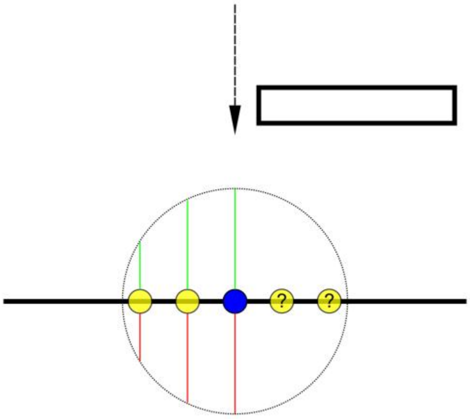
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

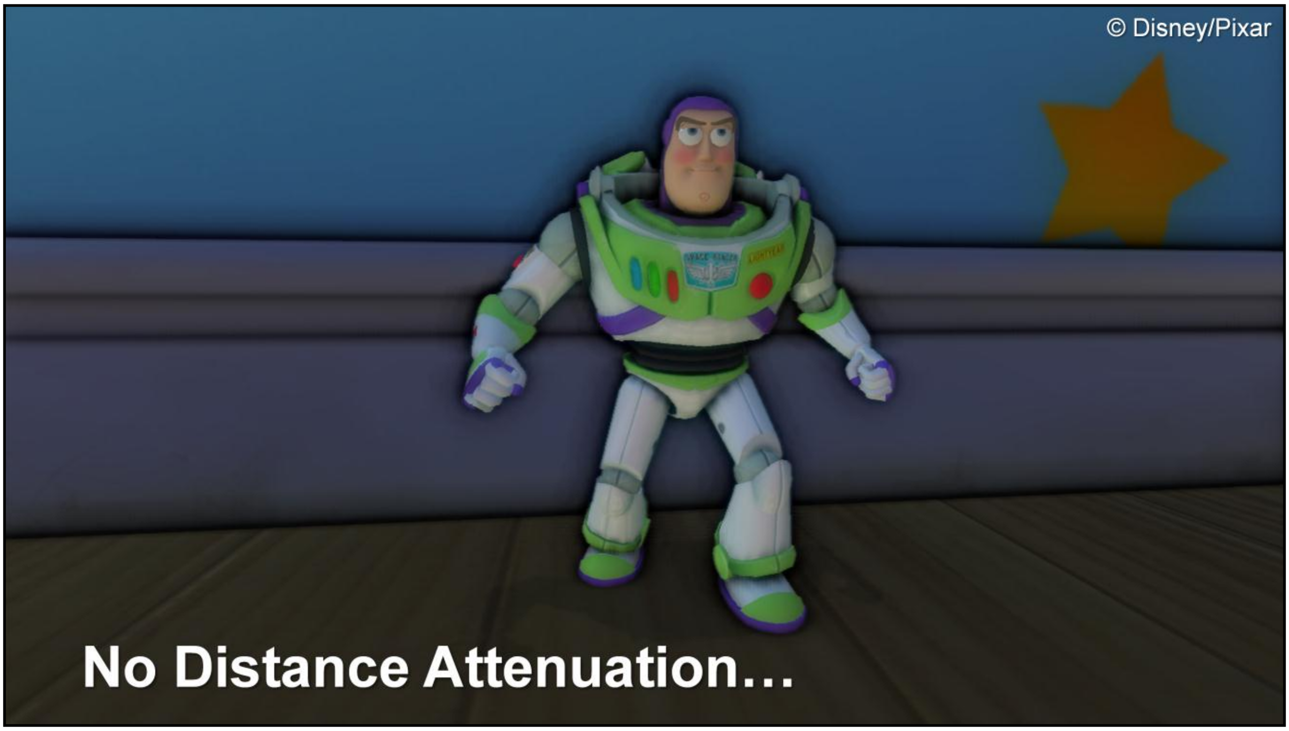


Every part of Buzz Lightyear occludes the wall, which is not what we want.

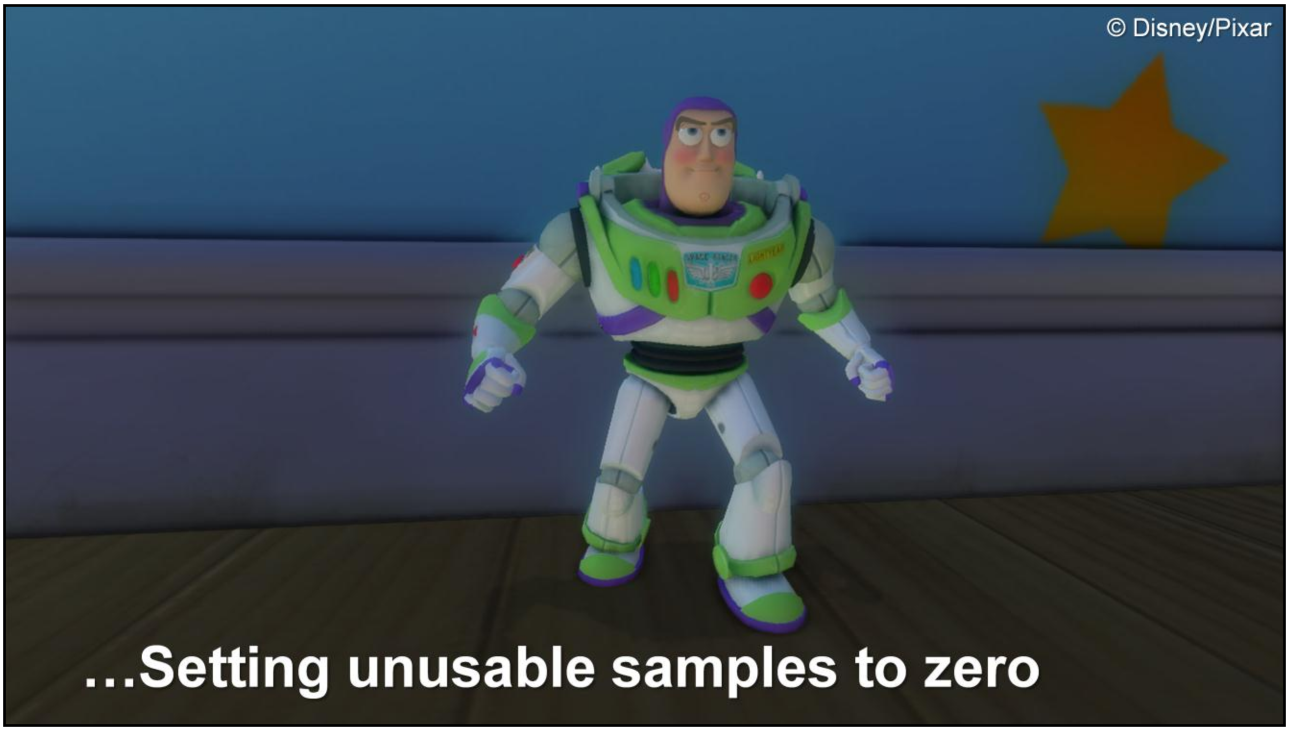
What about setting unusable samples to zero?



Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA



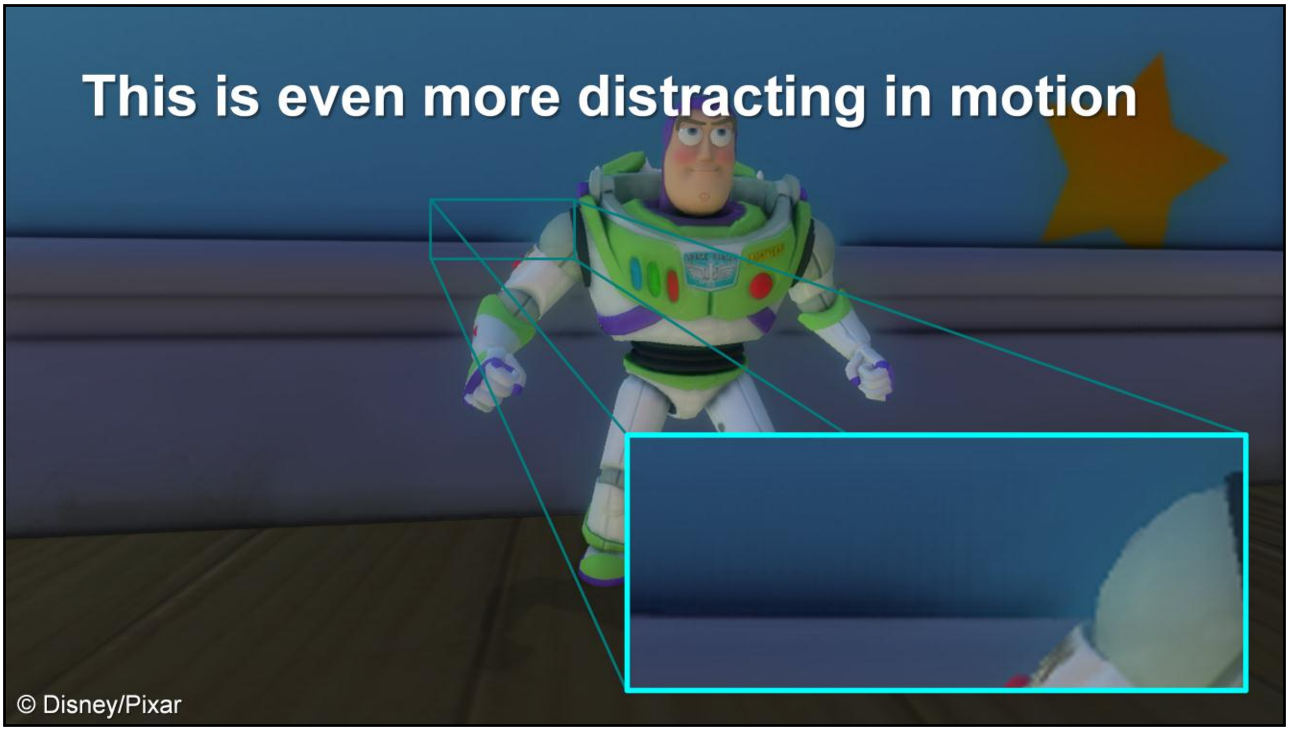
No Distance Attenuation…



© Disney/Pixar

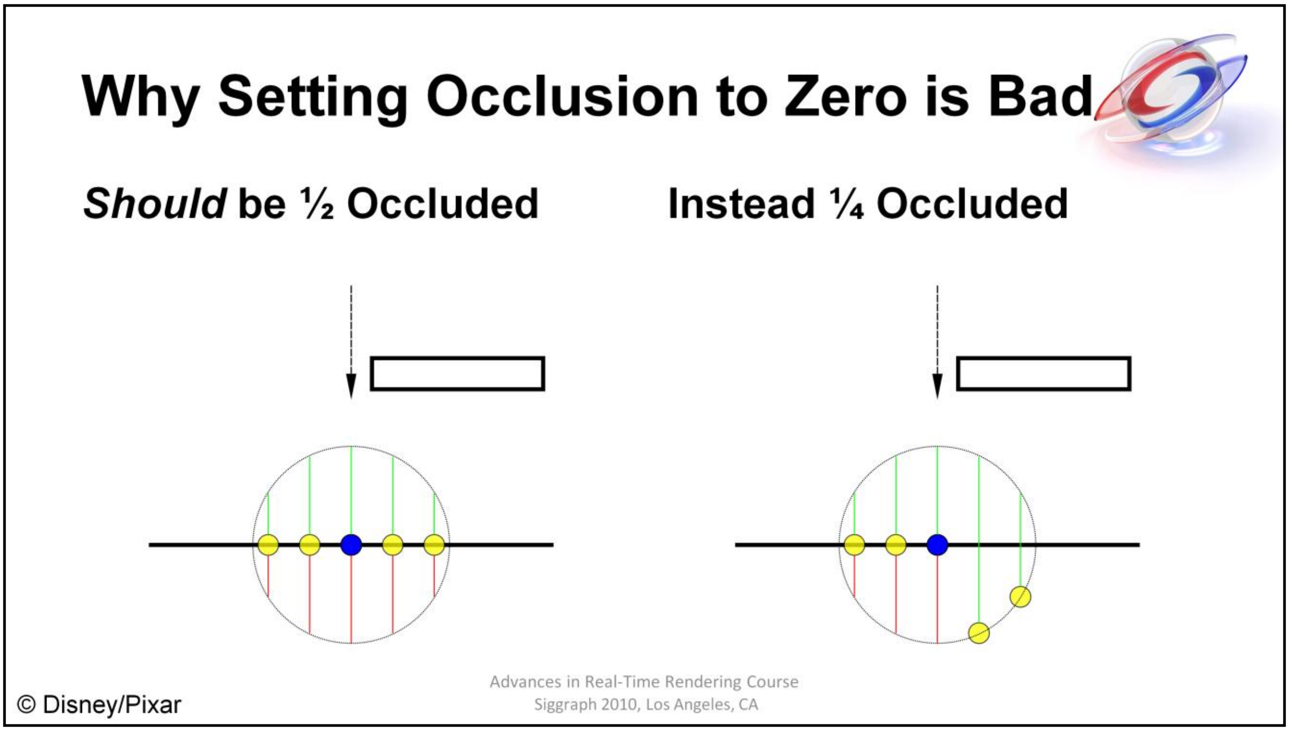
…Setting unusable samples to zero

This fixes the problem, but introduces a new one. Notice the halo around him where all occlusion is missing.



This is even more distracting in motion

© Disney/Pixar



Why Setting Occlusion to Zero is Bad

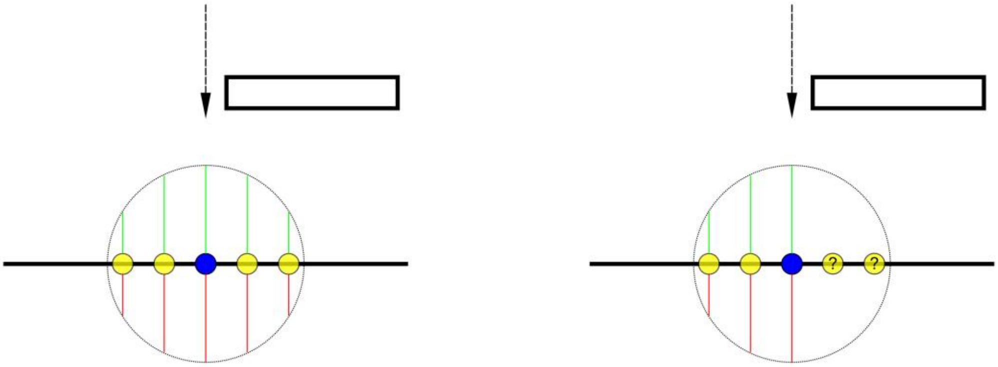
Should **be ½ Occluded** **Instead ¼ Occluded**

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The problem is that a flat plane should be ½ occluded, so setting unusable samples to zero skews the results toward being too unoccluded, leading to the halos.

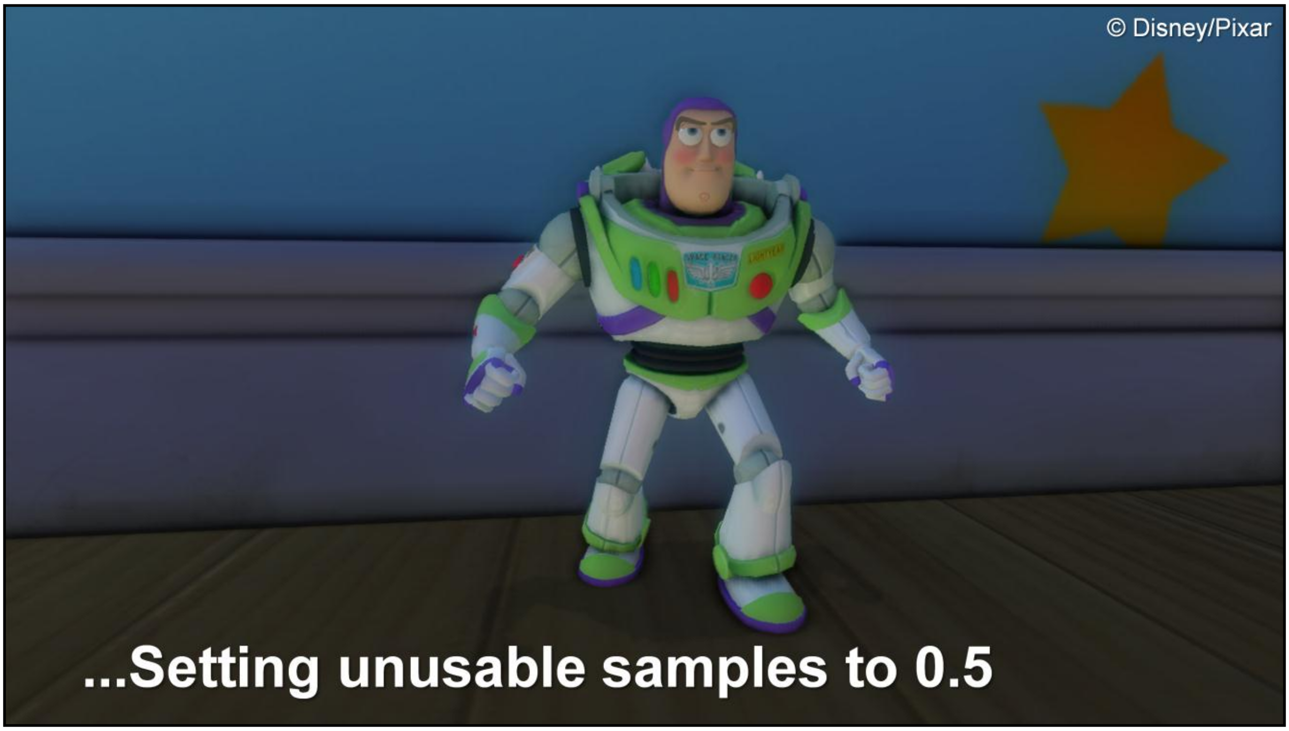
What about using 0.5 instead of 0?



© Disney/Pixar

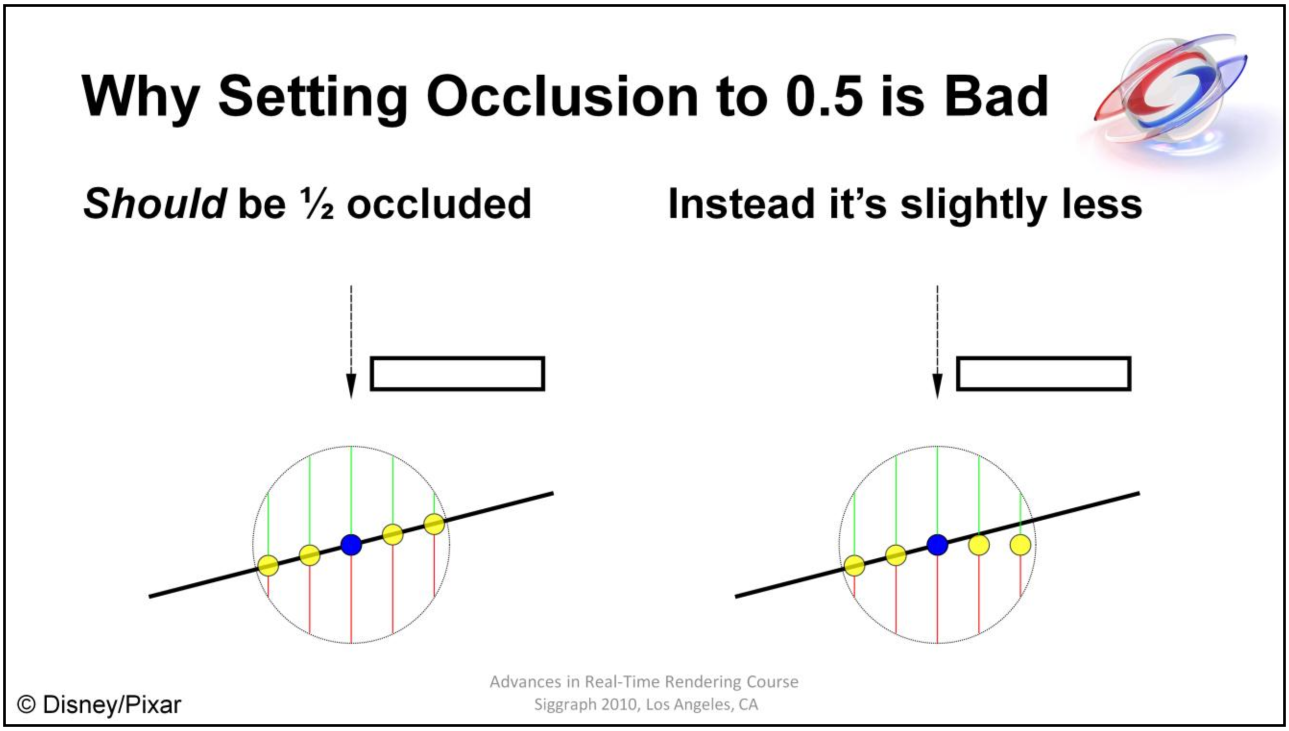
© Disney/Pixar

Still setting unusable samples to zero…



© Disney/Pixar

...Setting unusable samples to 0.5



Why Setting Occlusion to 0.5 is Bad

***Should* be ½ occluded**

Instead it's slightly less

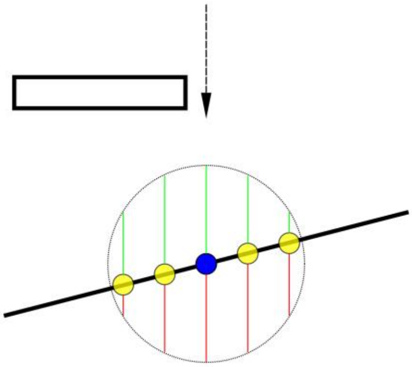
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

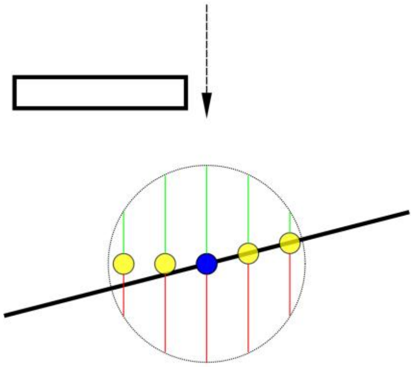
Using 0.5 works perfectly when there's a flat plane parallel with the view plane, but it breaks down with angled surfaces. In the figures above the result will be too little occlusion, and if the occluder were on the other side it would result in too much occlusion.

Why Setting Occlusion to 0.5 is Bad

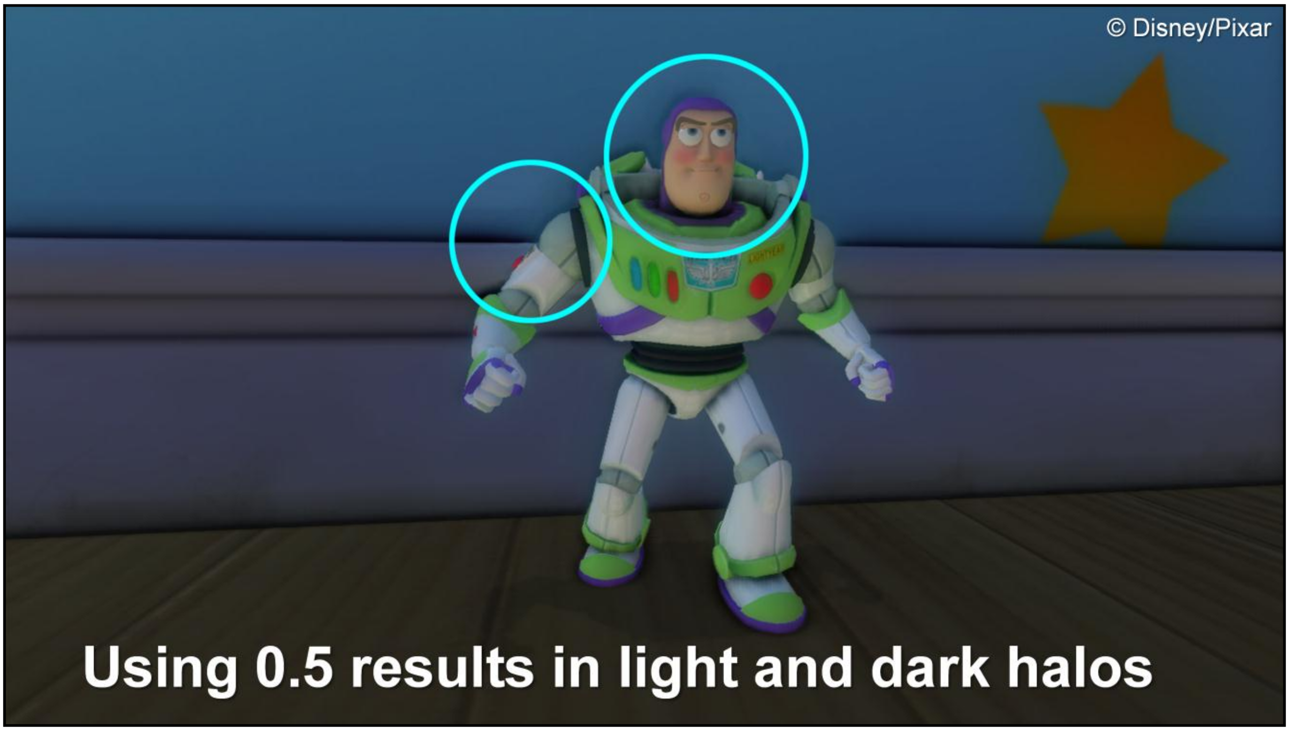
Should be ½ occluded



Instead it's slightly more

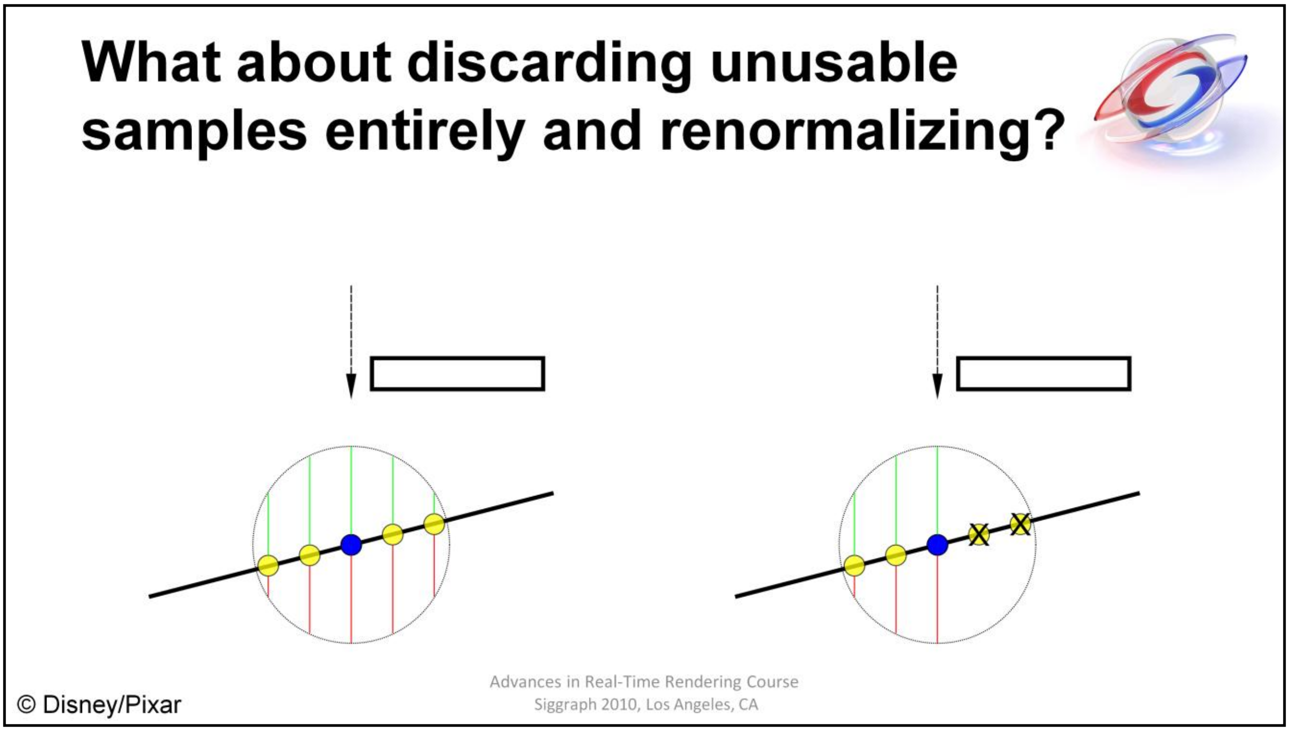


© Disney/Pixar



© Disney/Pixar

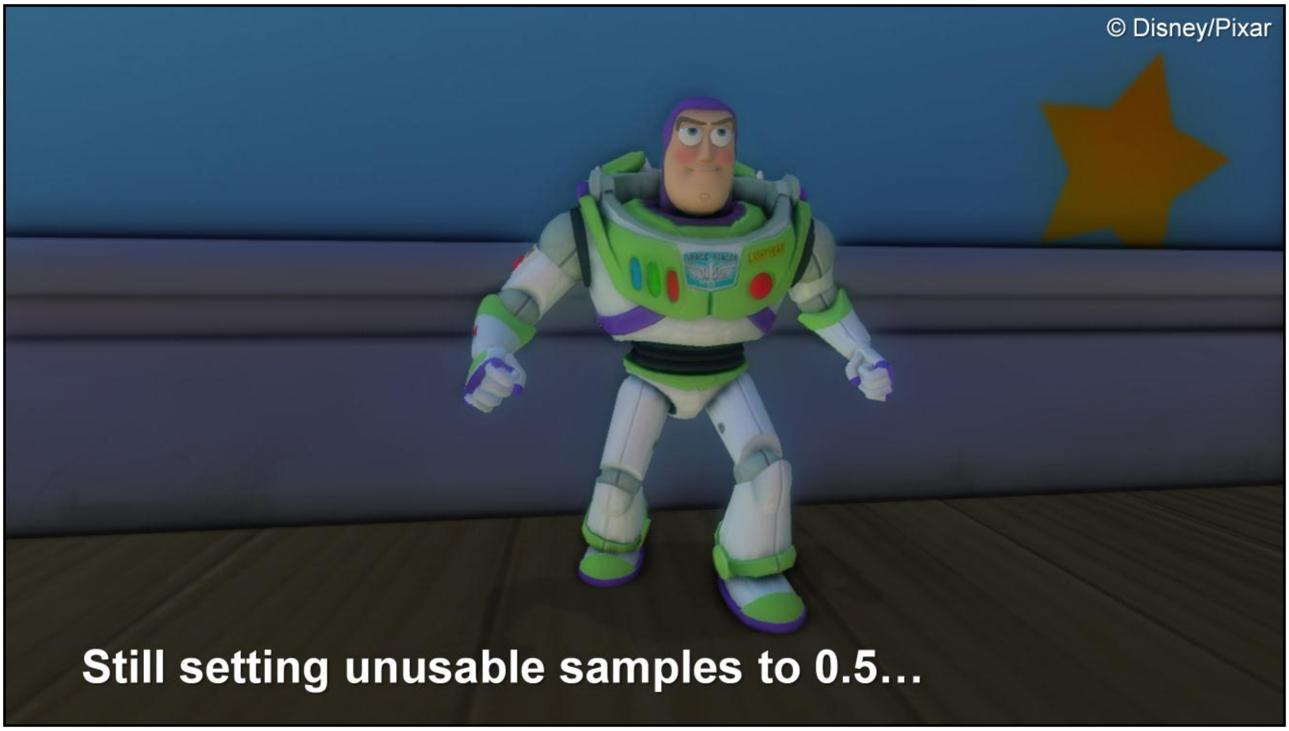
Using 0.5 results in light and dark halos



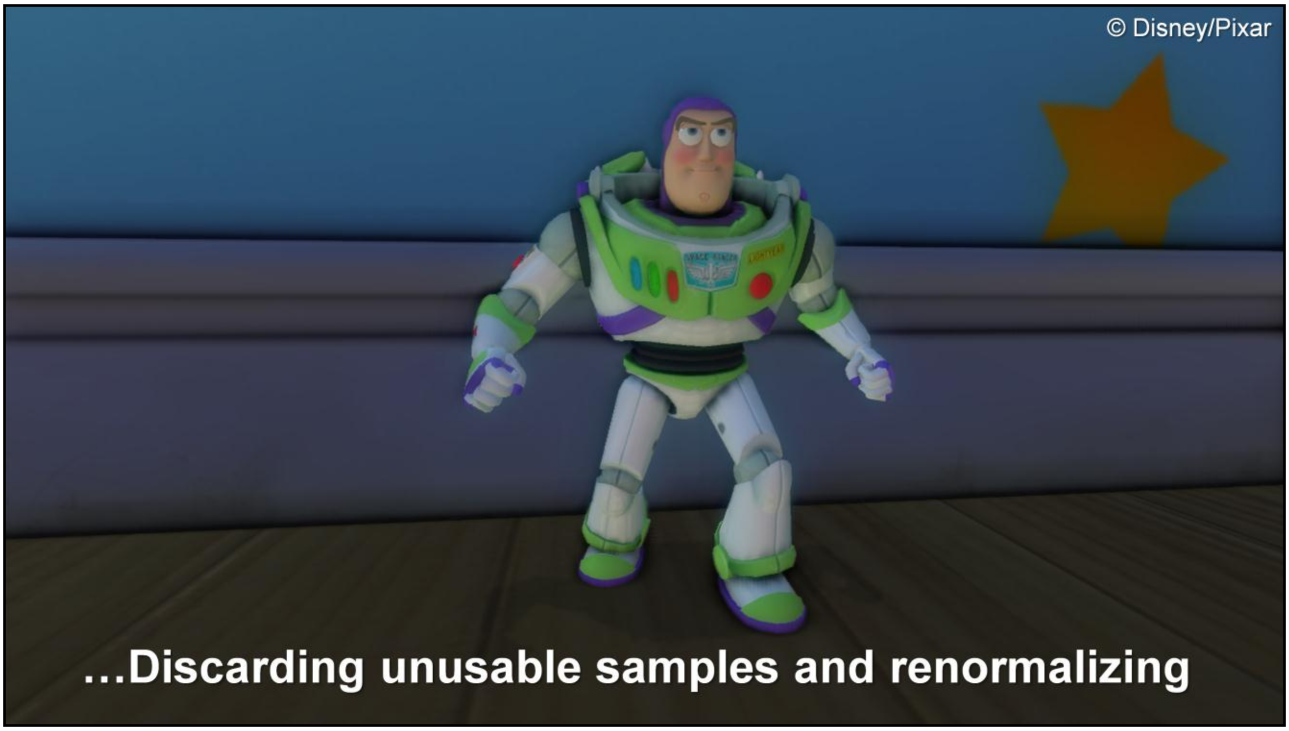
What about discarding unusable samples entirely and renormalizing?

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

We could try to only use valid samples rather than try to guess what information we’re missing.

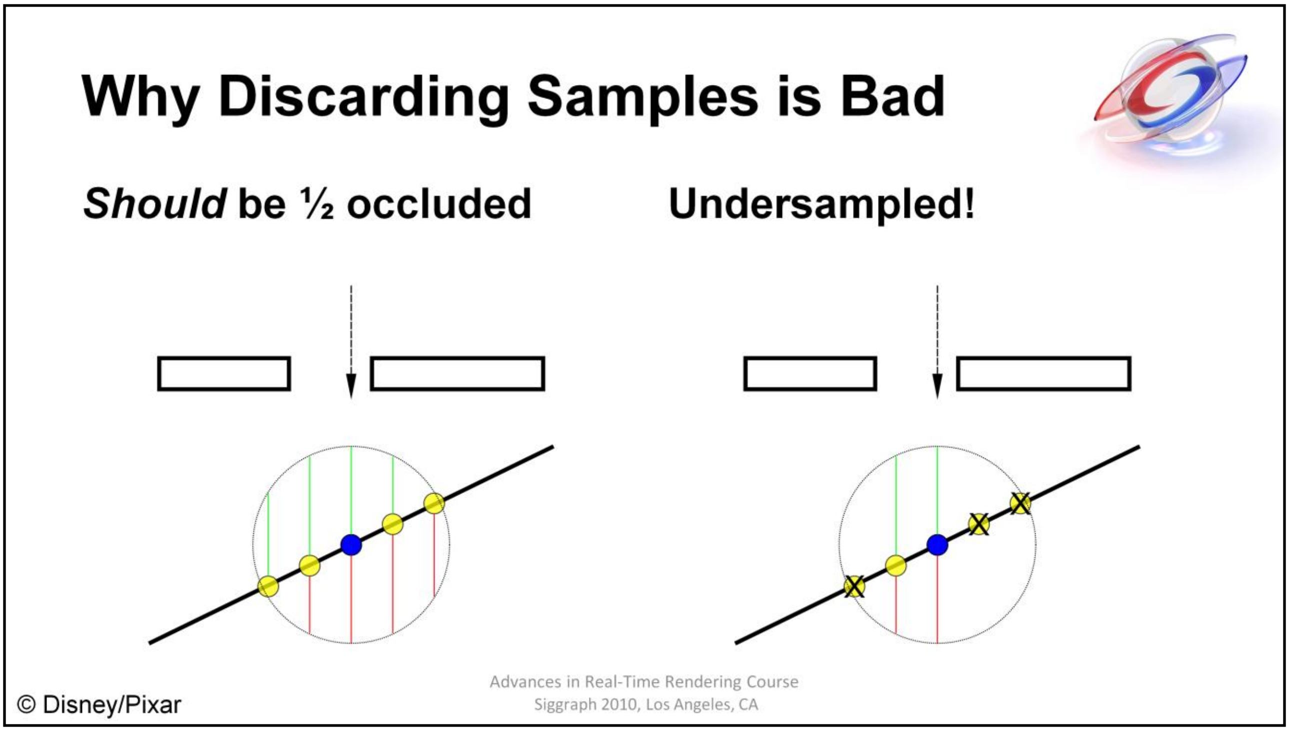


Still setting unusable samples to 0.5…



© Disney/Pixar

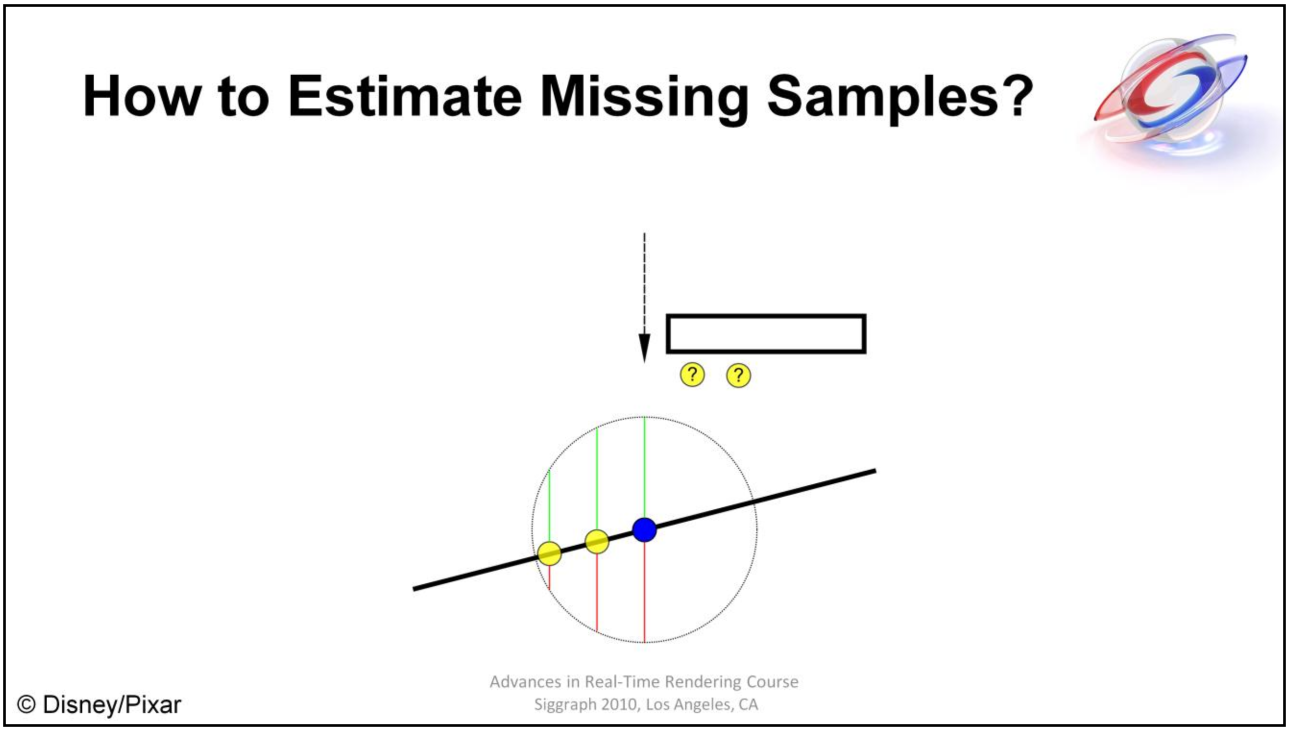
…Discarding unusable samples and renormalizing



Why Discarding Samples is Bad

Should be ½ occluded

Undersampled!

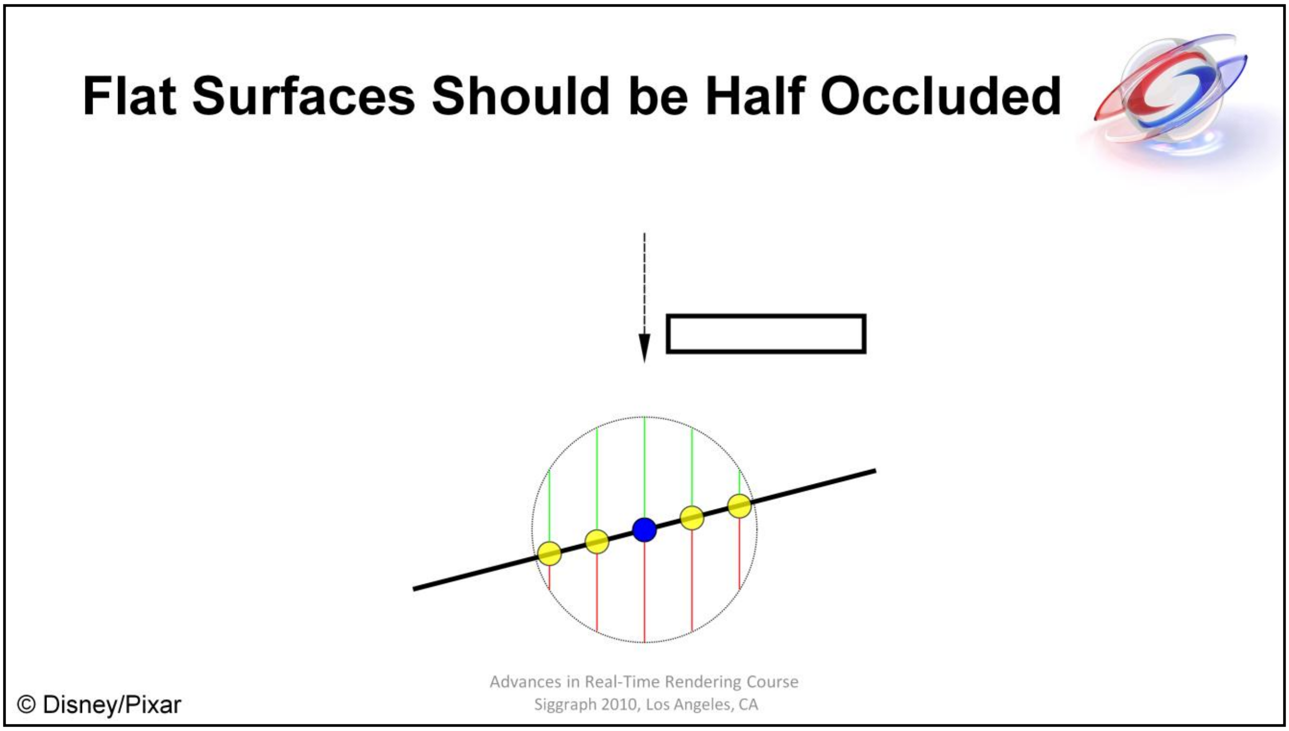


How to Estimate Missing Samples?

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

We would like some way of intelligently estimating the value of samples that we can’t use.



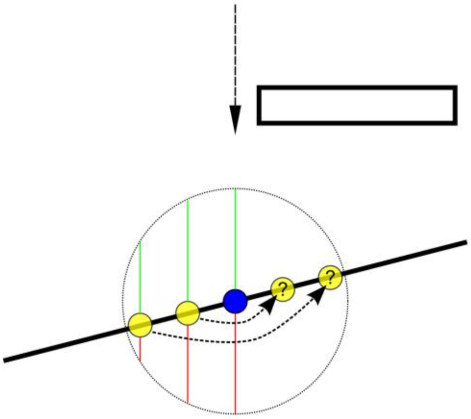
Flat Surfaces Should be Half Occluded

© Disney/Pixar

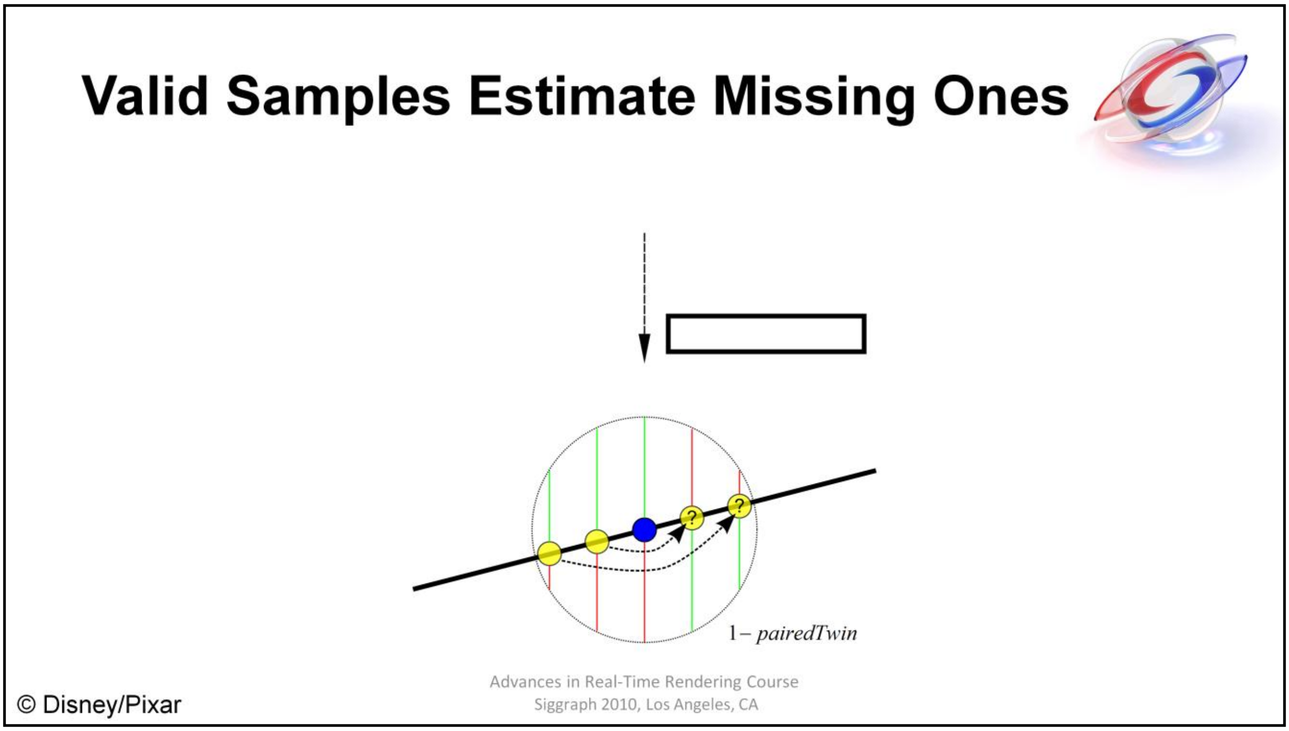
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

We have no way of knowing what information we're missing, but we can at least try to aim for the common case, which is a flat surface. In that case, all of the samples should average 0.5.

Valid Samples Estimate Missing Ones



If we choose our sampling pattern carefully, we can estimate the value for unusable samples to “cancel out” valid ones. This assumes a flat surface, which is an attempt to find a common case.

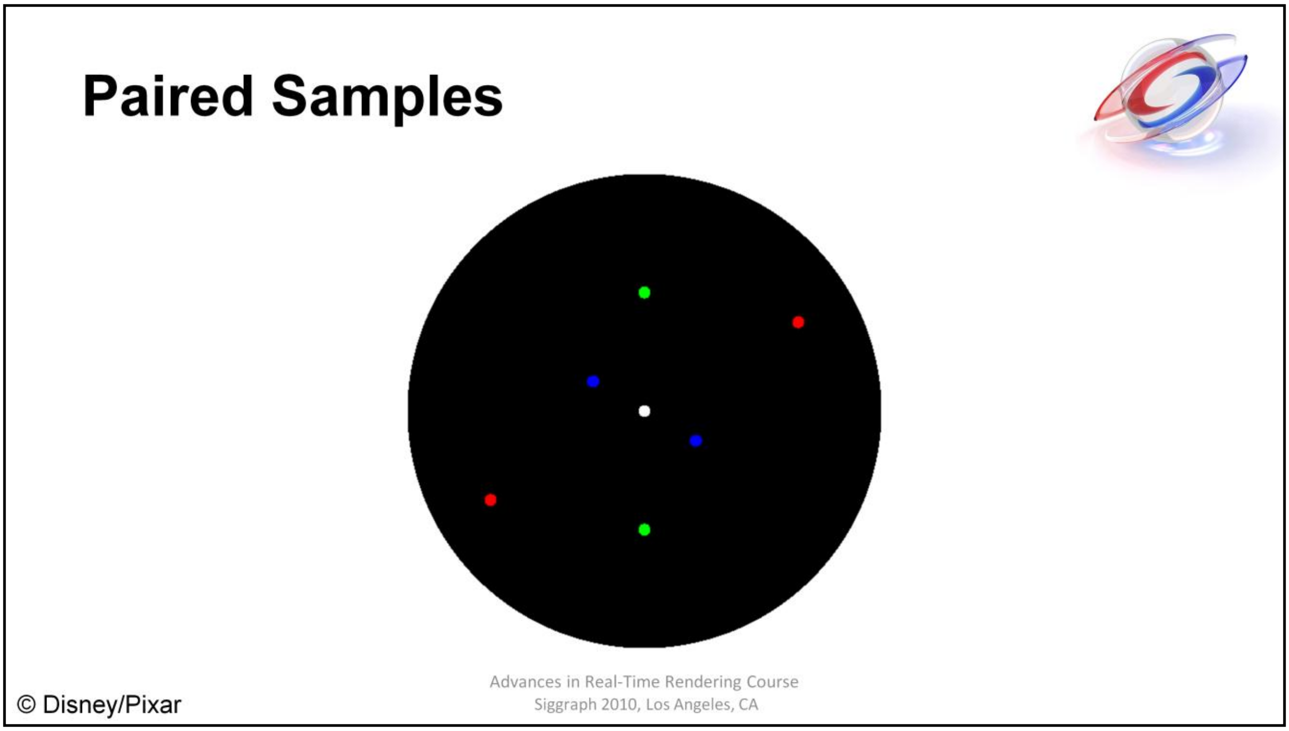


Valid Samples Estimate Missing Ones

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

© Disney/Pixar

If we choose our sampling pattern carefully, we can estimate the value for unusable samples to “cancel out” valid ones. This assumes a flat surface, which is an attempt to find a common case.

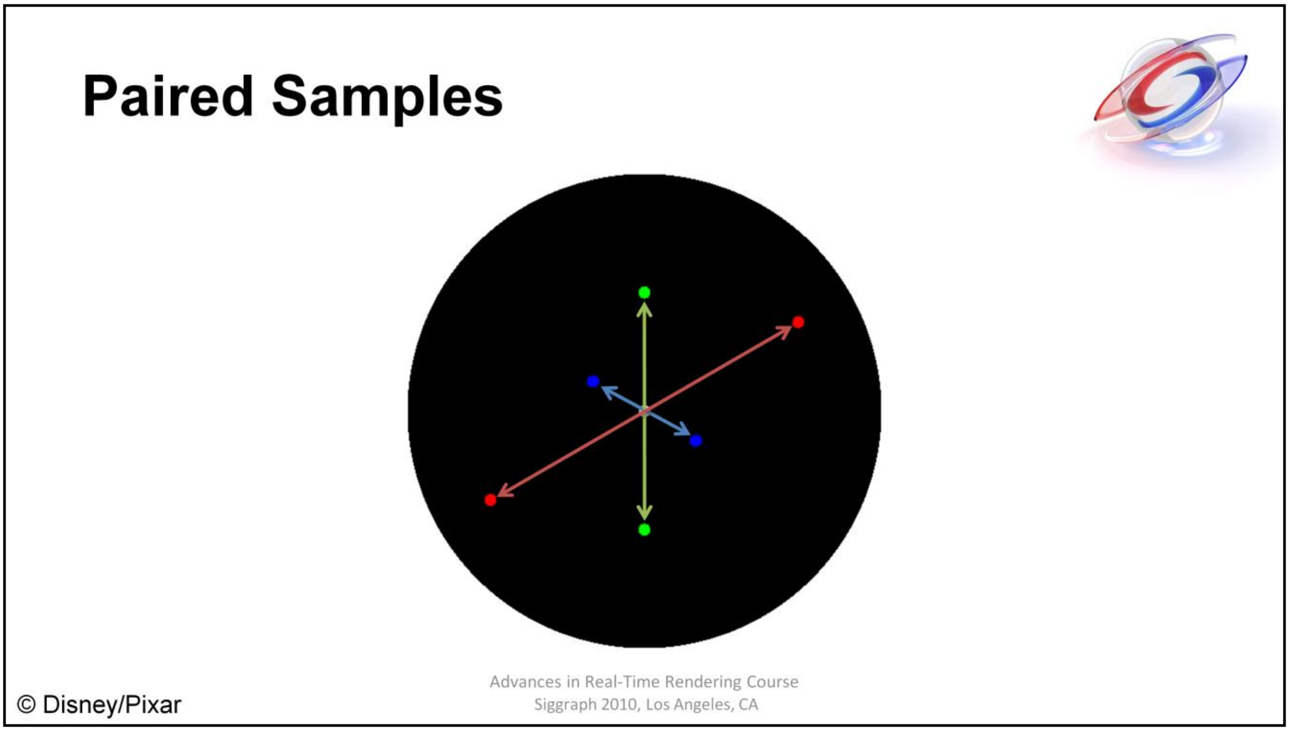


Paired Samples

© Disney/Pixar

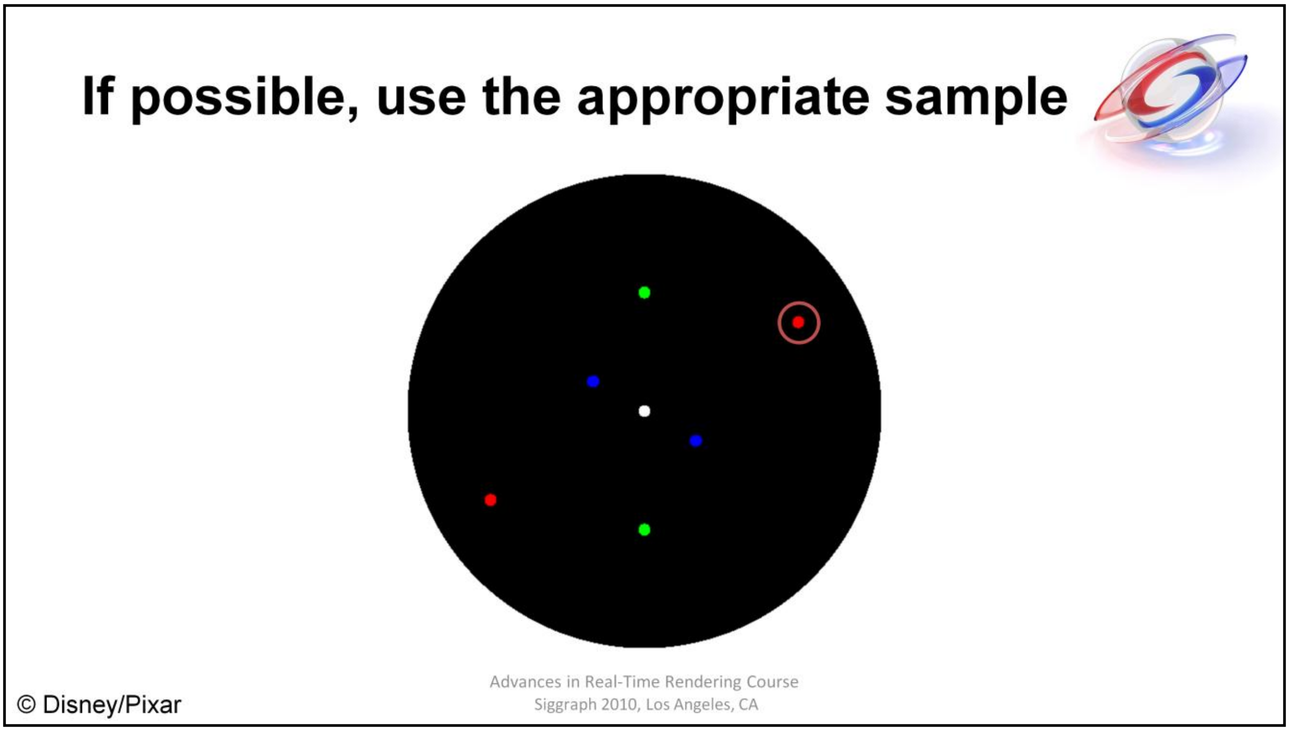
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

In order to be able to estimate missing samples in this way, we need to put a constraint on our sampling pattern that every sample is part of a pair. This finally explains the strange “Orion” sampling pattern seen previously ☺



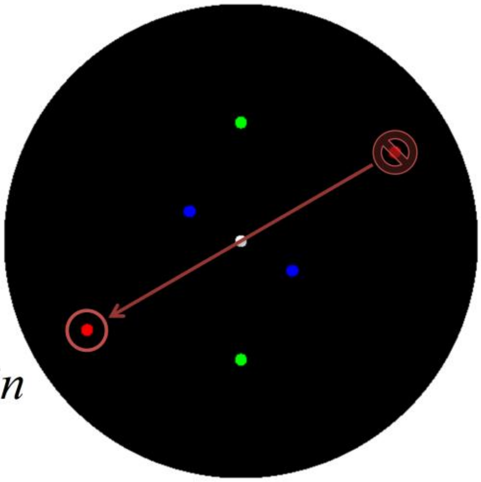
Paired Samples

© Disney/Pixar



If possible, use the appropriate sample

Otherwise, if possible, use its twin's inverse

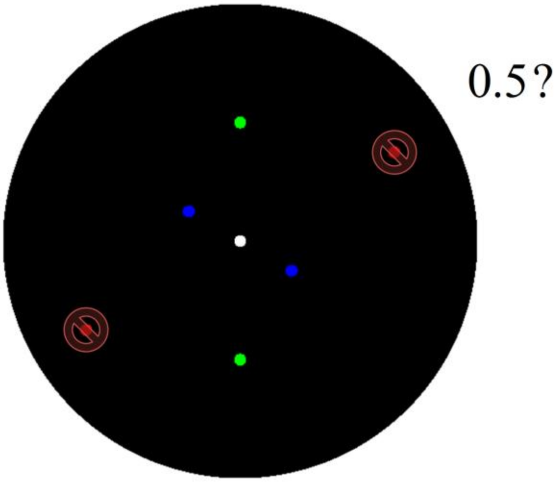


$1 - pairedTwin$

© Disney/Pixar

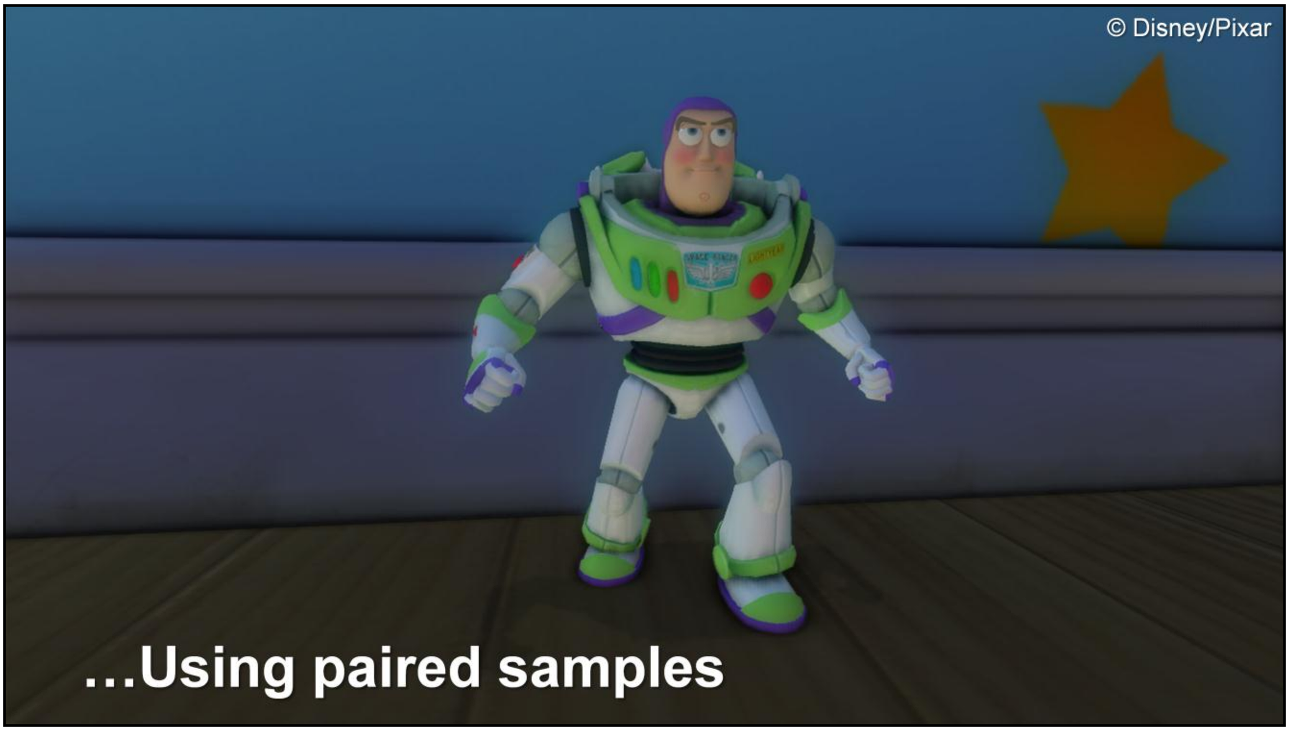
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Otherwise, use 0.5

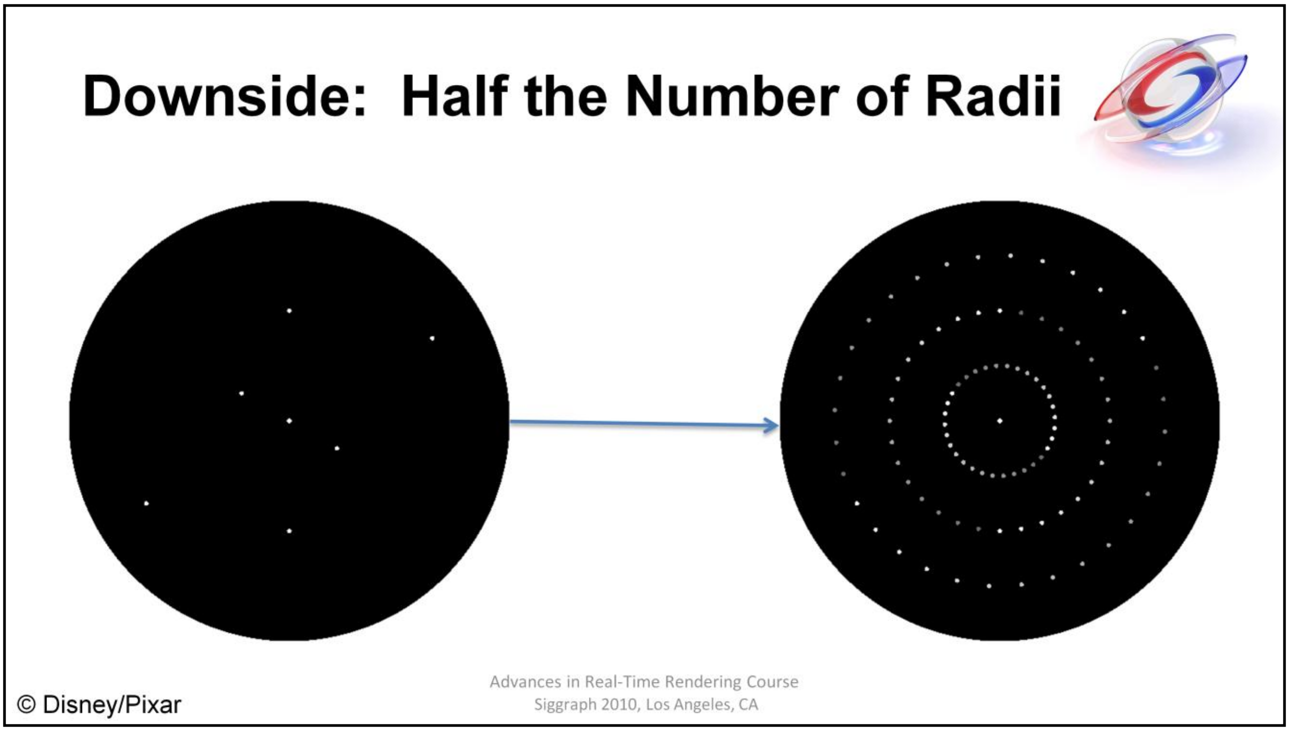


© Disney/Pixar

Still discarding unusable samples…



In some ways this doesn’t look as good as the previous tries, but it definitely leads to the most consistently inoffensive artifacts of the methods presented here.

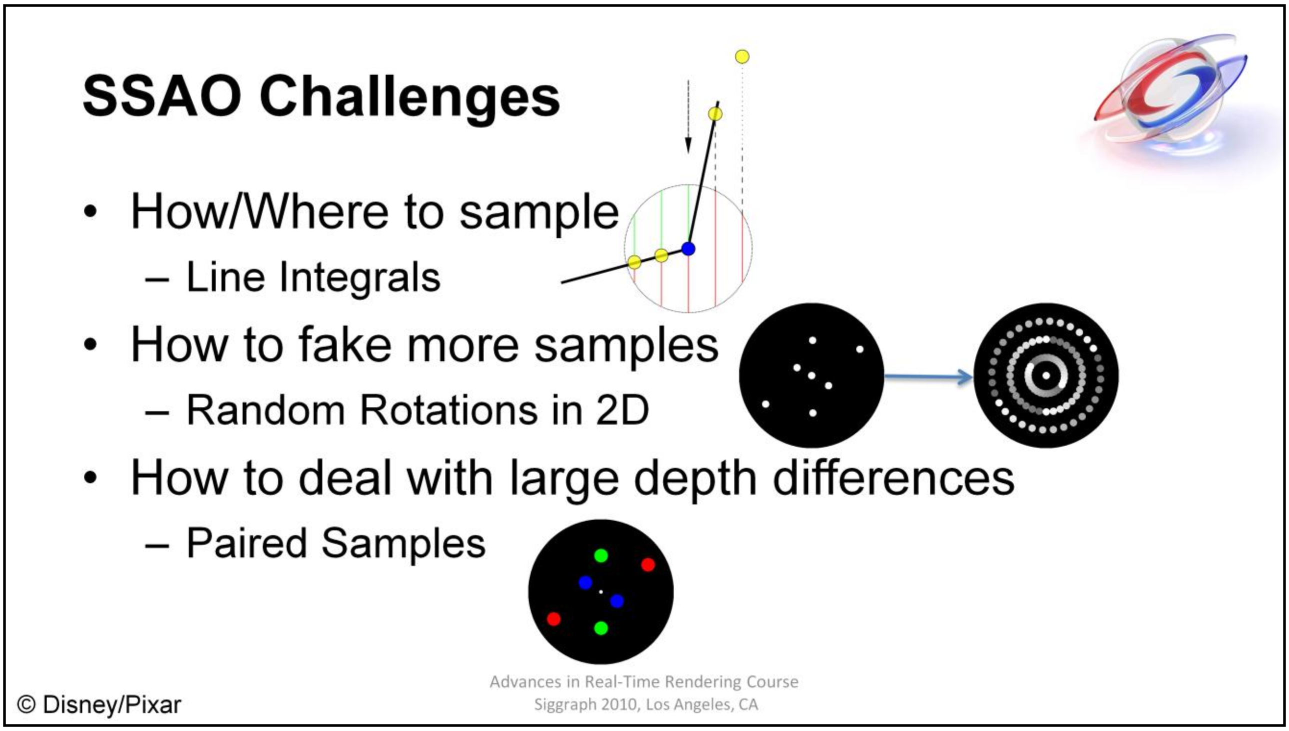


Downside: Half the Number of Radii

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

One unfortunate consequence of using paired samples is you don’t get as much radial coverage as you otherwise would. In our specific case, we only get three “rings” even though we are taking six samples per-pixel.



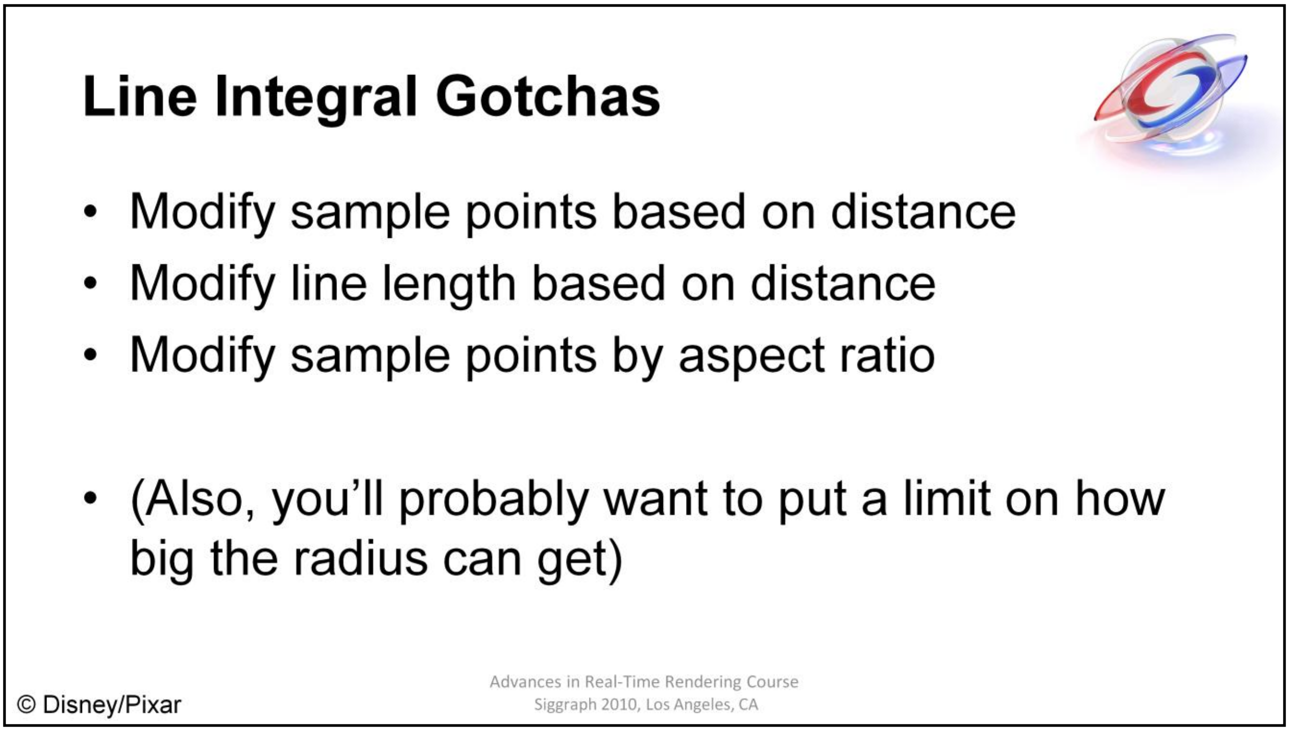
SSAO Challenges

- How/Where to sample
  - Line Integrals
- How to fake more samples
  - Random Rotations in 2D
- How to deal with large depth differences
  - Paired Samples

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Review

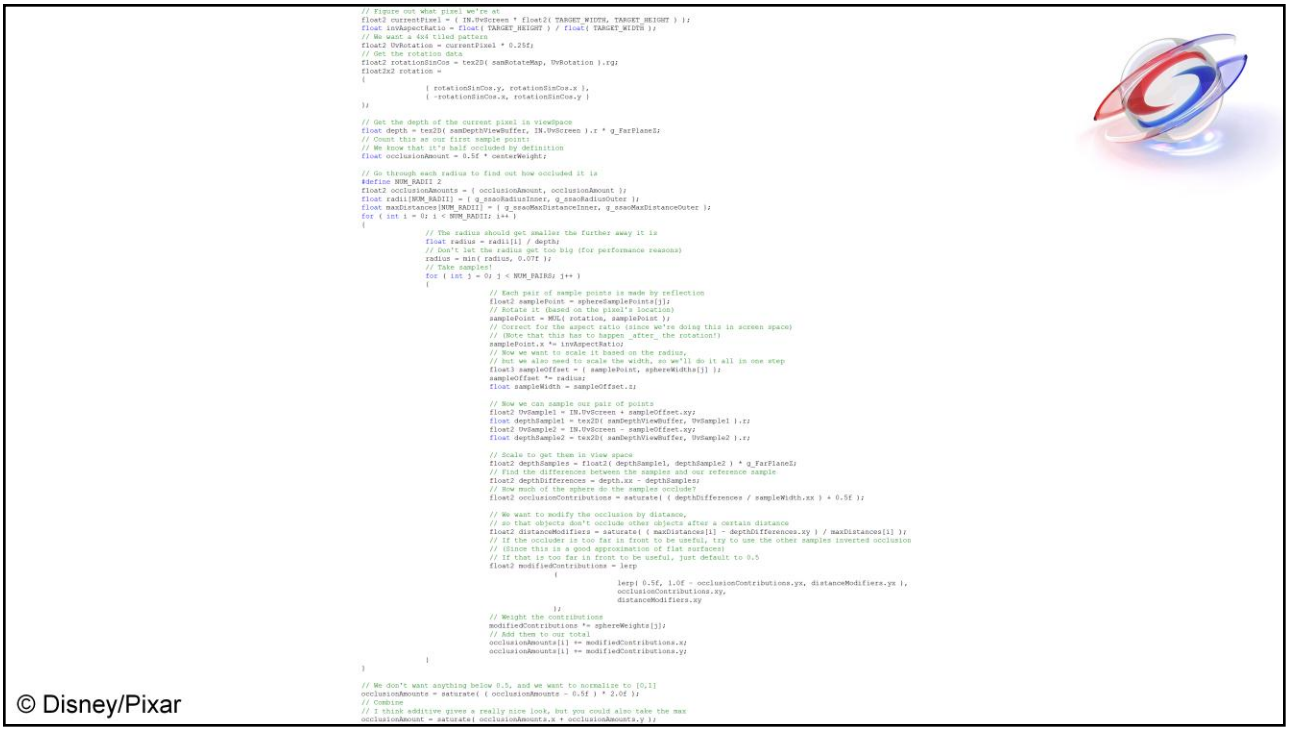


Line Integral Gotchas

- Modify sample points based on distance
- Modify line length based on distance
- Modify sample points by aspect ratio

- (Also, you’ll probably want to put a limit on how big the radius can get)

There are a few gotchas that gave me quite a bit of trouble while implementing this…
Make sure to read this slide if you’re considering it! ☺



```
// Figure out what pixel we're at
float2 currentPixel = ( IN.UvScreen * float2( TARGET_WIDTH, TARGET_HEIGHT ) );
float invAspectRatio = float( TARGET_HEIGHT ) / float( TARGET_WIDTH );
// We want a 4x4 tiled pattern
float2 UvRotation = currentPixel * 0.25f;
// Get the rotation data
float2 rotationSinCos = tex2D( samRotateMap, UvRotation ).rg;
float2x2 rotation =
{
        { rotationSinCos.y, rotationSinCos.x },
        { -rotationSinCos.x, rotationSinCos.y }
};

// Get the depth of the current pixel in viewSpace
float depth = tex2D( samDepthViewBuffer, IN.UvScreen ).r * g_FarPlaneZ;
// Count this as our first sample point:
// We know that it's half occluded by definition
float occlusionAmount = 0.5f * centerWeight;

// Go through each radius to find out how occluded it is
#define NUM_RADII 2
float2 occlusionAmounts = { occlusionAmount, occlusionAmount };
float radii[NUM_RADII] = { g_ssaoRadiusInner, g_ssaoRadiusOuter };
float maxDistances[NUM_RADII] = { g_ssaoMaxDistanceInner, g_ssaoMaxDistanceOuter };
for ( int i = 0; i < NUM_RADII; i++ )
{
        // The radius should get smaller the further away it is
        float radius = radii[i] / depth;
        // Don't let the radius get too big (for performance reasons)
        radius = min( radius, 0.07f );
        // Take samples!
        for ( int j = 0; j < NUM_PAIRS; j++ )
        {
                // Each pair of sample points is made by reflection
                float2 samplePoint = sphereSamplePoints[j];
                // Rotate it (based on the pixel's location)
                samplePoint = MUL( rotation, samplePoint );
                // Correct for the aspect ratio (since we're doing this in screen space)
                // (Note that this has to happen _after_ the rotation!)
                samplePoint.x *= invAspectRatio;
                // Now we want to scale it based on the radius,
                // but we also need to scale the width, so we'll do it all in one step
                float3 sampleOffset = ( samplePoint, sphereWidths[j] );
                sampleOffset *= radius;
                float sampleWidth = sampleOffset.z;

                // Now we can sample our pair of points
                float2 UvSample1 = IN.UvScreen + sampleOffset.xy;
                float depthSample1 = tex2D( samDepthViewBuffer, UvSample1 ).r;
                float2 UvSample2 = IN.UvScreen - sampleOffset.xy;
                float depthSample2 = tex2D( samDepthViewBuffer, UvSample2 ).r;

                // Scale to get them in view space
                float2 depthSamples = float2( depthSample1, depthSample2 ) * g_FarPlaneZ;
                // Find the differences between the samples and our reference sample
                float2 depthDifferences = depth.xx - depthSamples;
                // How much of the sphere do the samples occlude?
                float2 occlusionContributions = saturate( ( depthDifferences / sampleWidth.xx ) + 0.5f );

                // We want to modify the occlusion by distance,
                // so that objects don't occlude other objects after a certain distance
                float2 distanceModifiers = saturate( ( maxDistances[i] - depthDifferences.xy ) / maxDistances[i] );
                // If the occluder is too far in front to be useful, try to use the other samples inverted occlusion
                // (Since this is a good approximation of flat surfaces)
                // If that is too far in front to be useful, just default to 0.5
                float2 modifiedContributions = lerp
                        (
                                lerp( 0.5f, 1.0f - occlusionContributions.yx, distanceModifiers.yx ),
                                occlusionContributions.xy,
                                distanceModifiers.xy
                        );
                // Weight the contributions
                modifiedContributions *= sphereWeights[j];
                // Add them to our total
                occlusionAmounts[i] += modifiedContributions.x;
                occlusionAmounts[i] += modifiedContributions.y;
        }
}

// We don't want anything below 0.5, and we want to normalize to [0,1]
occlusionAmounts = saturate( ( occlusionAmounts - 0.5f ) * 2.0f );
// Combine
// I think additive gives a really nice look, but you could also take the max
occlusionAmount = saturate( occlusionAmounts.x + occlusionAmounts.y );
```

© Disney/Pixar

I’m not including any of our actual sample pattern data, but here is the basic shader algorithm we used.

Again, if you are interested in implementing SSAO with line integrals I would strongly recommend reading the paper.

Chris Hall

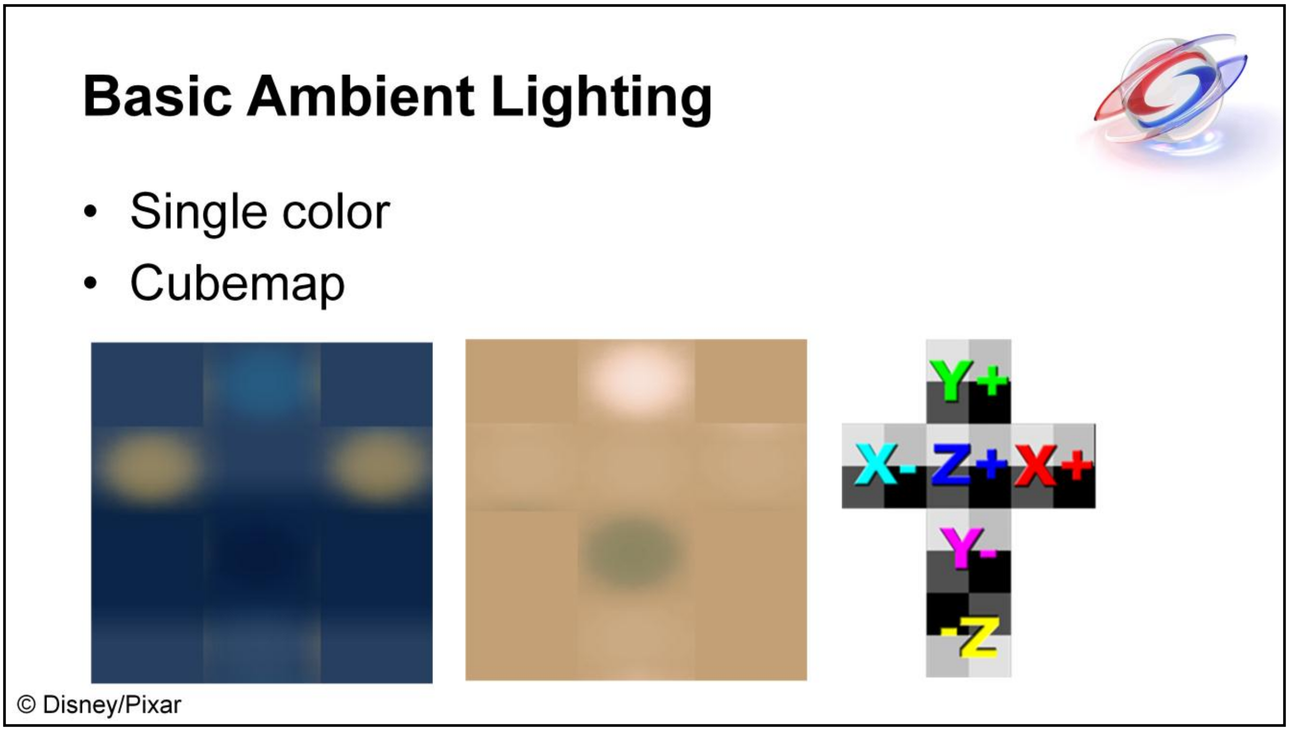
christopher.hall@disney.com

AMBIENT LIGHTING

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Chris Hall

AMBIENT LIGHTING



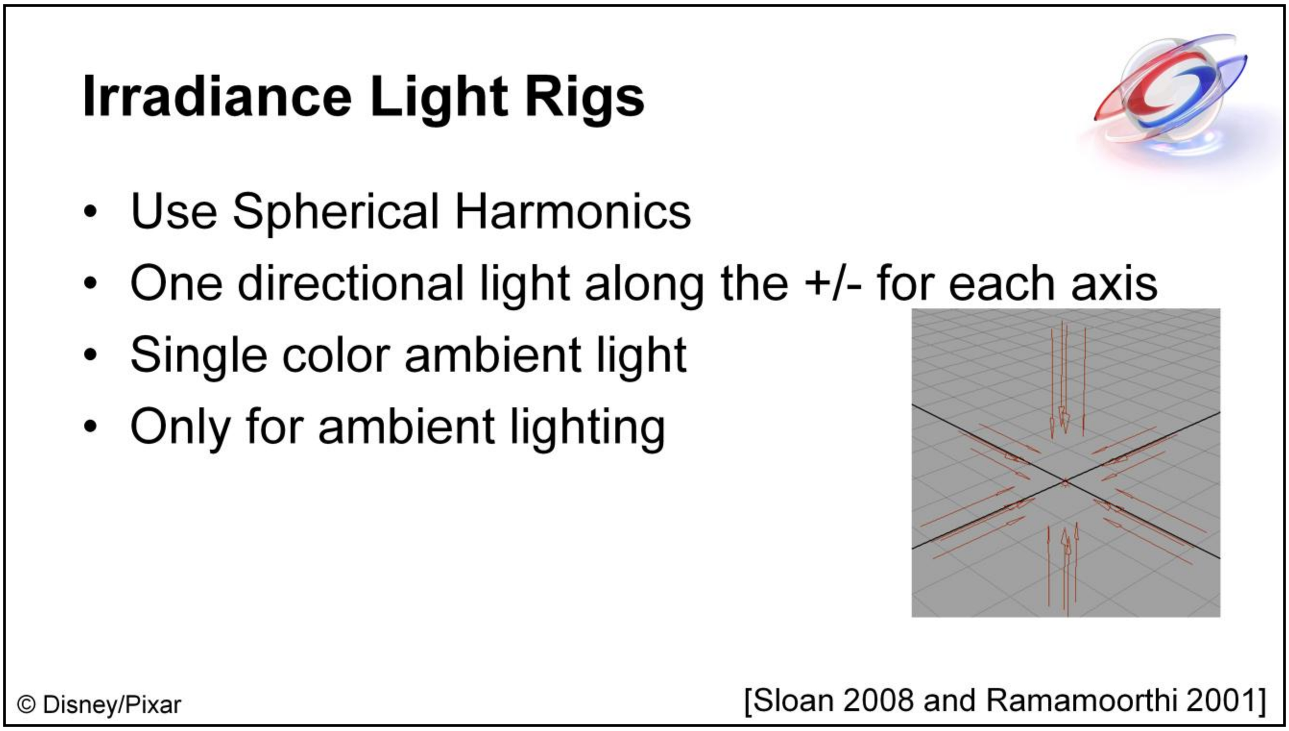
Basic Ambient Lighting

- Single color
- Cubemap

In Toy Story 3, all light was dynamic. This includes the ambient light. First off, I want to cover two basic approaches to dynamic ambient lighting.

Single color
Most basic model for ambient lighting. Works but really bad in shadow because there is no directionality to it. The solid color makes everything look flat.

Cubemap
This gave us an improved look. The lighting was no longer flat, and normal maps would show up in shadow. This is what was used in Bolt. But since it was a texture, it wasn't live tweakable, and the artists had a hard time painting the cubemap.



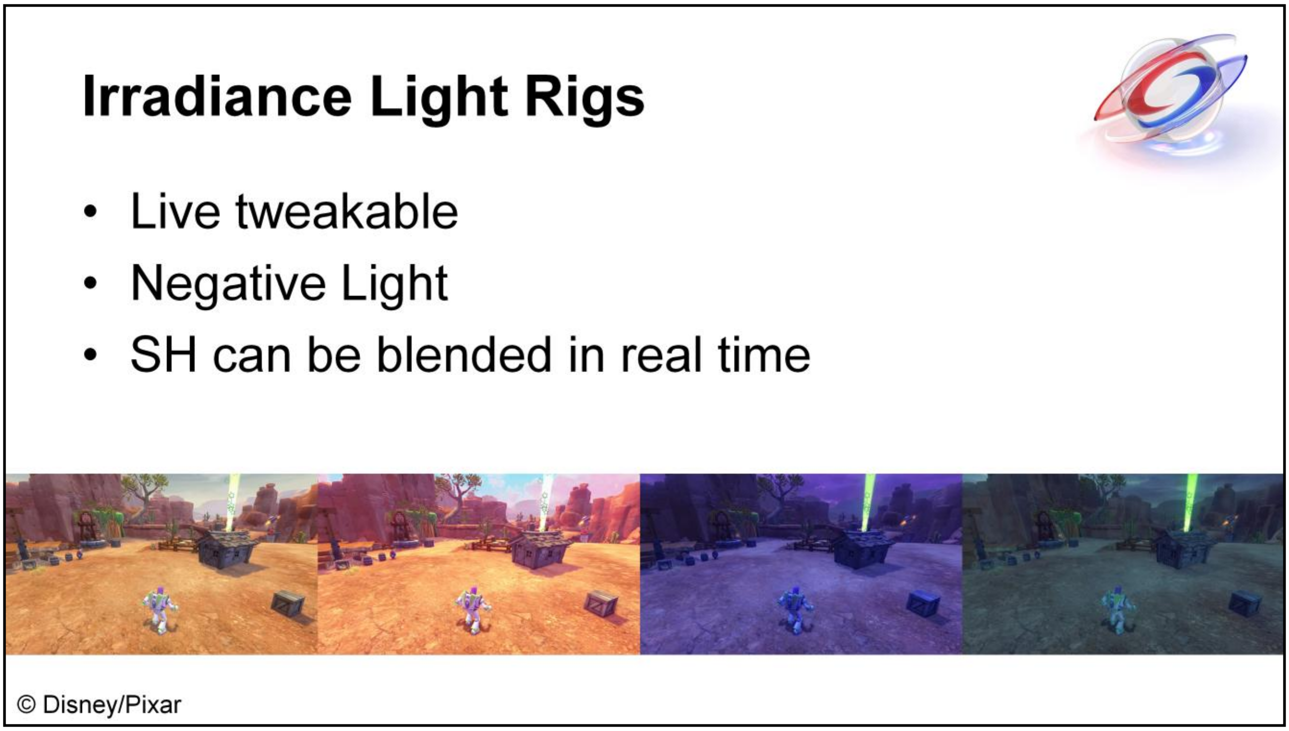
Irradiance Light Rigs

- Use Spherical Harmonics
- One directional light along the +/- for each axis
- Single color ambient light
- Only for ambient lighting

[Sloan 2008 and Ramamoorthi 2001]

Irradiance Light Rigs

Basic approach used. We just set up a light rig along each axis and bake each of the lights into a set of spherical harmonic coefficients. In the shader, I'd evaluate the spherical harmonic coefficients with the pixel normal to get the ambient light color. Even though this setup could be used for diffuse lighting, in this case, it was only used for the ambient lighting.



Irradiance Light Rigs

- Live tweakable
- Negative Light
- SH can be blended in real time

© Disney/Pixar

Live Tweakable

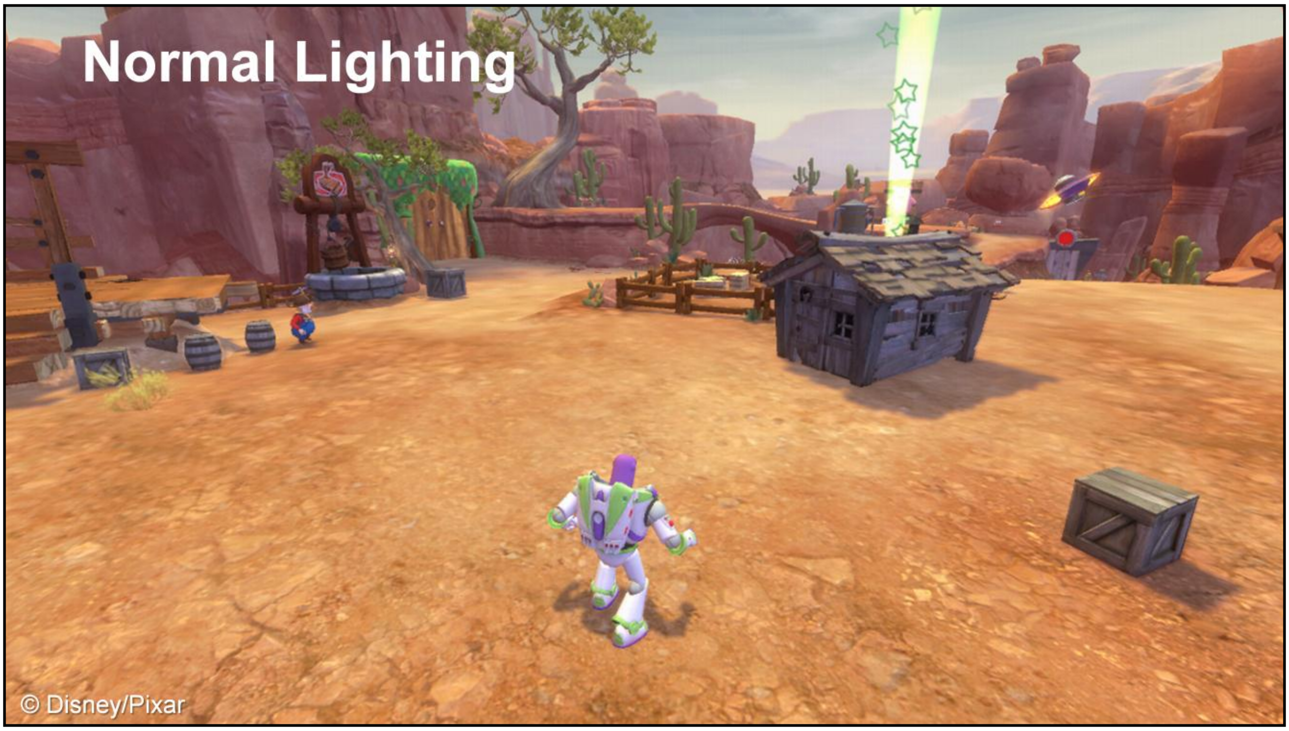
Because the calculations were quick, we could recalculate the SH coefficients in real time. This meant that artists could instantly see the changes they made to the light rig. Almost immediately after the artists started using the light rigs, our ambient lighting improved dramatically. Being able to see the changes instantaneously provided feedback that let them tweak it until it looked fantastic.

Negative Light

Some artists used negative light intensities, mainly on the light pointing up from the negative y direction. This darkened the bottom of the lighting and gave everything a slight shadow. This was something I didn’t plan for but ended up being used on quite a few levels.

Blend in real time

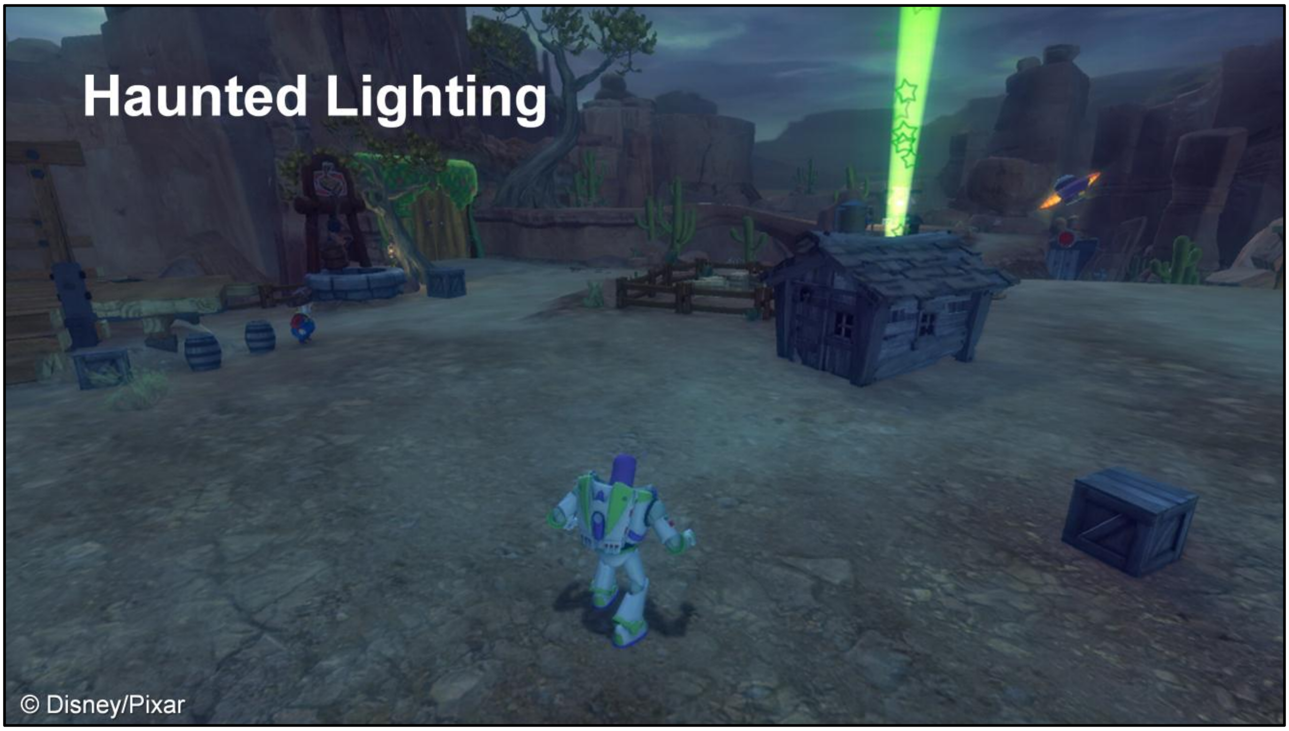
We could blend between different light rigs instantaneously without any additional shader cost. This would make it easy to gradually change the ambient lighting or blend between more than one rig.



Pictures showing 4 different types of lighting used. The lighting would switch depending on the gameplay, plus the player could change them at will.

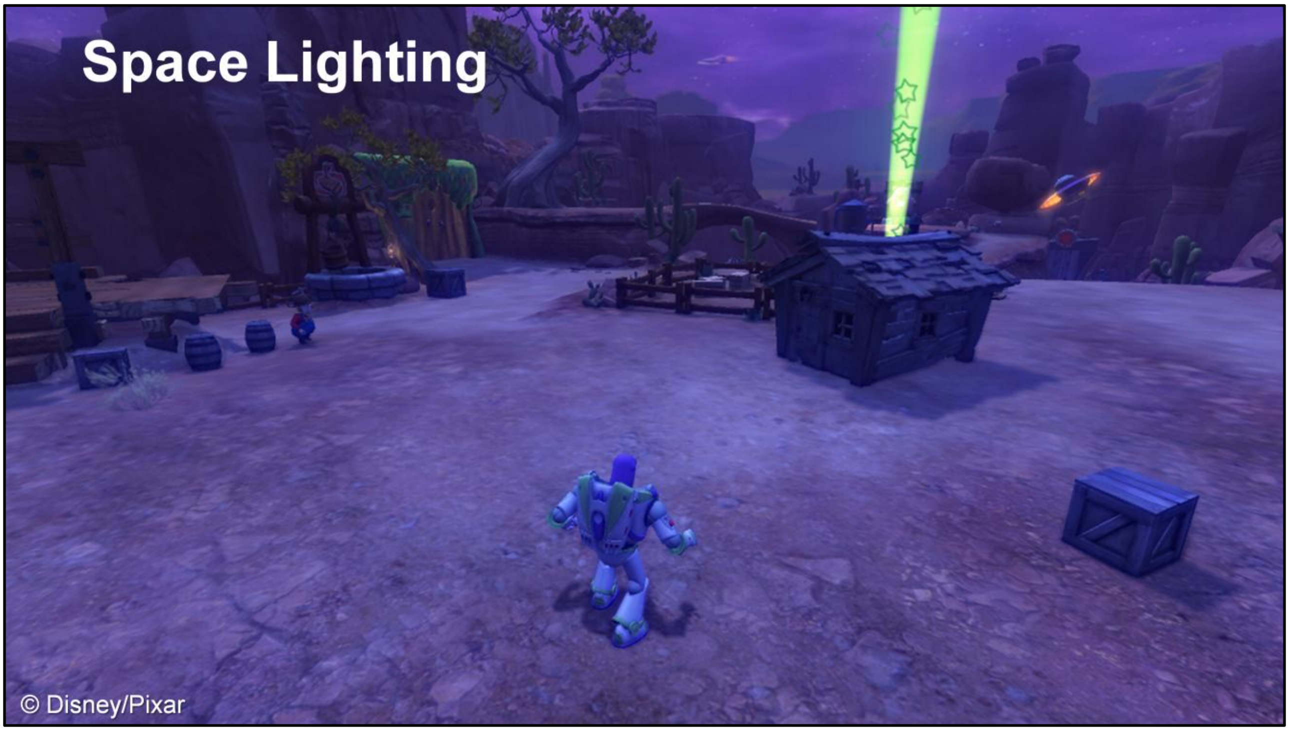
Enchanted Lighting

© Disney/Pixar



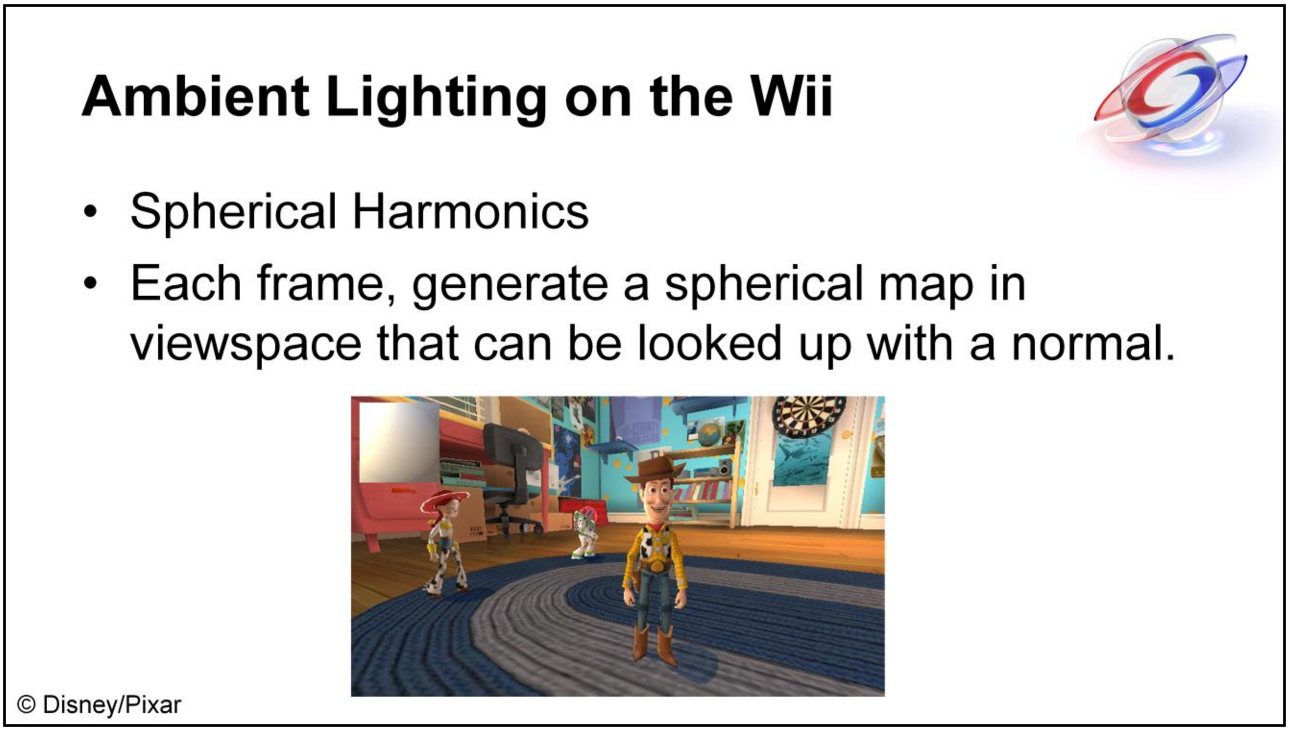
Haunted Lighting

© Disney/Pixar



Space Lighting

© Disney/Pixar

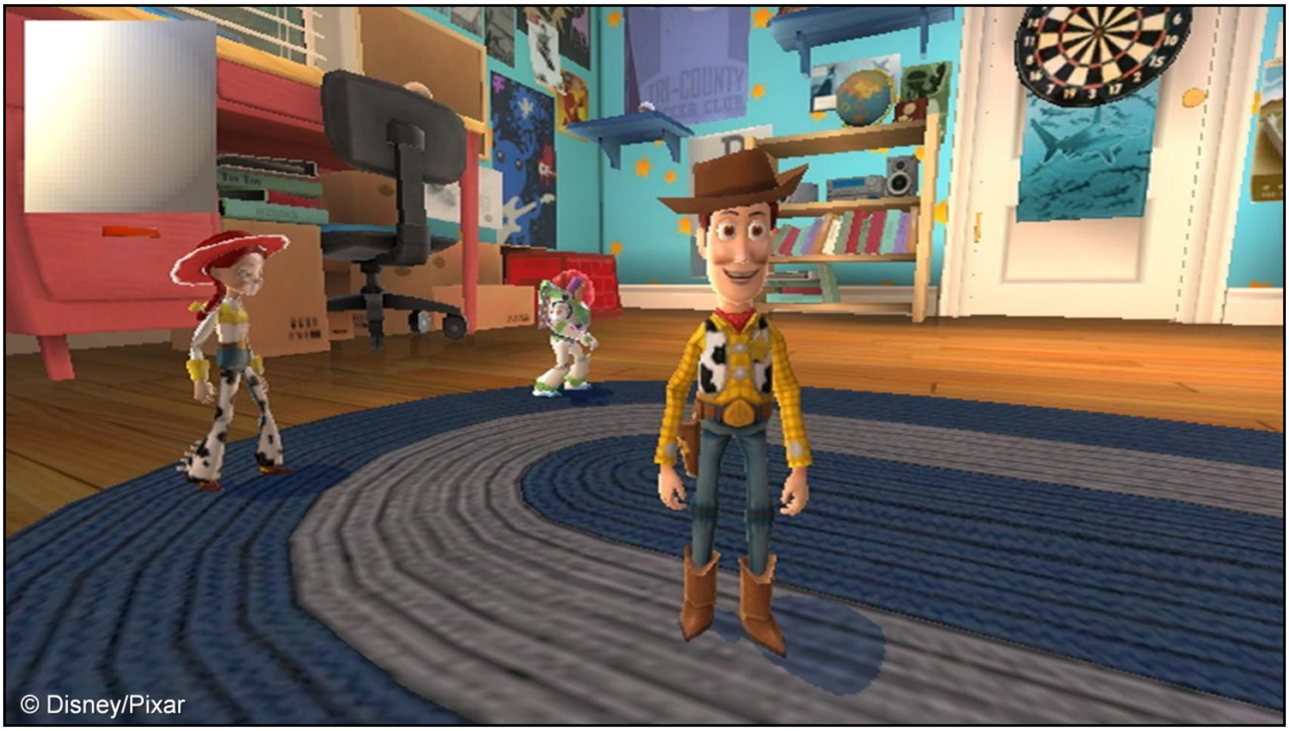


Ambient Lighting on the Wii

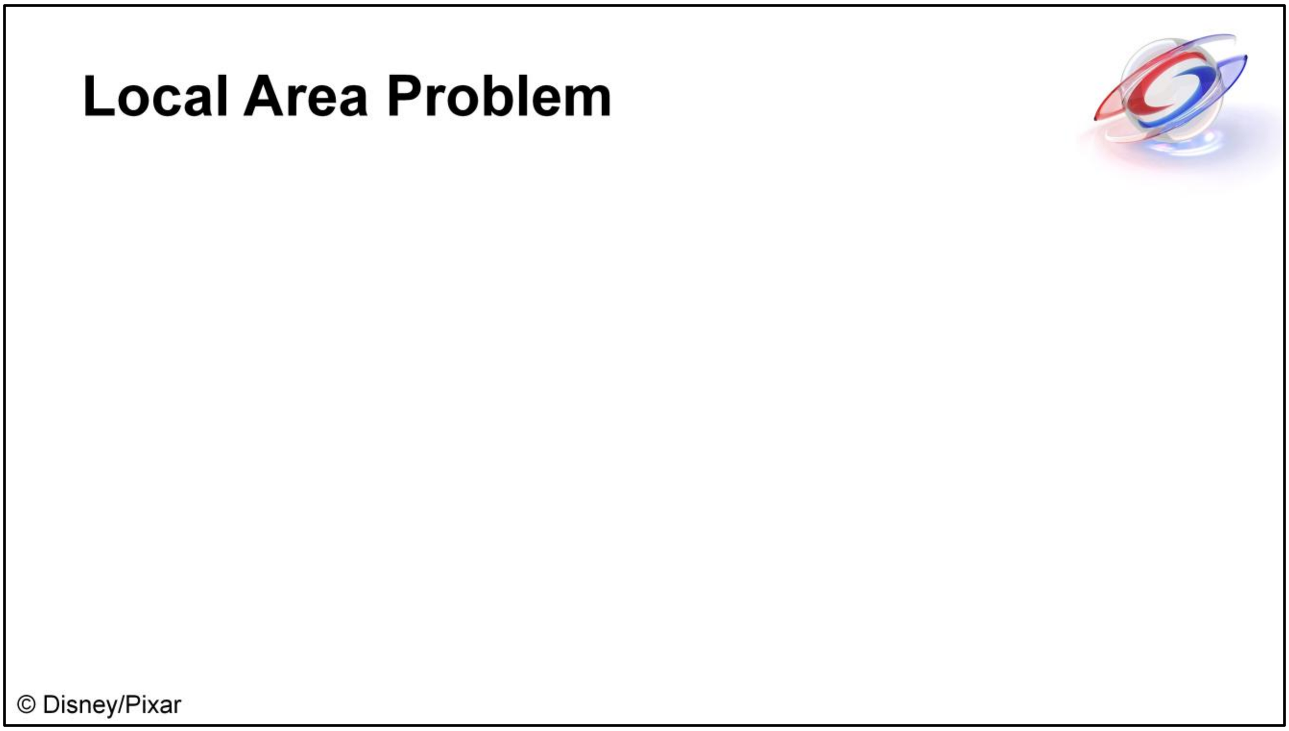
- Spherical Harmonics
- Each frame, generate a spherical map in viewspace that can be looked up with a normal.

© Disney/Pixar

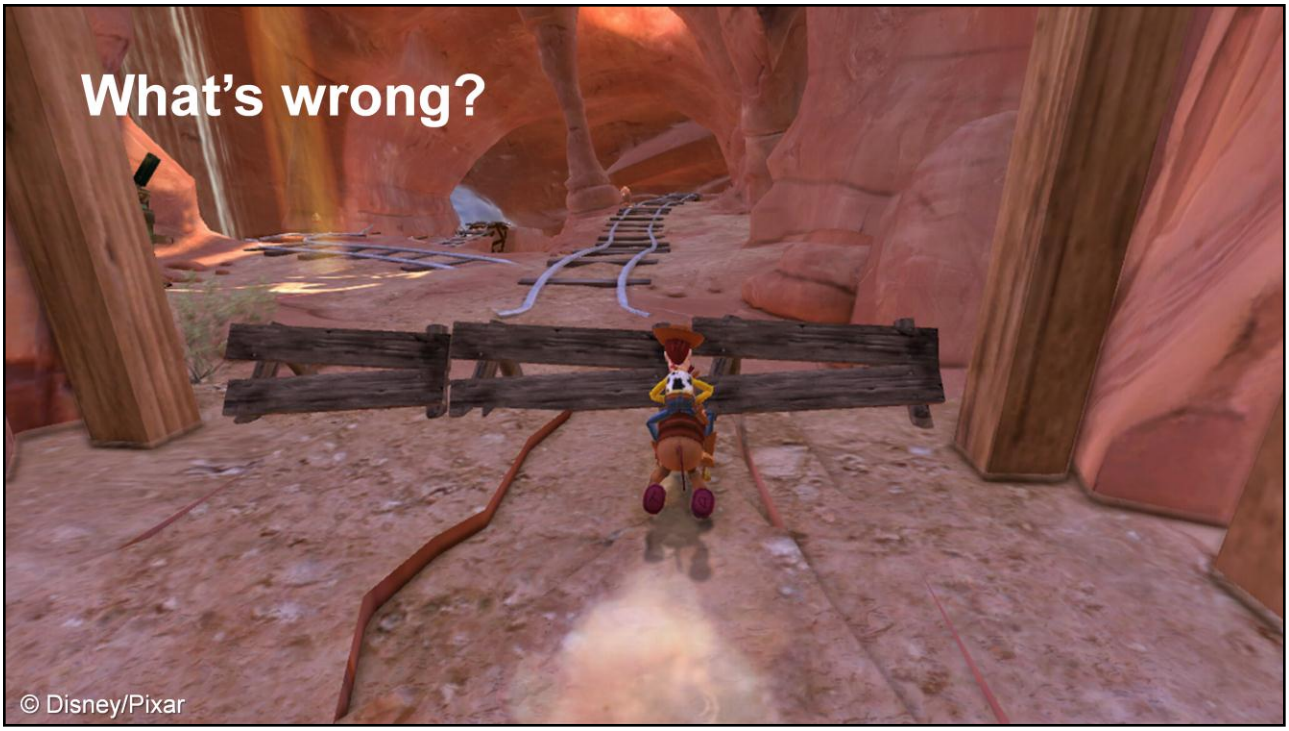
We not only use spherical harmonics for ambient light on the “next-gen” platforms, but also on the Wii. Aside from the obvious advantages over a flat ambient color, this also allows us to have fairly similar lighting setups on all platforms. Every frame, we transform the lights into view space, and render a spherical texture that can be used as a lookup. This has a few visual artifacts at steep angles at the edge of the screen, but in general it looks quite good.



Full screen view of the last picture

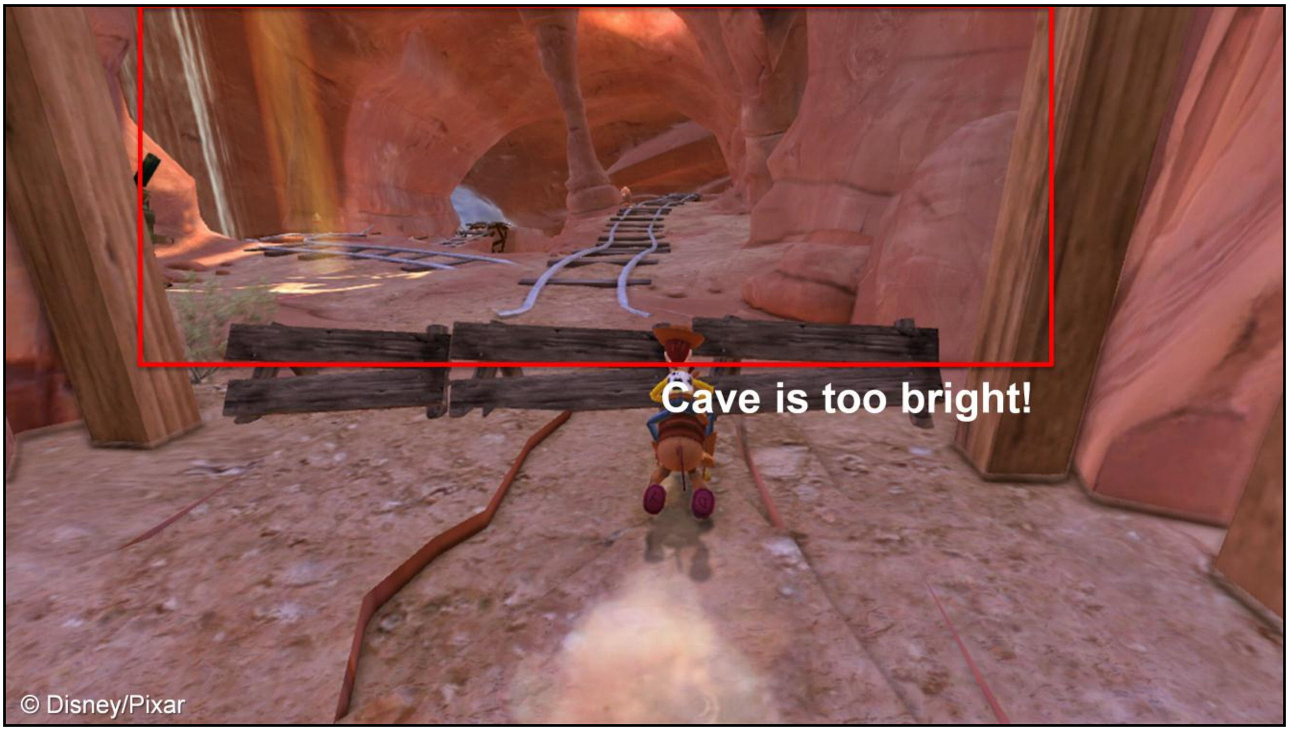


Local Area Problem

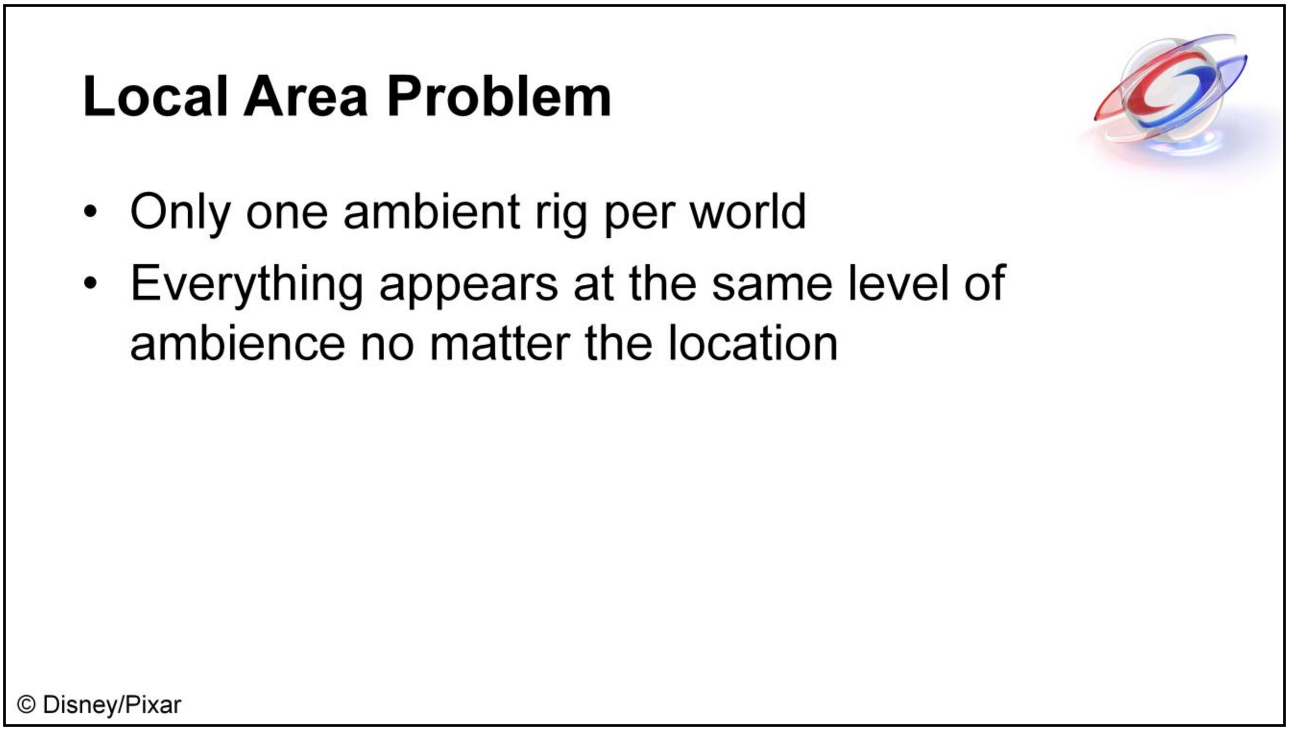


What's wrong?

© Disney/Pixar



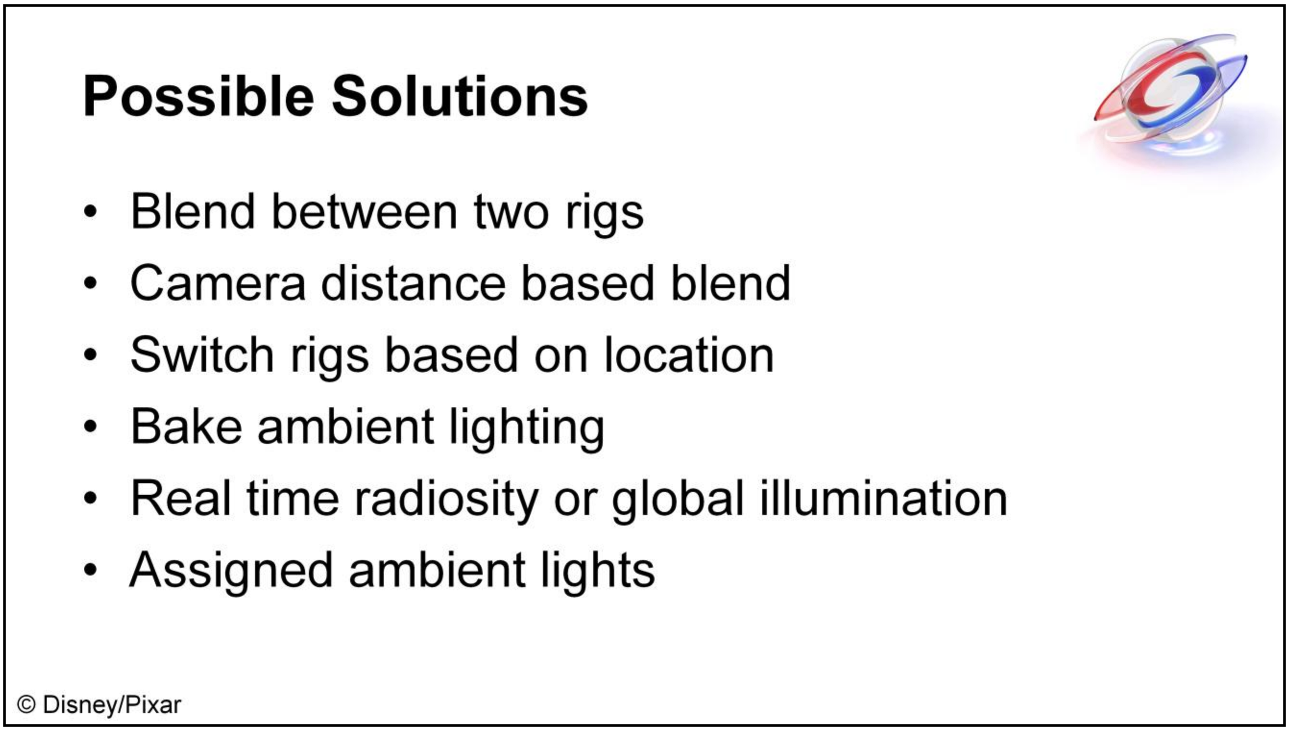
Cave is too bright!



Local Area Problem

- Only one ambient rig per world
- Everything appears at the same level of ambience no matter the location

The artists were fine with having one ambient rig per world, but they quickly noticed the limitations of it. Since the lighting is the same throughout the world, this means that a lot of places look too bright. This was especially noticeable indoors. Therefore, there was a big push to do something about these situations.



Possible Solutions

- Blend between two rigs
- Camera distance based blend
- Switch rigs based on location
- Bake ambient lighting
- Real time radiosity or global illumination
- Assigned ambient lights

Here is a list of some of my initial solutions to the problem. The first three were quick hacks that didn’t accomplish much.

Blend between two rigs

Doing this only changes the global blend which means everything changes at the same time.

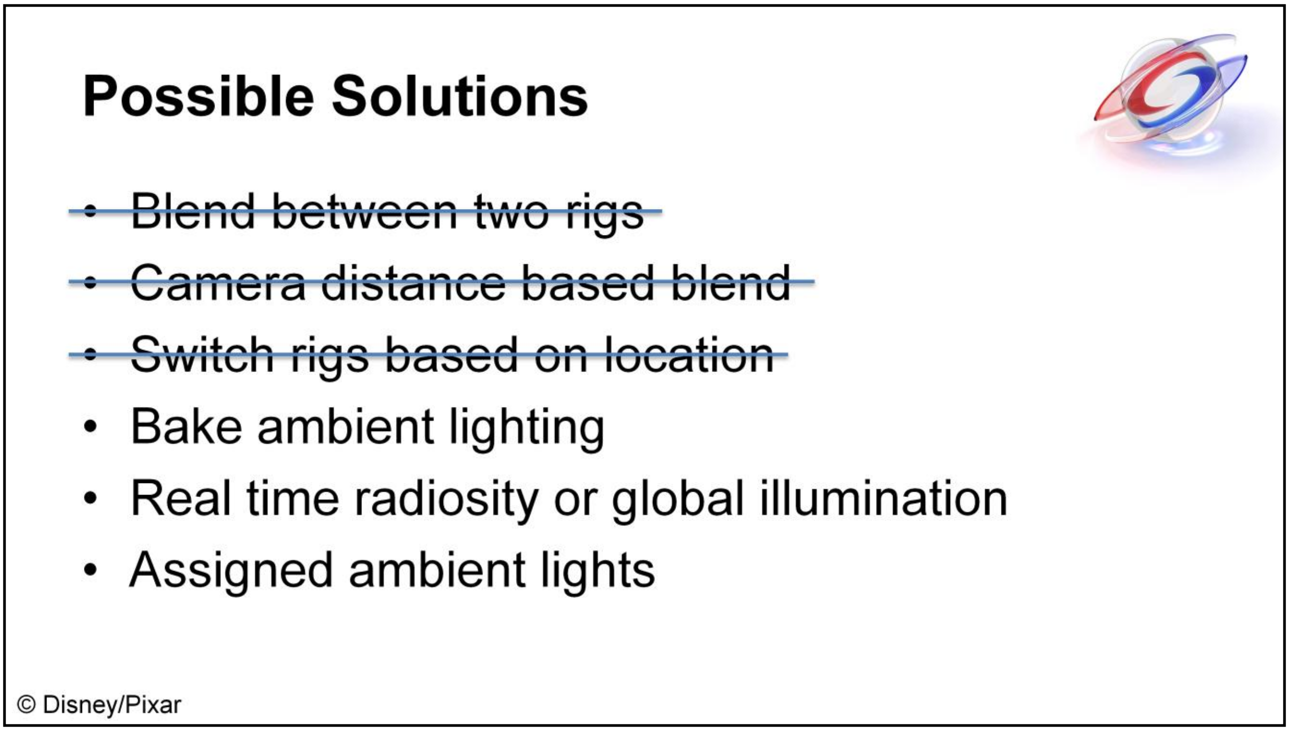
Camera distance based blend

Blend the ambience to other rigs depending on where the camera is and how far it is away from certain areas.

Same problem with blending between two rigs. It is global not localized.

Switch rigs based on location

We actually did some of this at the start. When you go to a dark area, pop it to a darker ambience. It was easy to see the pop and it affected everything. This was a good step in the right direction but not ideal.



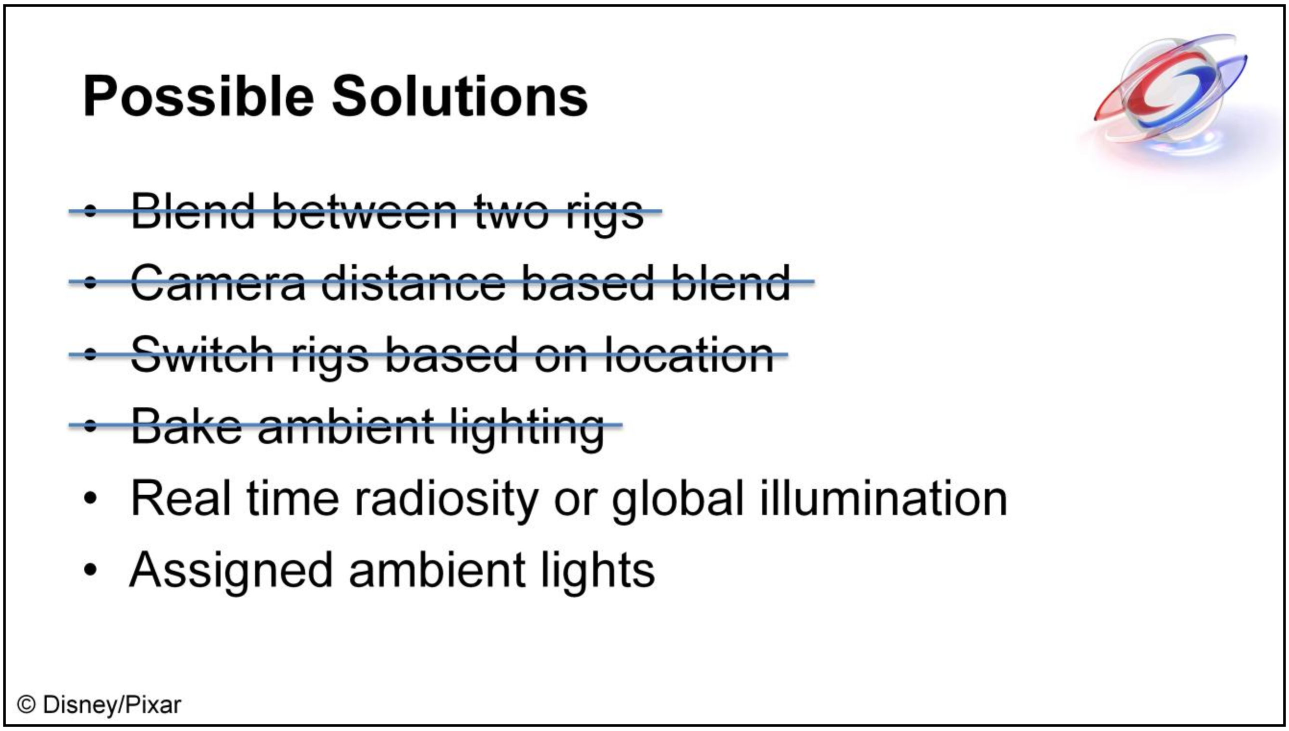
Possible Solutions

- ~~Blend between two rigs~~
- ~~Camera distance based blend~~
- ~~Switch rigs based on location~~
- Bake ambient lighting
- Real time radiosity or global illumination
- Assigned ambient lights

© Disney/Pixar

Bake ambient lighting

Since our lighting is all dynamic and artist controlled, we couldn't bake ambient light at all. We had no pipeline to do it too.

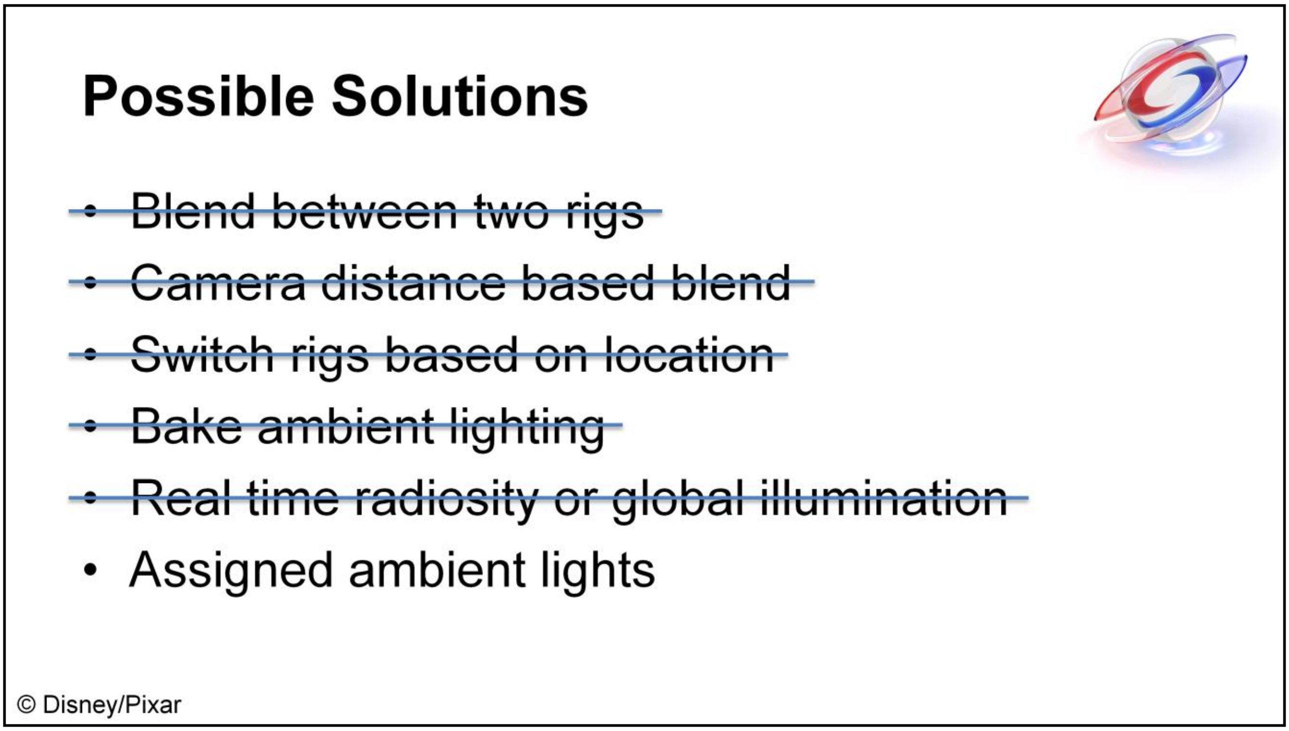


Possible Solutions

- ~~Blend between two rigs~~
- ~~Camera distance based blend~~
- ~~Switch rigs based on location~~
- ~~Bake ambient lighting~~
- Real time radiosity or global illumination
- Assigned ambient lights

Real time radiosity/global illumination

This could have solved the problem but we didn’t have the performance budget to allow it neither did we have the time to spend developing a solution.

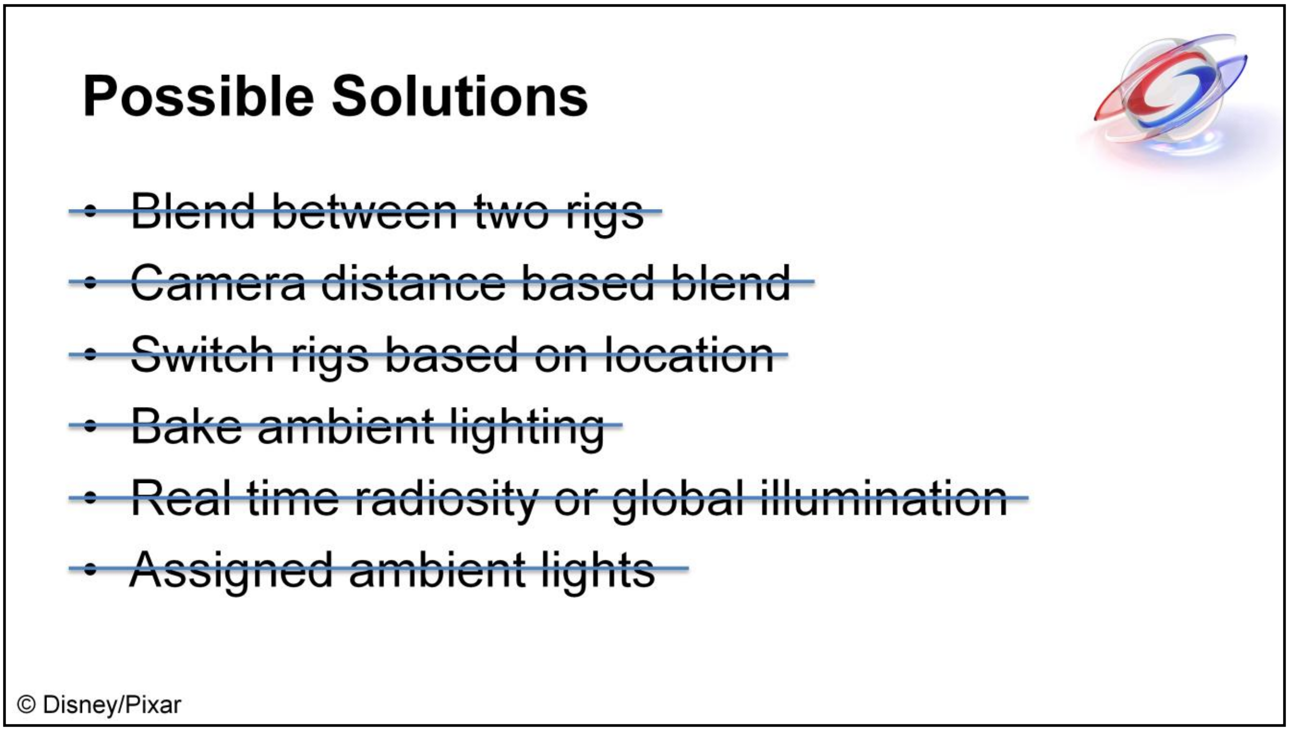


Possible Solutions

- ~~Blend between two rigs~~
- ~~Camera distance based blend~~
- ~~Switch rigs based on location~~
- ~~Bake ambient lighting~~
- ~~Real time radiosity or global illumination~~
- Assigned ambient lights

Assigned ambient lights

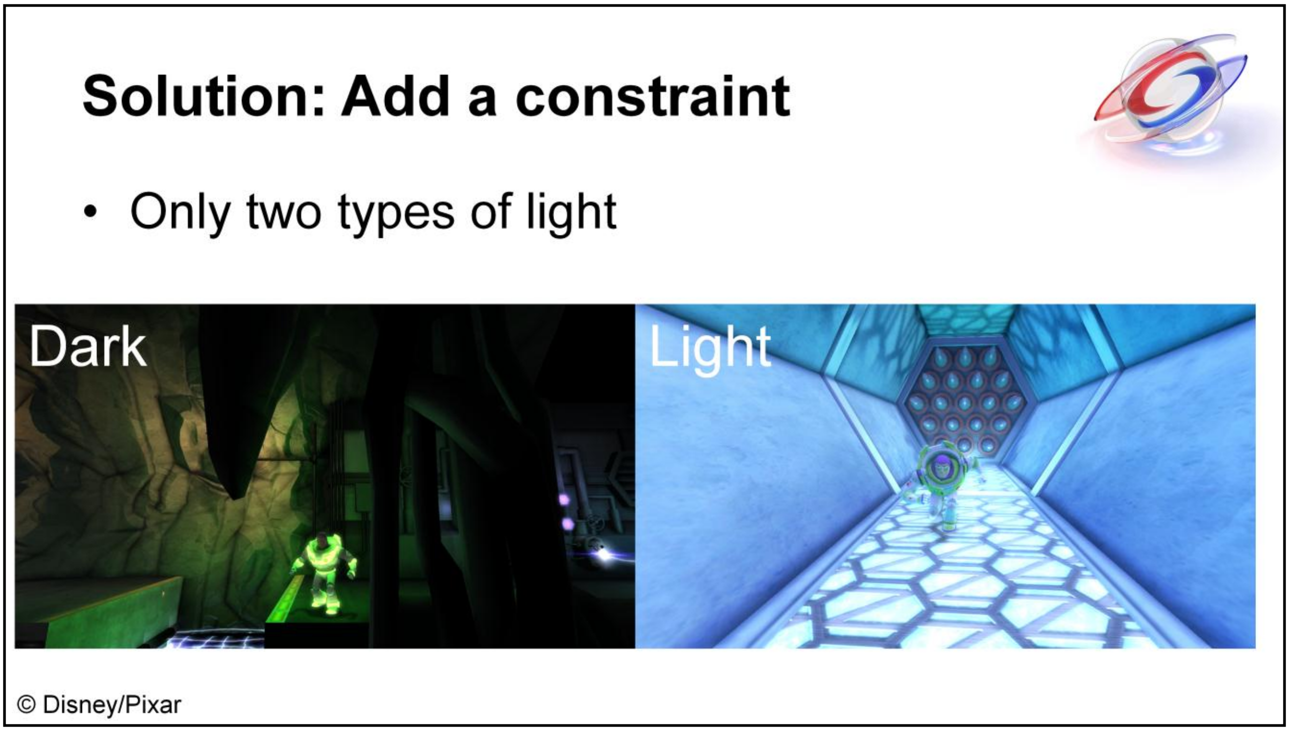
I tried this out. You split up meshes based on where the ambient lights are located and assign a closest ambient light to each mesh. Then you just blend from the global ambient lighting to the assigned ambient light based on for far the pixel is away. Doing this would require splitting meshes and there were seam issues when there was any amount of overlap. We already did it for our direct lighting so doing even more splitting wasn't worth the trouble nor the extra lighting cost.



Possible Solutions

- ~~Blend between two rigs~~
- ~~Camera distance based blend~~
- ~~Switch rigs based on location~~
- ~~Bake ambient lighting~~
- ~~Real time radiosity or global illumination~~
- ~~Assigned ambient lights~~

So what can we do?

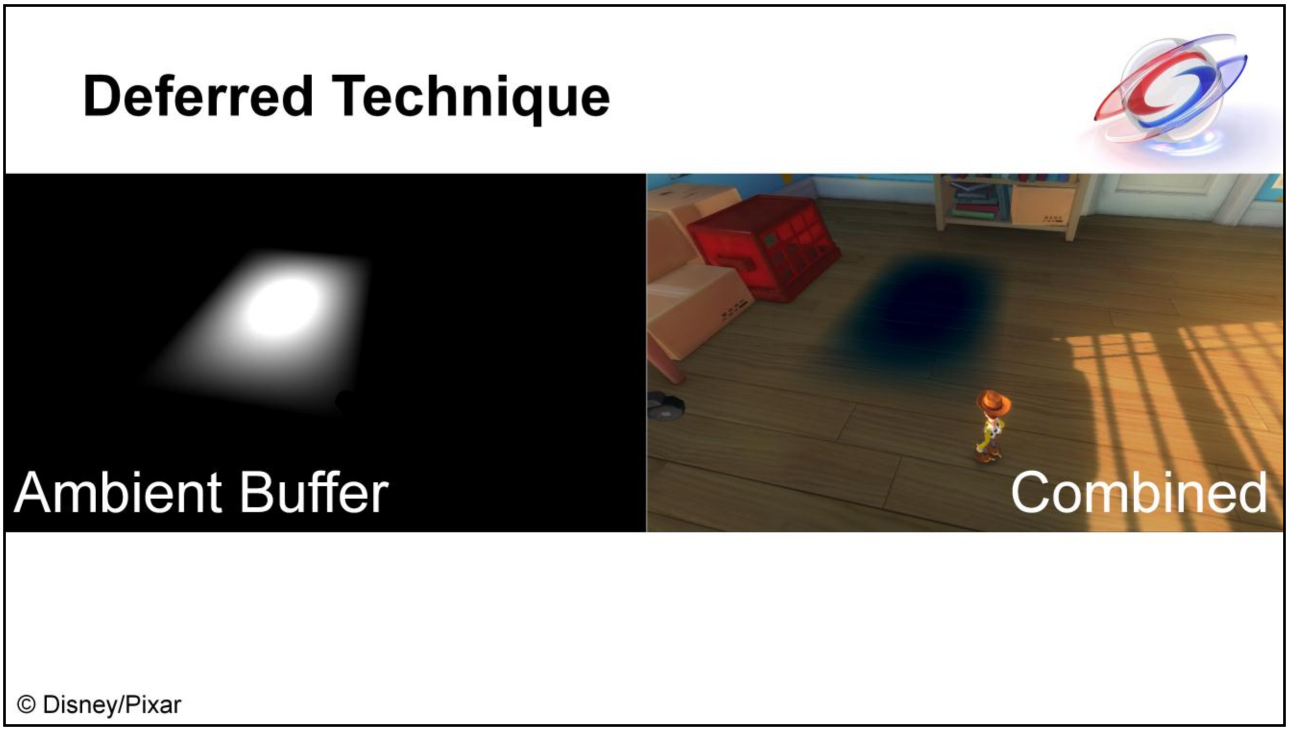


Solution: Add a constraint

- Only two types of light

© Disney/Pixar

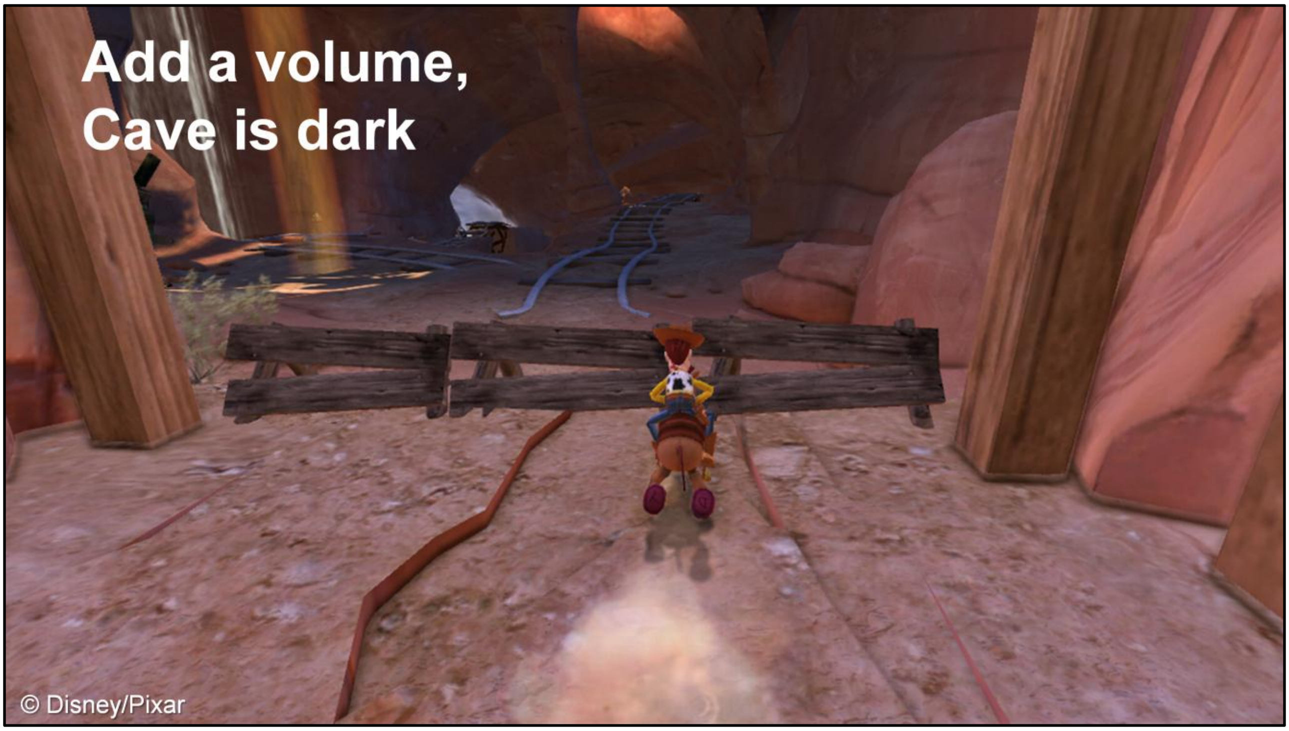
The experiments using traditional lighting failed to give me the results I wanted. After I switched the ambient lighting to use spherical harmonics, I came back to the problem and starting looking into ways to make it easier to solve. The key to this was to assume that we only had two types of ambient light instead of one global one. This second type was mainly to provide darkness or an alternative look that still matched the same sunlight lighting. Since the spherical harmonics coefficients could be blended, all I had to figure out was how to do the blend. This constraint ended up being easy for artists to accept and allowed me to find a great solution.



Deferred Technique

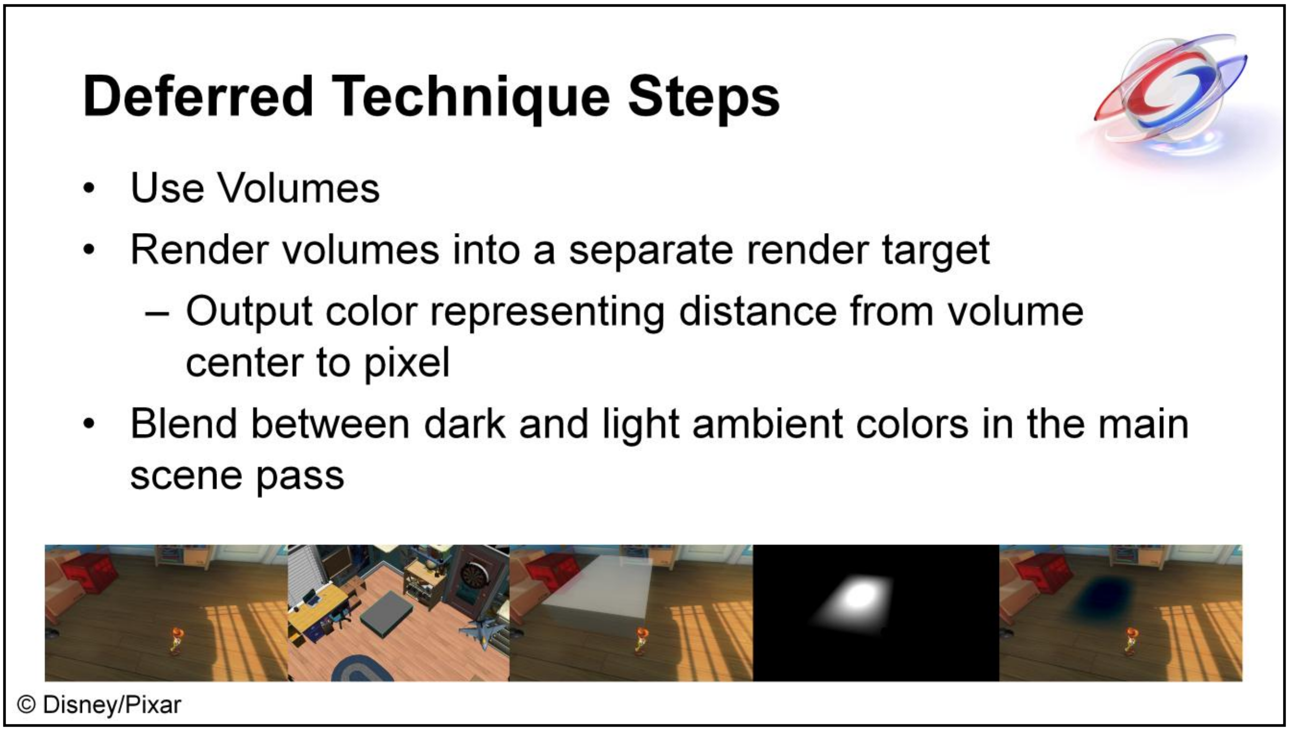
This solution works by rendering volumes into a deferred buffer and blending between the light and dark ambient rigs based on the buffer’s value. The primary advantage is that artists have more control over the ambient lighting.

Back to our problem…



Add a volume,
Cave is dark

© Disney/Pixar

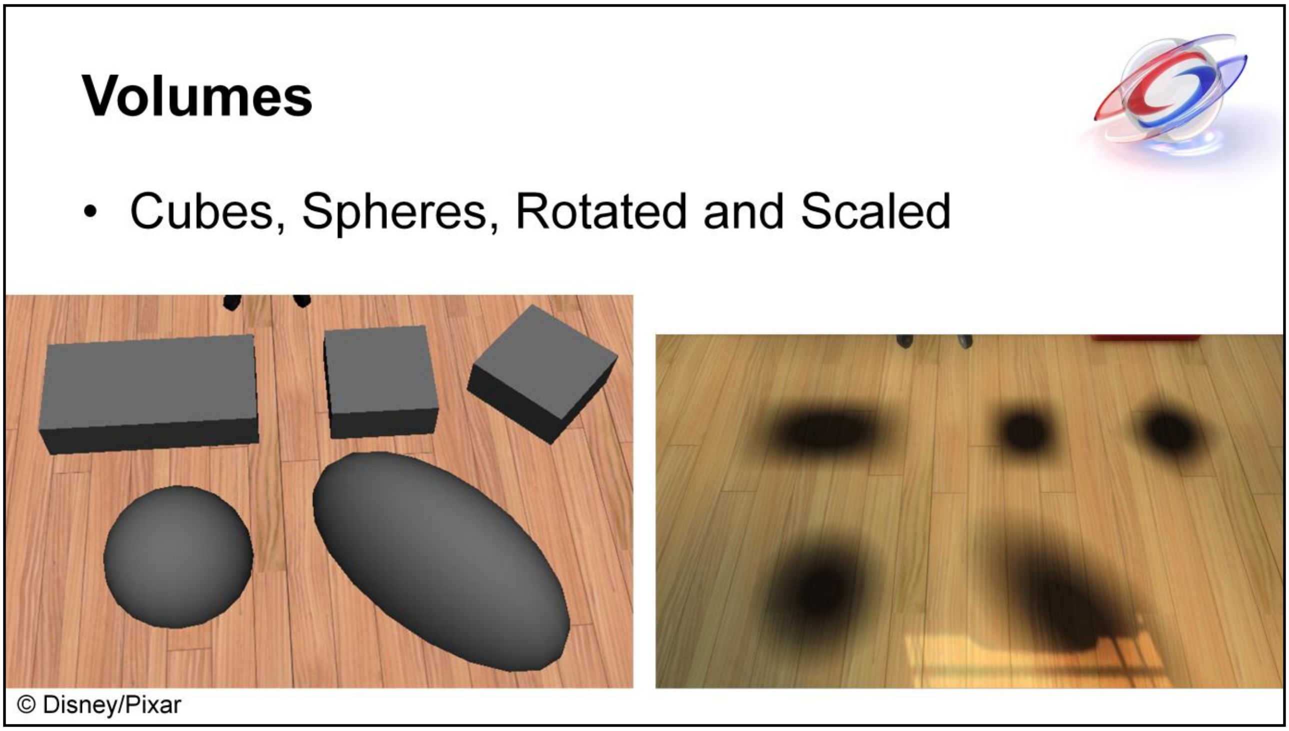


Deferred Technique Steps

- Use Volumes
- Render volumes into a separate render target
  - Output color representing distance from volume center to pixel
- Blend between dark and light ambient colors in the main scene pass

© Disney/Pixar

Artists place cubes or spheres into their worlds. When rendering in the engine, we have a separate pass that renders these volumes into a render target. The color written in this buffer represents the blend between the dark and the light ambient rigs In the main pass, we look up the value, and lerp between the light and dark ambient colors calculated from the spherical harmonic coefficients. To make it a little faster, only the first 4 dark SH coefficients were used to calculate the dark color.



Volumes

- Cubes, Spheres, Rotated and Scaled

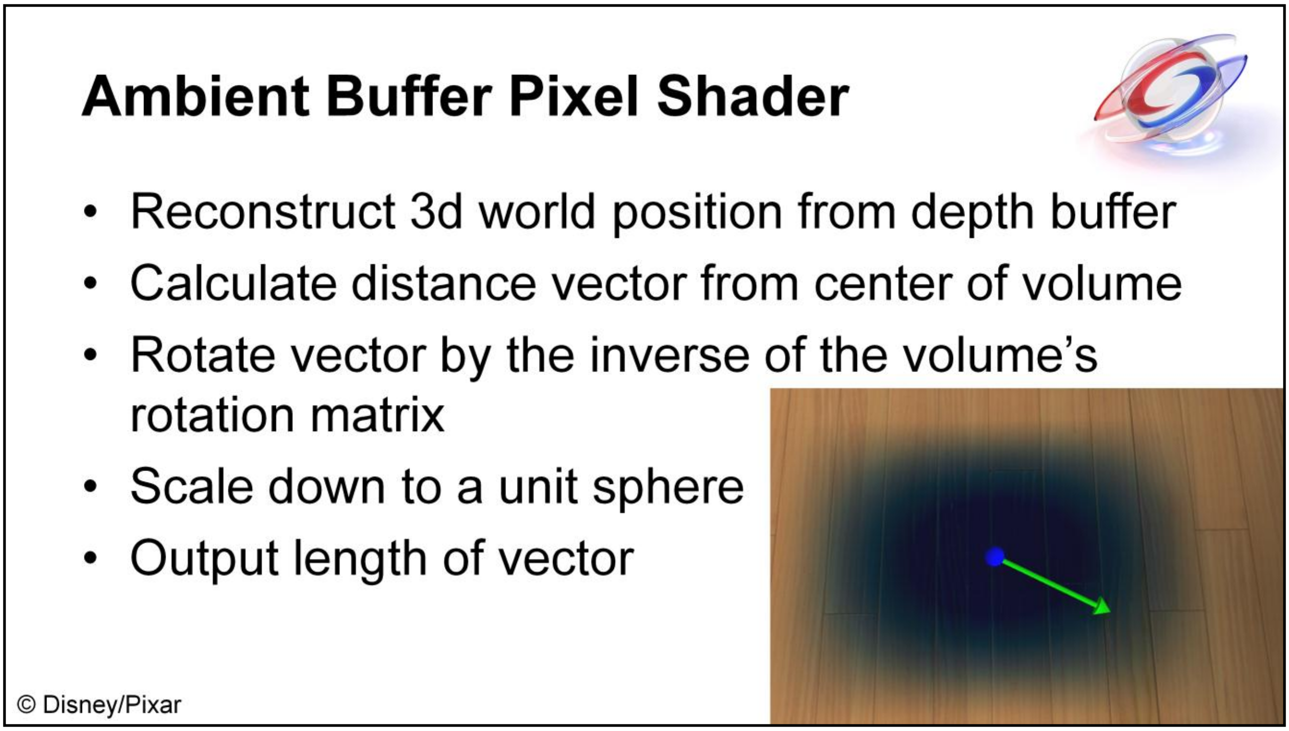
© Disney/Pixar

Spheres acted exactly like point lights in normal deferred lighting.

Cubes were a great fit for indoor areas but doing them was a little more complicated

I experimented with planes, but they weren’t as useful as I had first thought. I had great ideas of cutting the world with this plane so everything on the other slide would smoothly blend into the local ambient light. But the plane had to be visible for it to work. In the end, the box did everything the plane did and was easier to visualize.

I also added the capability to rotate and scale the volumes. This gave artists the flexibility to make them fit into the world better.

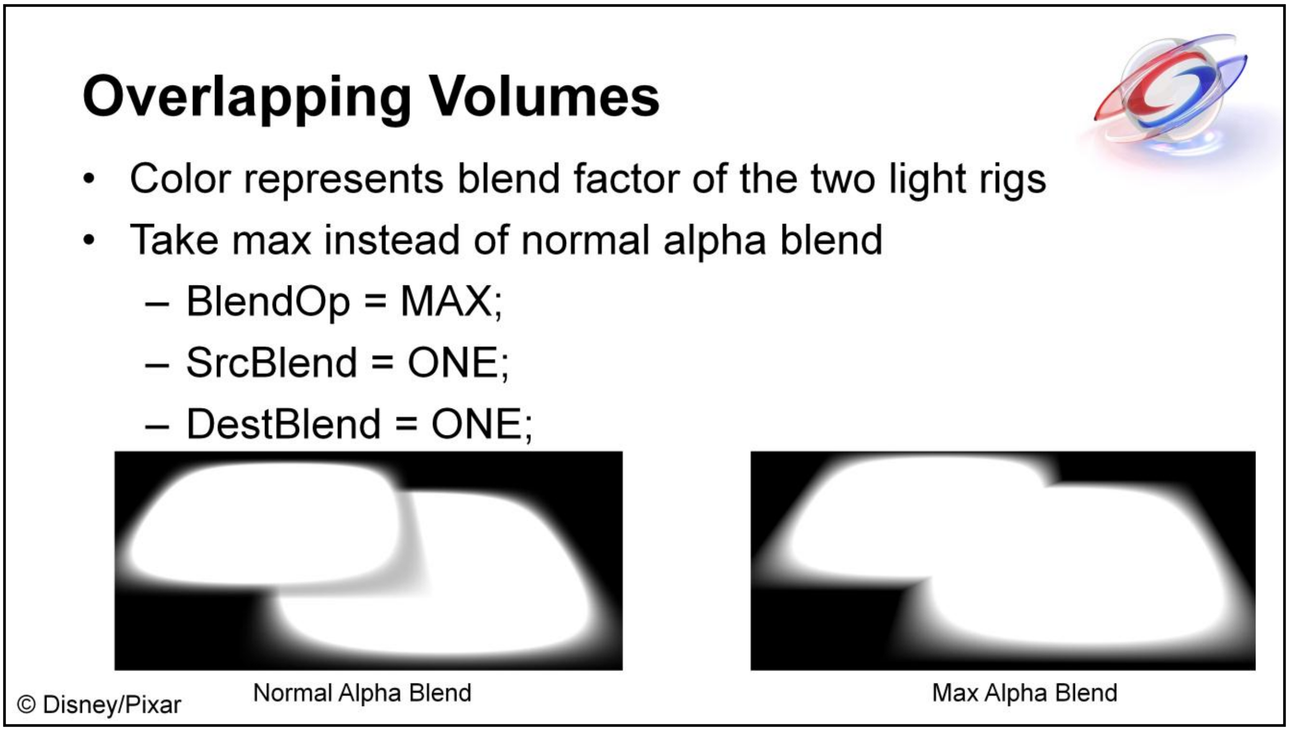


Ambient Buffer Pixel Shader

- Reconstruct 3d world position from depth buffer
- Calculate distance vector from center of volume
- Rotate vector by the inverse of the volume’s rotation matrix
- Scale down to a unit sphere
- Output length of vector

© Disney/Pixar

The result of this shader should be a float between 0 and 1 representing the blend factor between the two rigs.



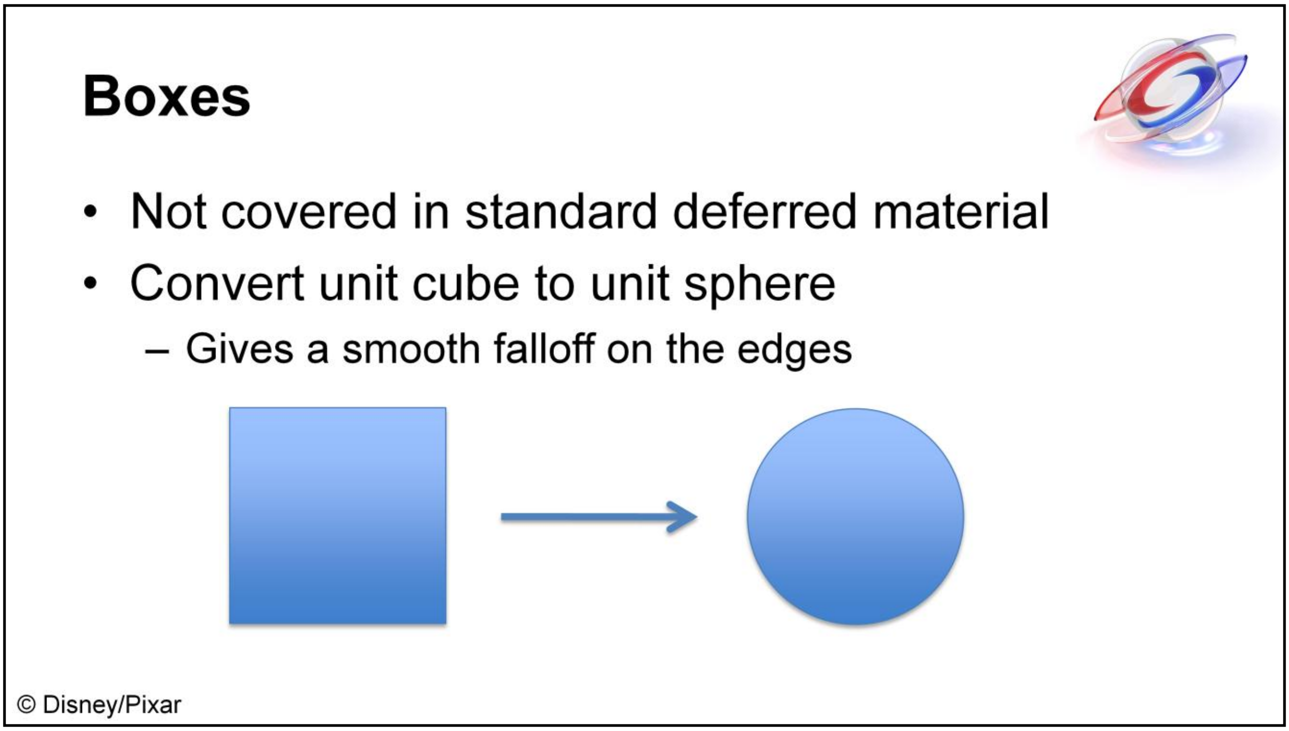
Overlapping Volumes

- Color represents blend factor of the two light rigs
- Take max instead of normal alpha blend
  - BlendOp = MAX;
  - SrcBlend = ONE;
  - DestBlend = ONE;

Normal Alpha Blend

Max Alpha Blend

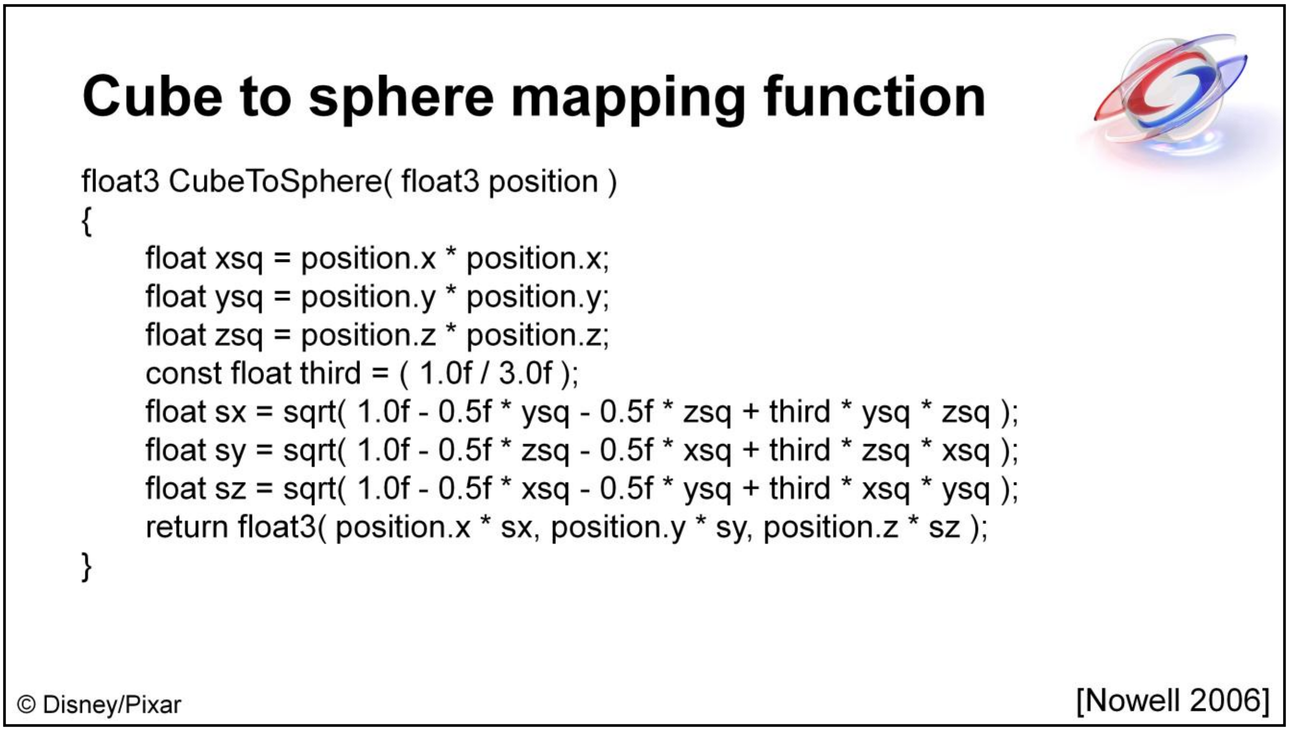
To handle blending for overlapping volumes we just need a different set of blend states. This allows the overlap to look correct.



Boxes

- Not covered in standard deferred material
- Convert unit cube to unit sphere
  - Gives a smooth falloff on the edges

All the material I read about deferred lighting only covered how to render point lights and spot lights. I wanted a smooth falloff to the edge of the cube, and allow the artists to place the volumes directly in the level. To get this falloff, I had to project the cube into a sphere. After that, I could do the rest of the calculation like a sphere.

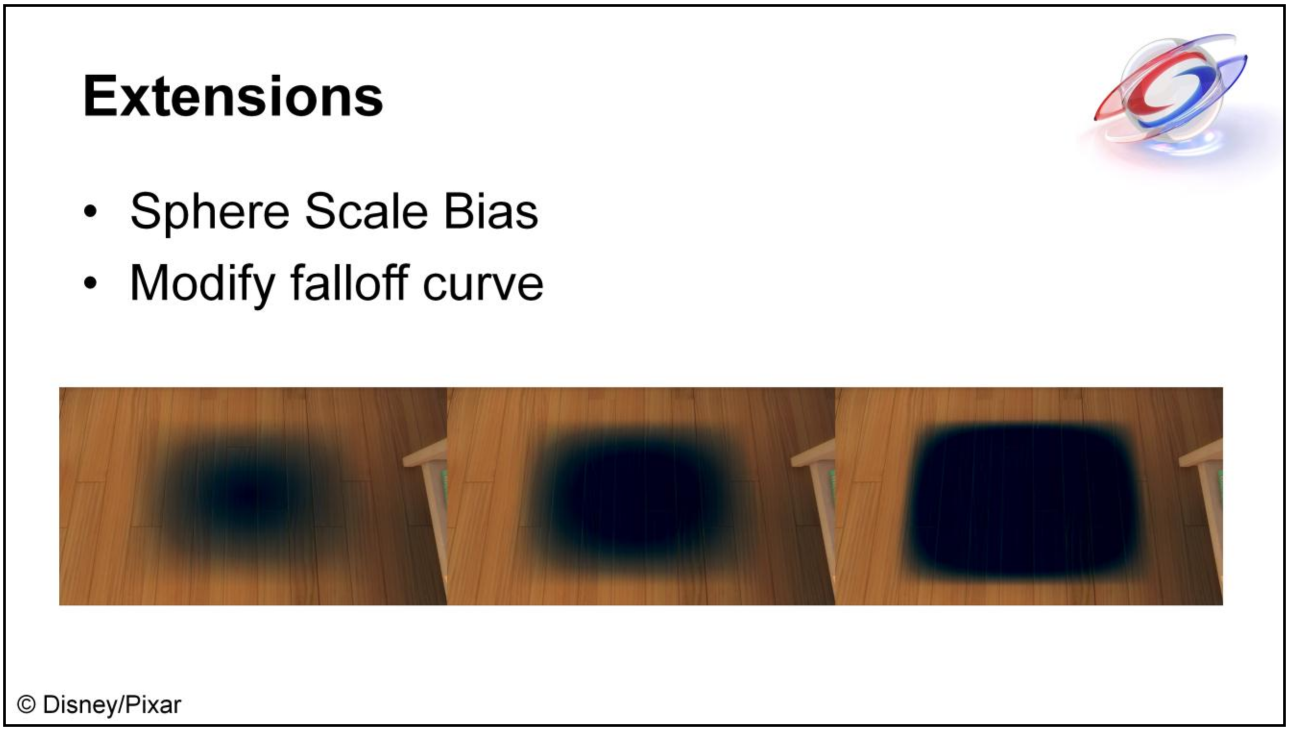


Cube to sphere mapping function

```
float3 CubeToSphere( float3 position )
{
      float xsq = position.x * position.x;
      float ysq = position.y * position.y;
      float zsq = position.z * position.z;
      const float third = ( 1.0f / 3.0f );
      float sx = sqrt( 1.0f - 0.5f * ysq - 0.5f * zsq + third * ysq * zsq );
      float sy = sqrt( 1.0f - 0.5f * zsq - 0.5f * xsq + third * zsq * xsq );
      float sz = sqrt( 1.0f - 0.5f * xsq - 0.5f * ysq + third * xsq * ysq );
      return float3( position.x * sx, position.y * sy, position.z * sz );
}
```

[Nowell 2006]

The reference contains more info into how this mapping looks and works.



Extensions

- Sphere Scale Bias
- Modify falloff curve

© Disney/Pixar

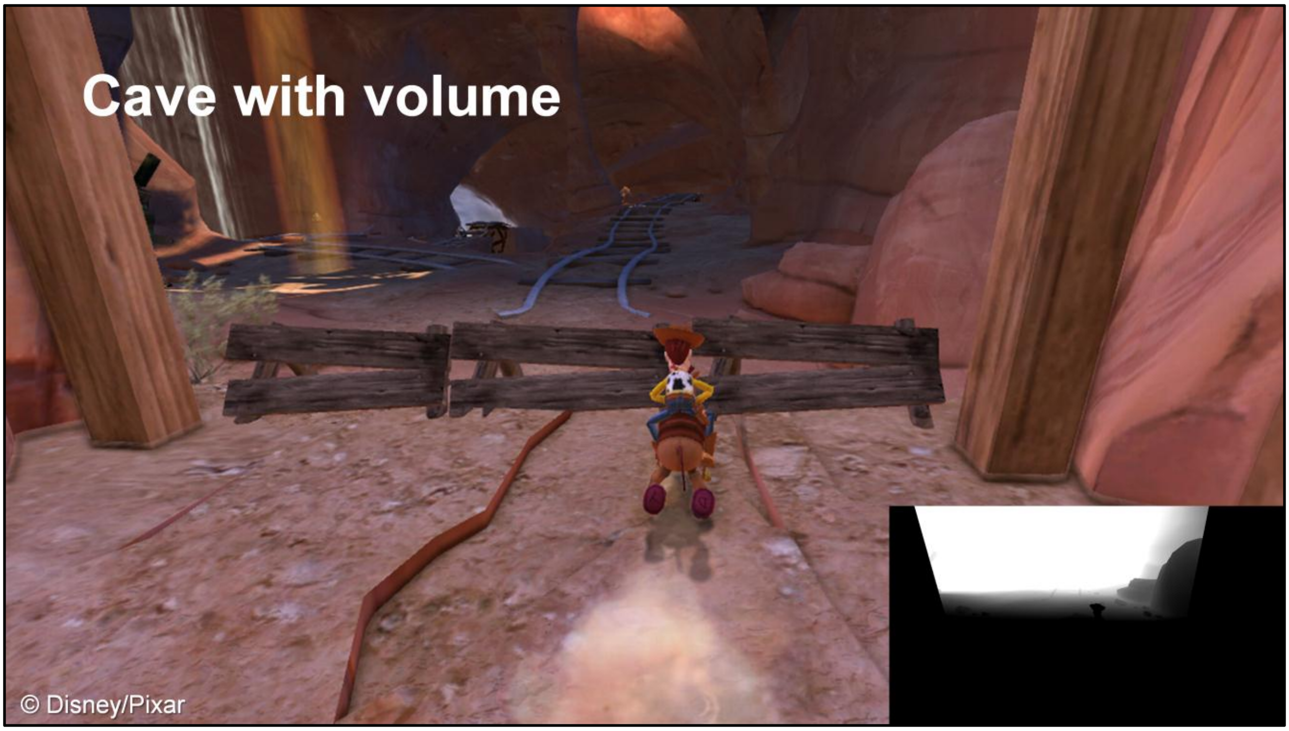
Sphere Scale Bias

Since artists place the volumes in the world, the spheres are tessellated coarsely as we didn’t need lots of vertices. When they get large, then there is a mismatch between the edges of the sphere and the shader that calculates based on a perfectly circular sphere. This is just a value that the artists bump up from 1.05 to increase the sphere scale thus eliminating the seams.

Modify falloff curve

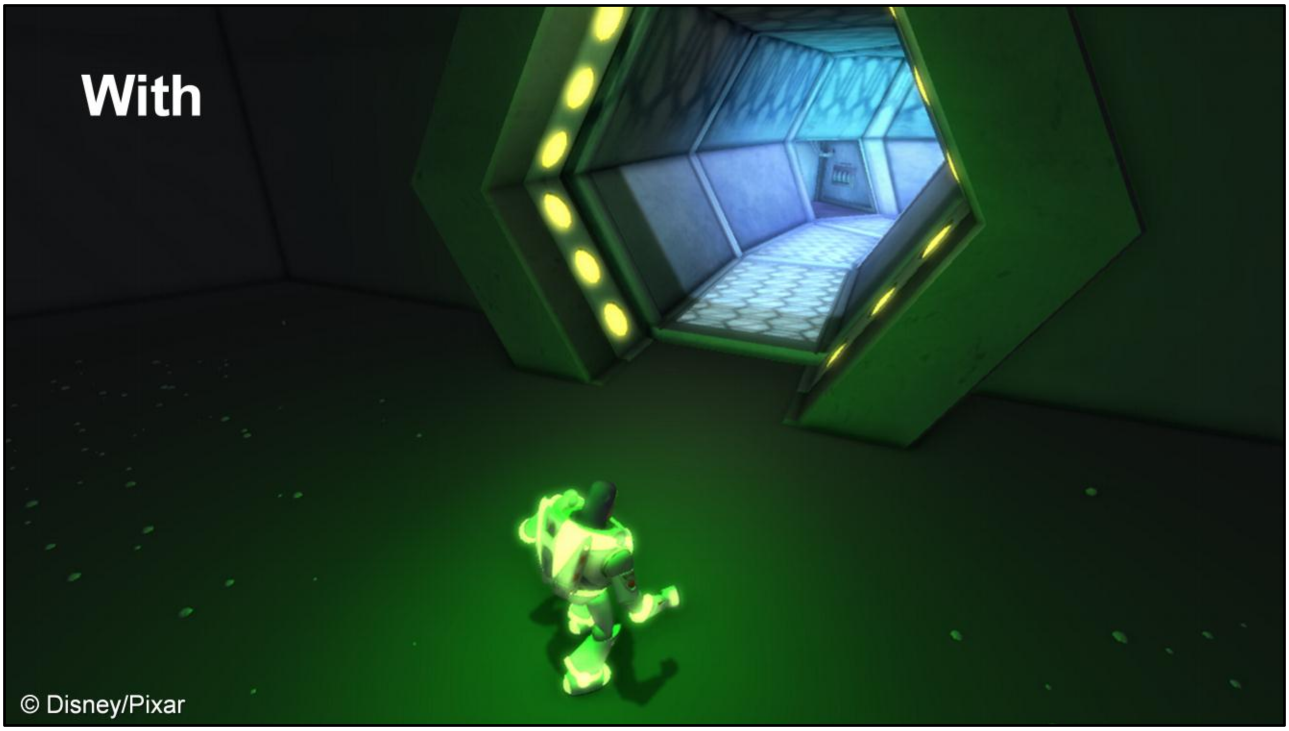
This just delays the distance before the falloff happens. The image on the left is with no falloff curve. The middle goes out 0.5 before beginning the falloff. The right waits to falloff until the length is more than 0.85. I didn’t experiment with any other curve modification.

Without
© Disney/Pixar

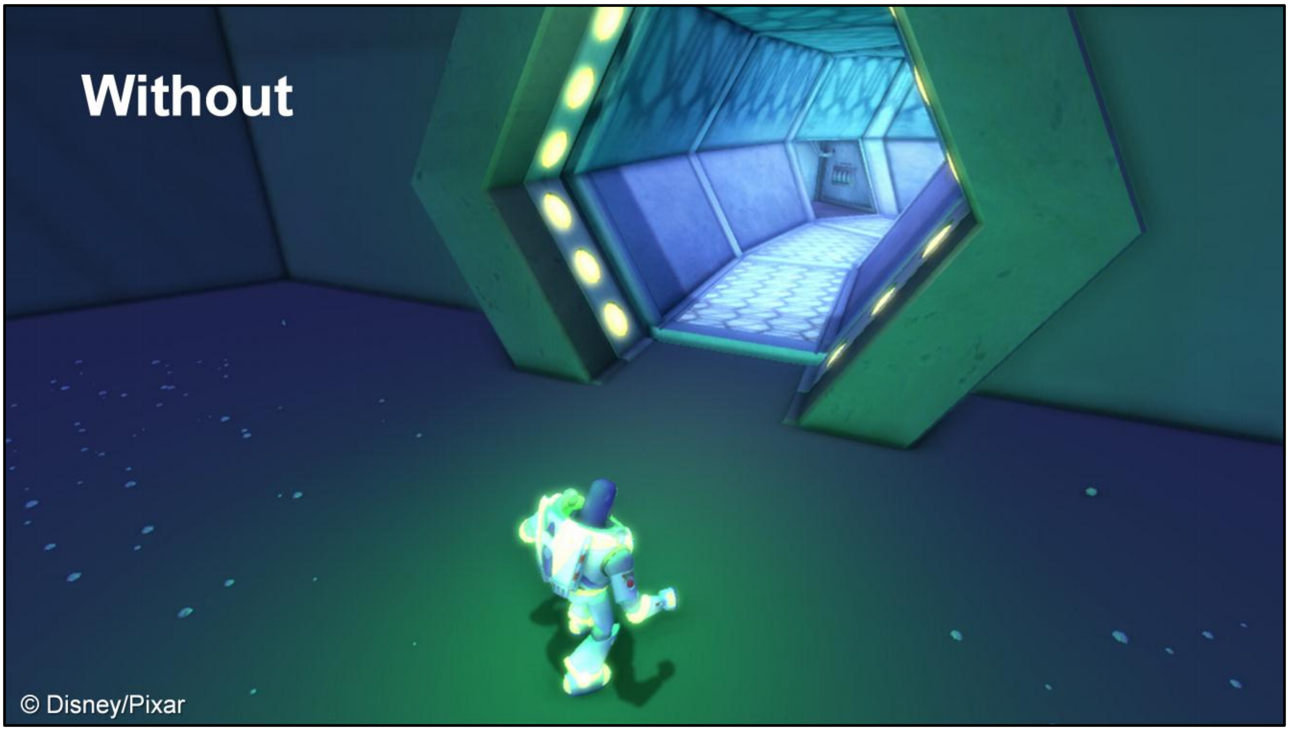


Cave with volume

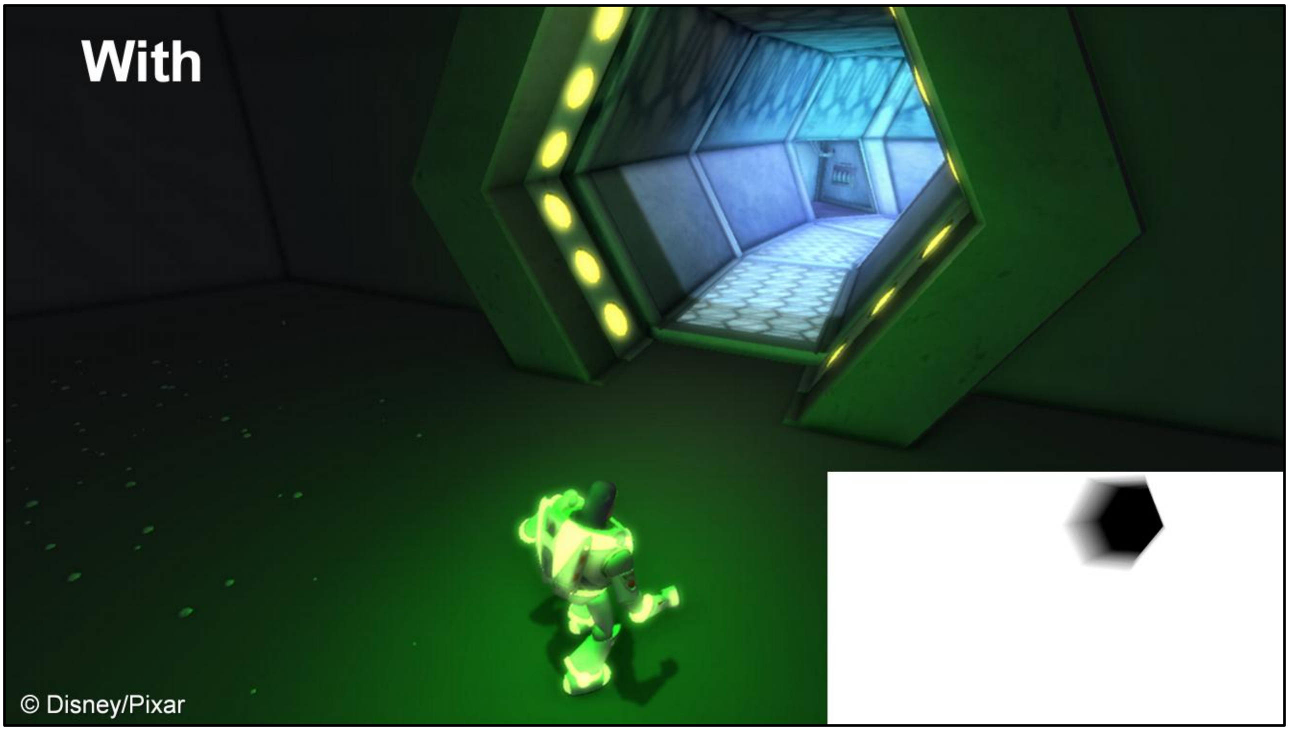
© Disney/Pixar



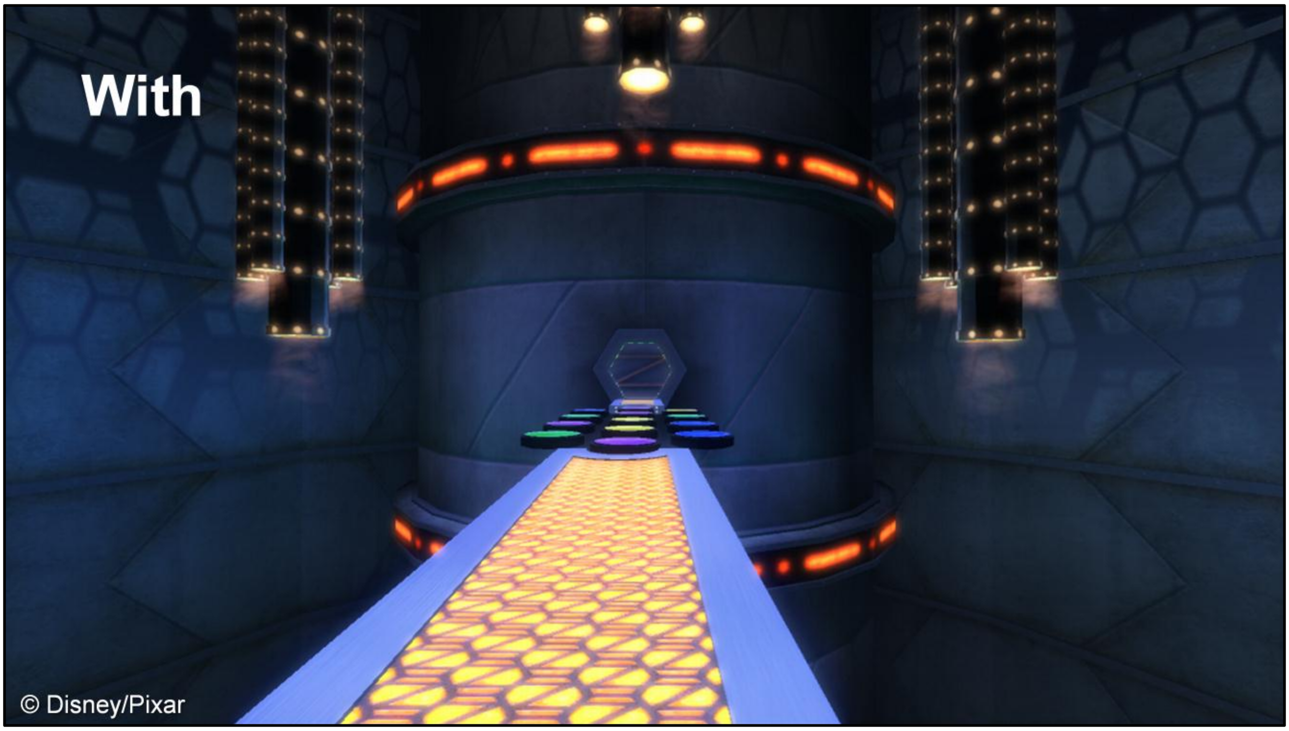
© Disney/Pixar


Without
© Disney/Pixar

With

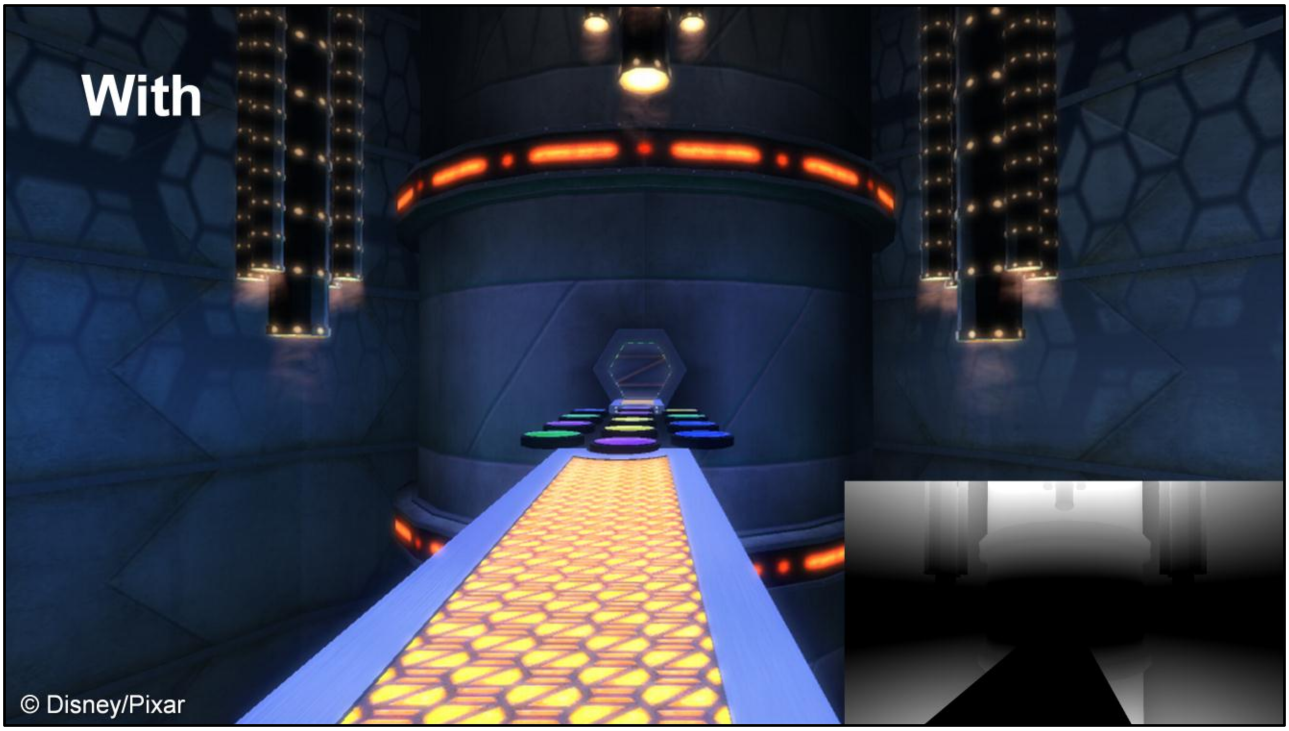


© Disney/Pixar

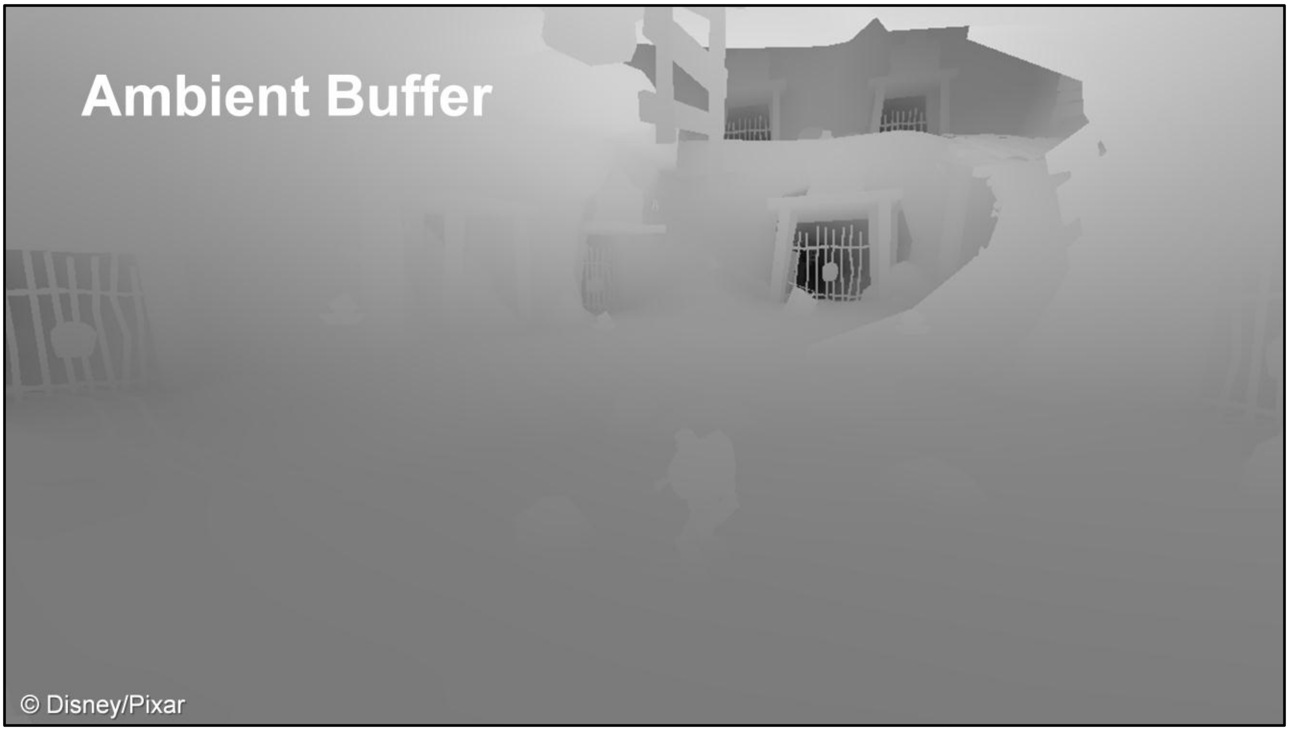

With

Ambient Buffer

© Disney/Pixar

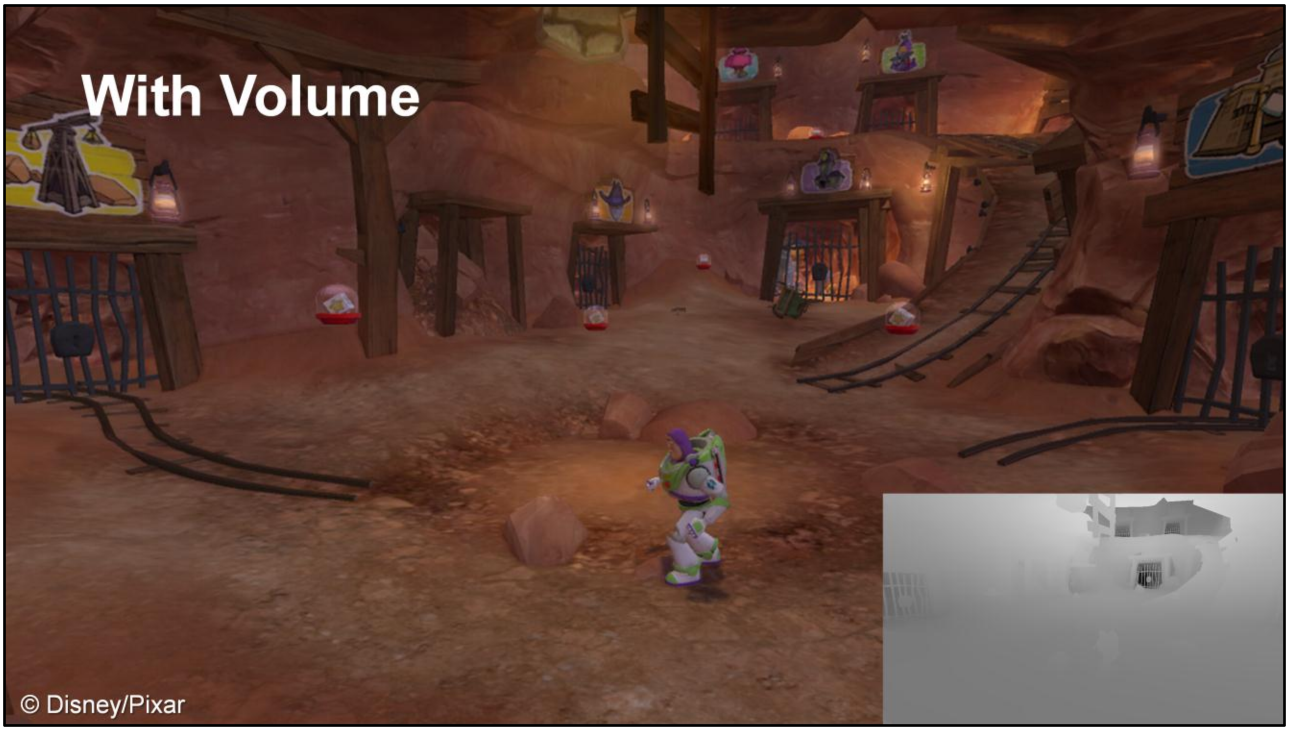


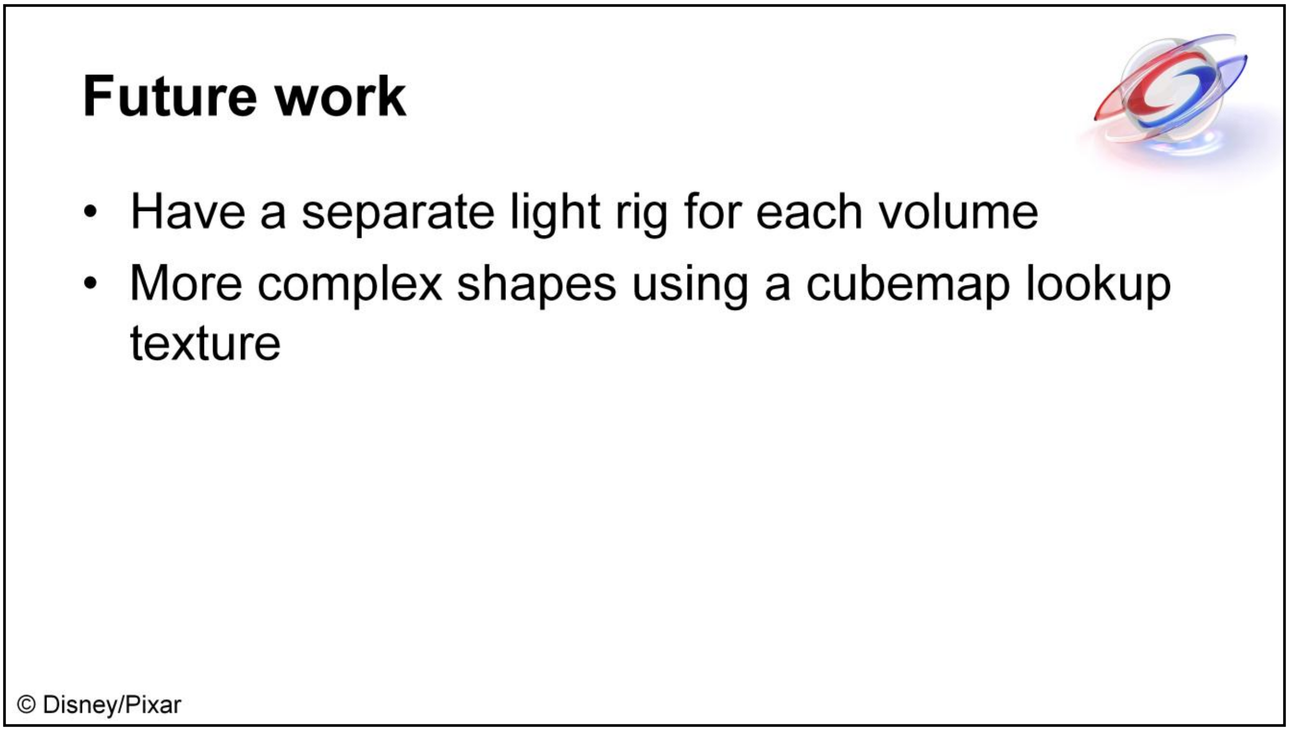
© Disney/Pixar



Ambient Buffer

© Disney/Pixar


With Volume
© Disney/Pixar



Future work

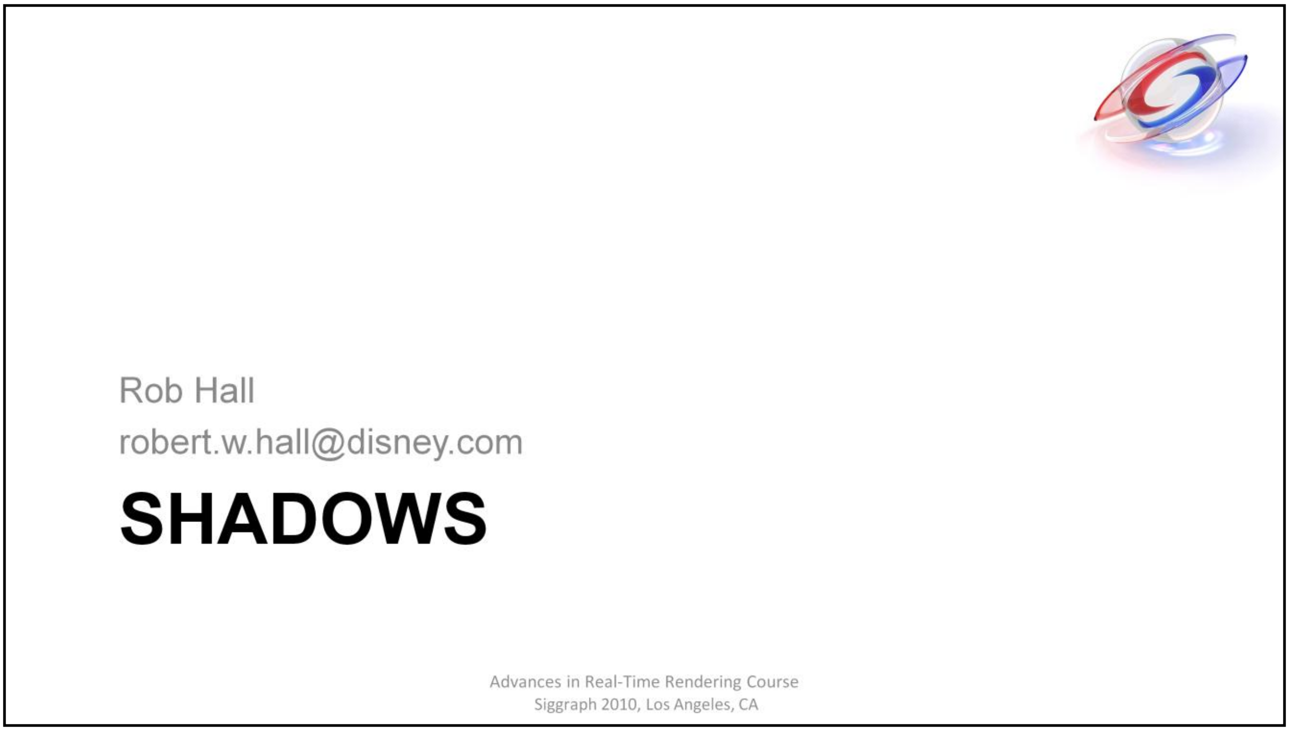
- Have a separate light rig for each volume
- More complex shapes using a cubemap lookup texture

Separate Light Rig

The volumes show the amount of blend to the dark ambient rig. Instead, each volume could blend the surface to it’s own unique ambient rig. Since we use forward rendering, this is difficult to do, but with deferred lighting it would be easier.

More complex shapes

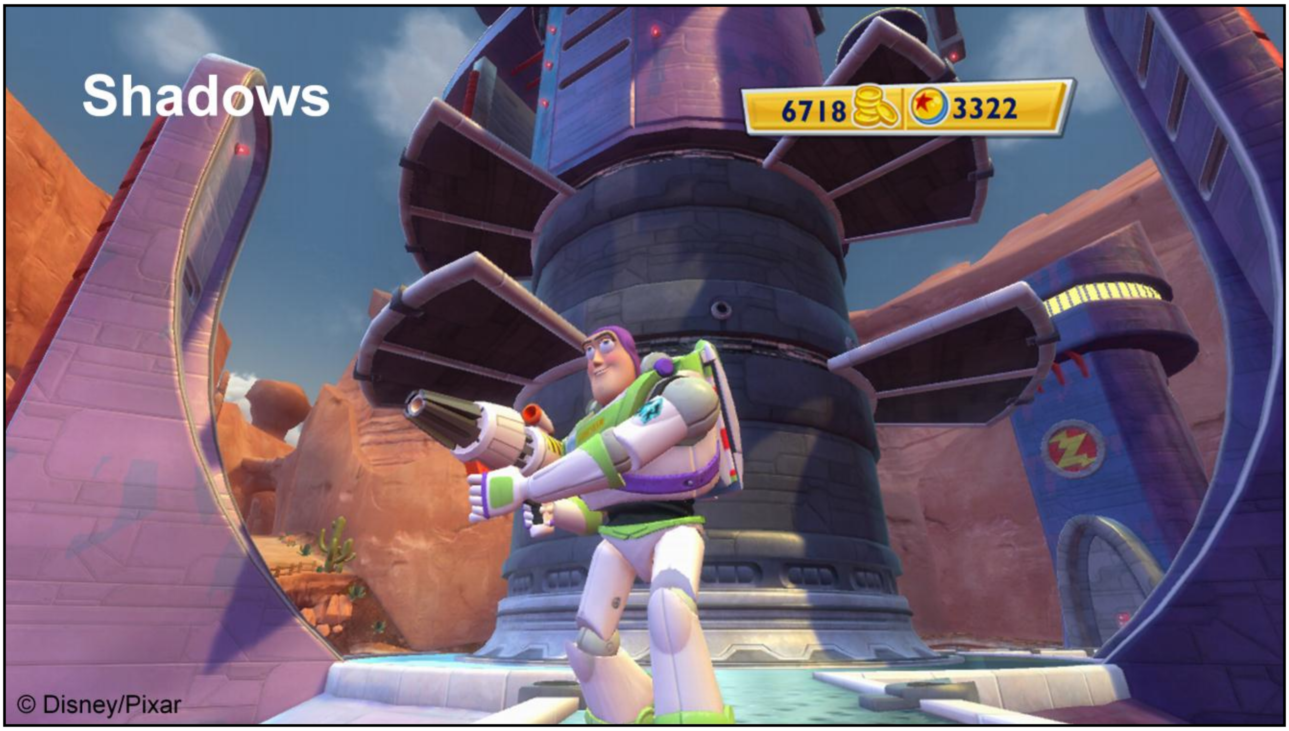
A cubemap lookup could be used so that the actual shaded shape can be different. This would make it easier to fit these volumes into the world.



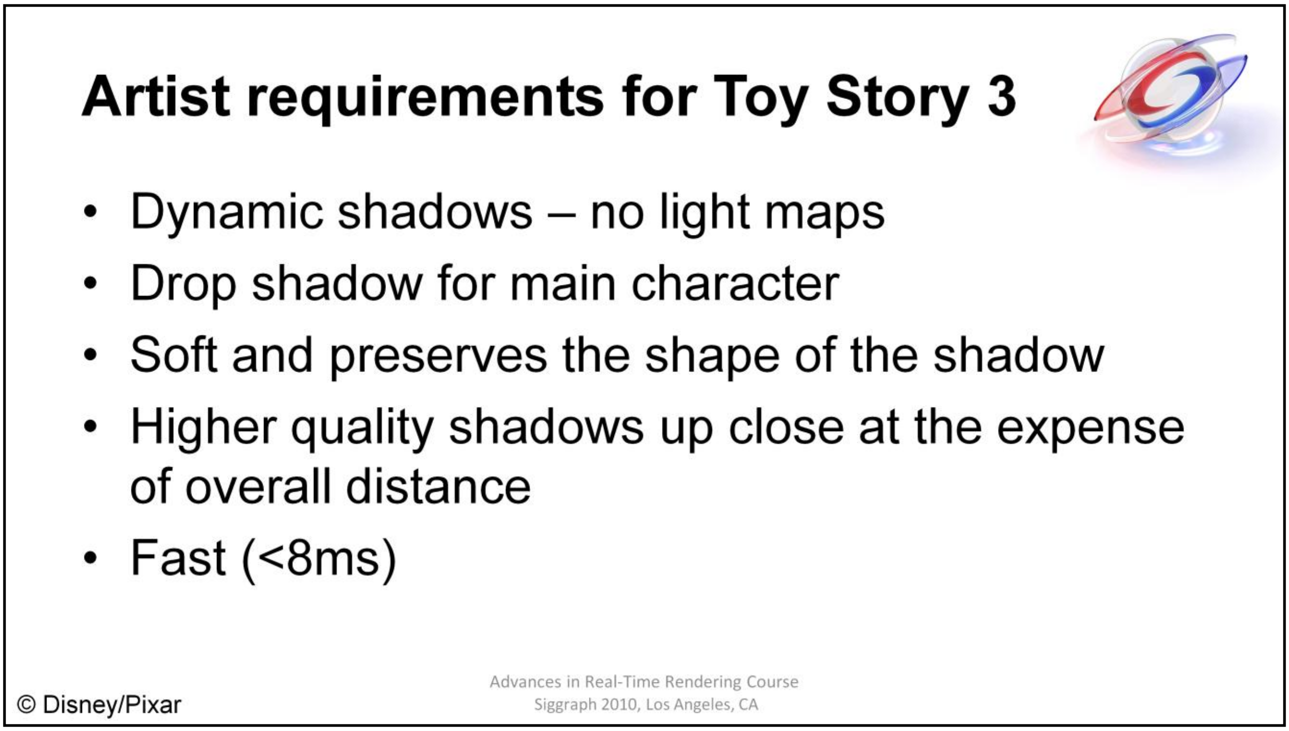
Rob Hall
robert.w.hall@disney.com

SHADOWS

Shadows



© Disney/Pixar



Artist requirements for Toy Story 3

- Dynamic shadows – no light maps
- Drop shadow for main character
- Soft and preserves the shape of the shadow
- Higher quality shadows up close at the expense of overall distance
- Fast (<8ms)

© Disney/Pixar

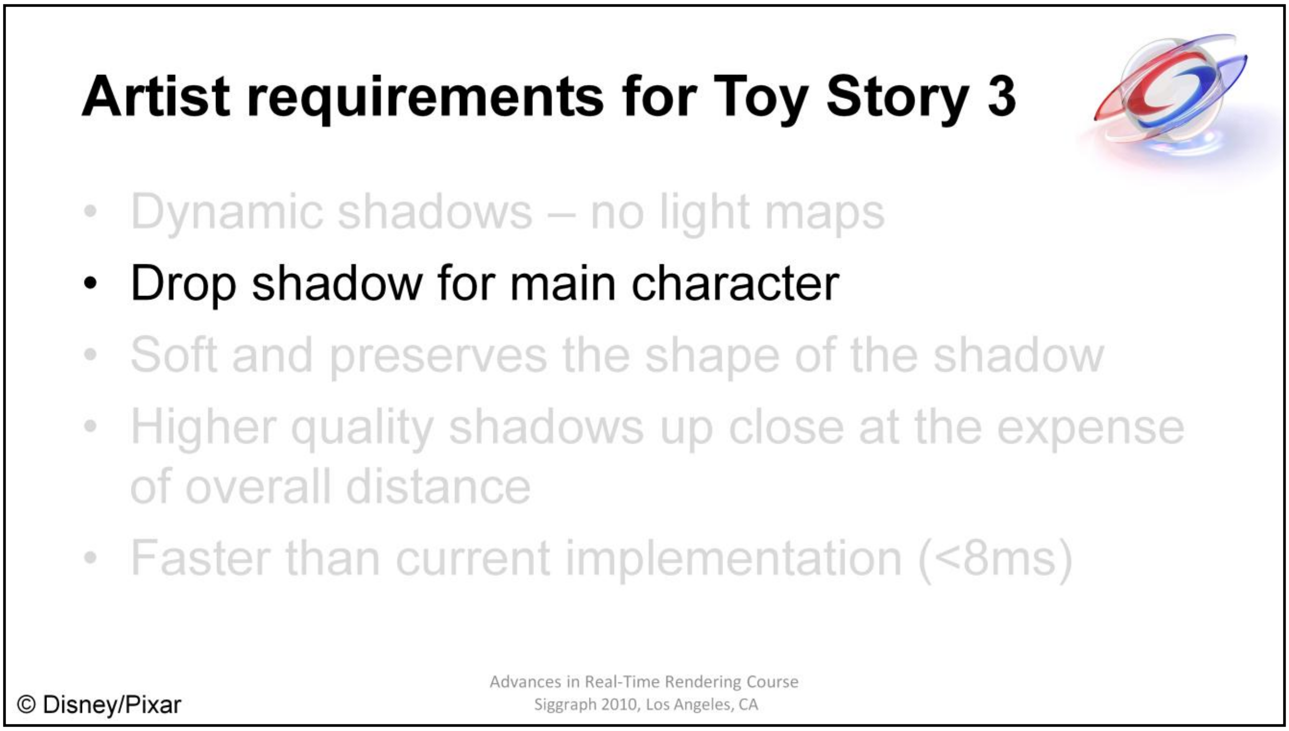
Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

To achieve a similar look that the movie has, it was very important to define a list of requirements for shadows

Game’s lighting and shadows are dynamic, so light maps are not an option

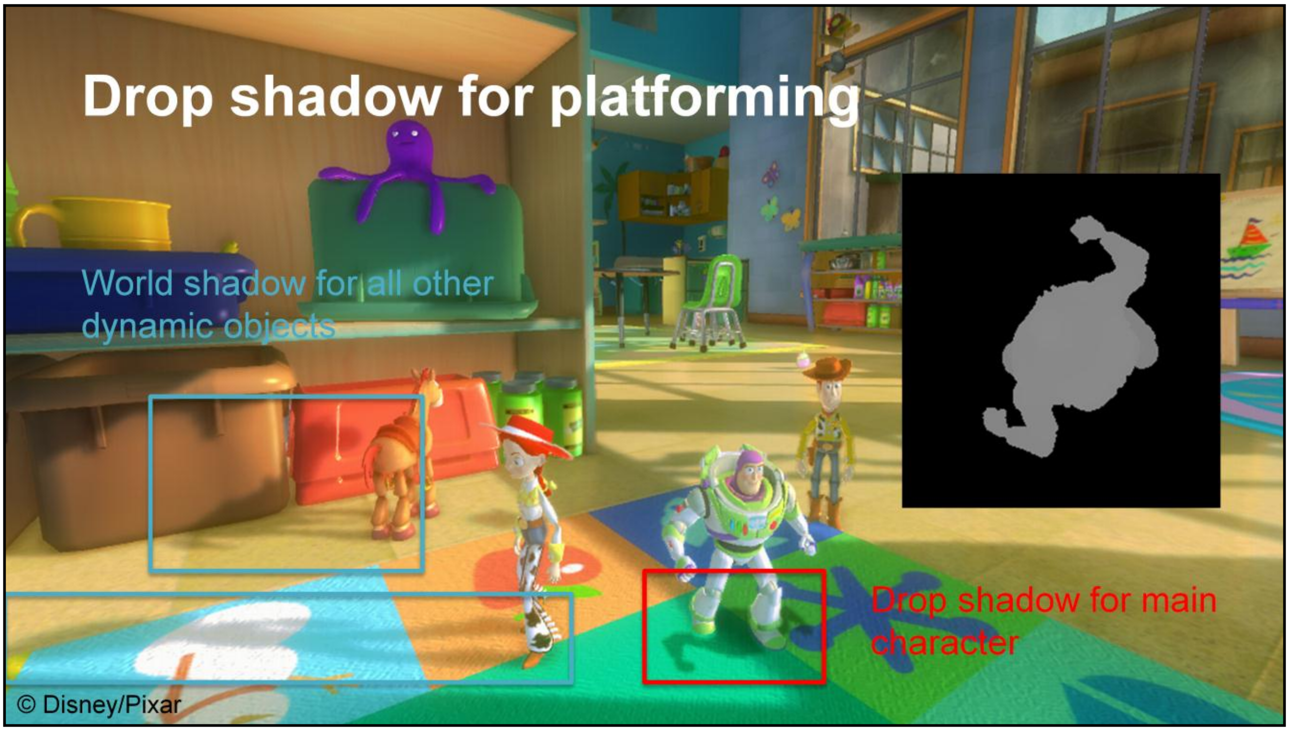
The main character needs a drop shadow to use as a cue for platforming

The world shadows have to be soft, high quality and inexpensive for consoles.



Artist requirements for Toy Story 3

- Dynamic shadows – no light maps
- Drop shadow for main character
- Soft and preserves the shape of the shadow
- Higher quality shadows up close at the expense of overall distance
- Faster than current implementation (<8ms)



Drop shadow for platforming

World shadow for all other dynamic objects

Drop shadow for main character

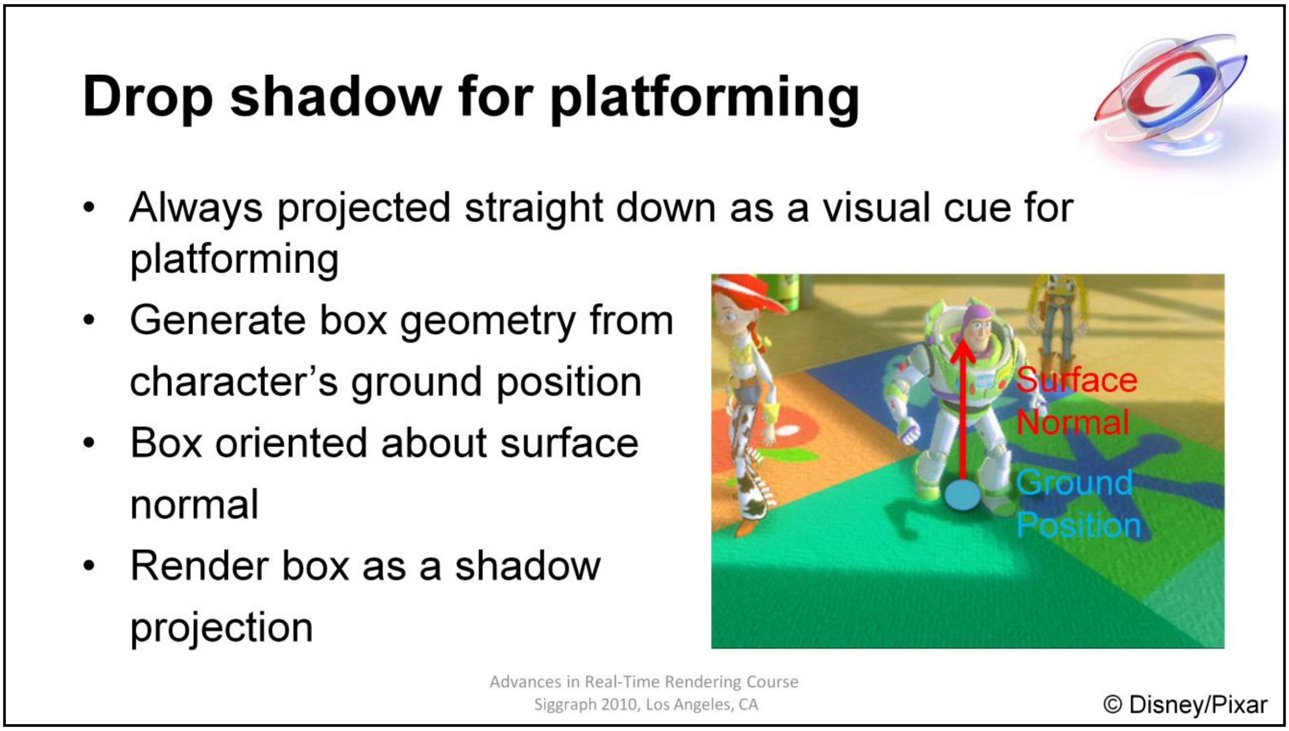
© Disney/Pixar

Only the main character used the drop shadow

See how buzz has a shadow projected straight down while Bullseye and Jessie still have the world shadow projected from the sun’s direction.

The drop shadow is the only option for platforming, because it will always be visible even if you walk inside a shadowed region.

It is projected straight down so that players can know where they will be landing after jumping



Drop shadow for platforming

- Always projected straight down as a visual cue for platforming
- Generate box geometry from character's ground position
- Box oriented about surface normal
- Render box as a shadow projection

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

© Disney/Pixar

The most inexpensive approach is to render it as geometry into the shadow mask

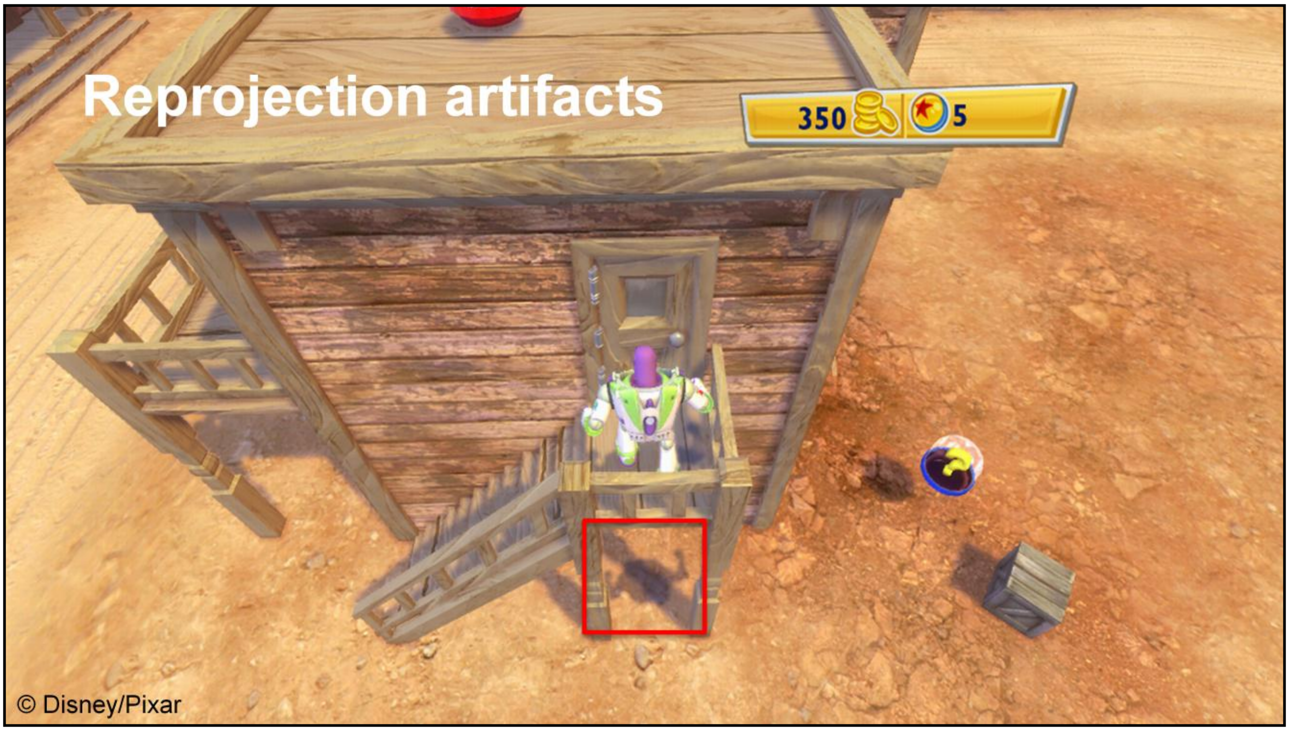
Without having to do any complicated physics calculations, we only find the ground position and surface normal for the main character

From this, we generate a box with artist defined dimensions of width and height

Since the shadow map is square, we used width for the other dimension

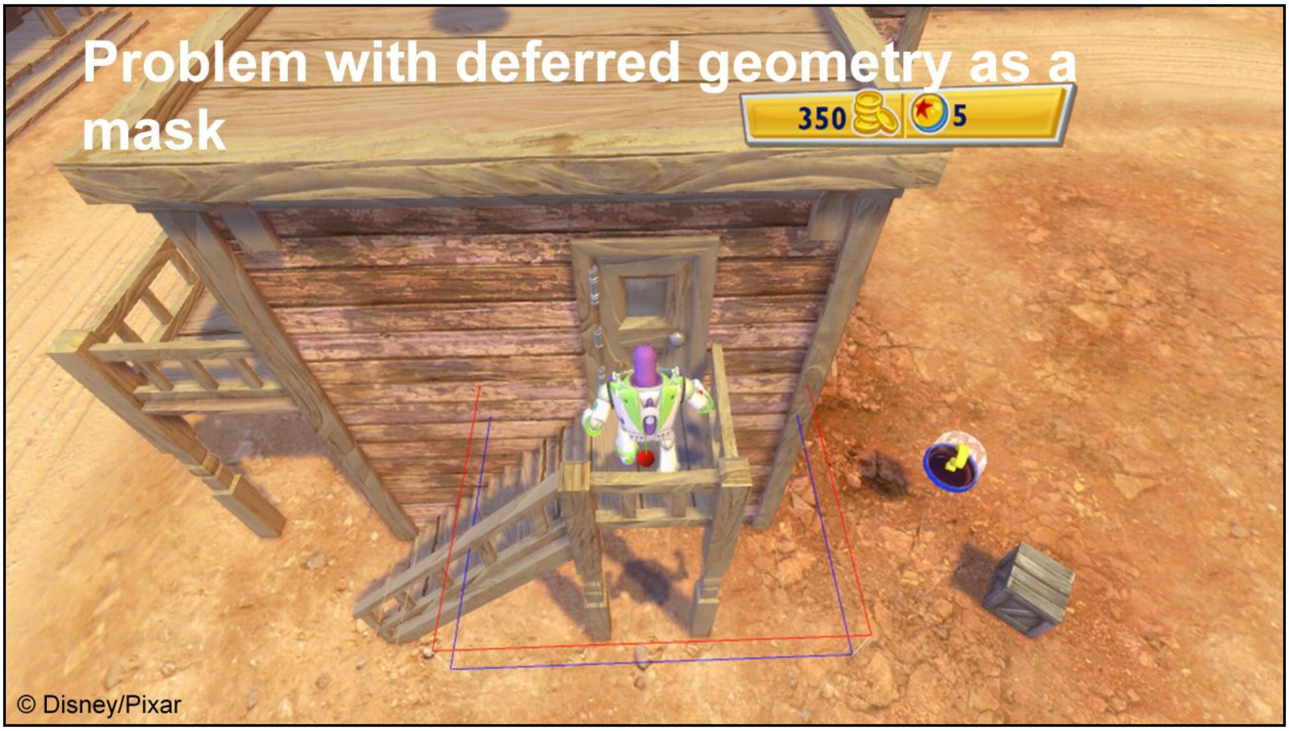
The pixel shader used for this box is a simple shadow map projection

Reprojection artifacts



© Disney/Pixar

Unfortunately, this approach exposes some reprojection artifacts

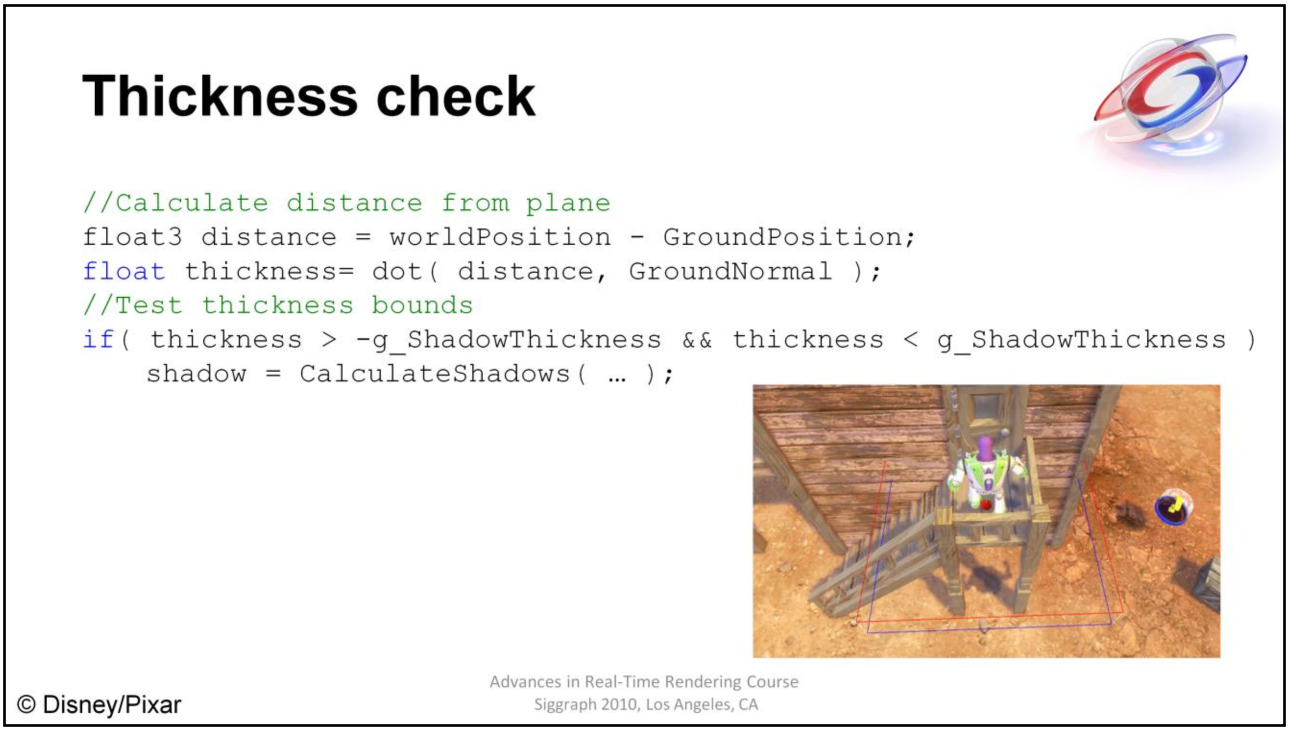


Problem with deferred geometry as a mask

© Disney/Pixar

Showing the actual box geometry shows the nature of the problem

The geometry is only used as a mask, so any reprojection that happens inside the mask cannot be removed



Thickness check

```
//Calculate distance from plane
float3 distance = worldPosition - GroundPosition;
float thickness= dot( distance, GroundNormal );
//Test thickness bounds
if( thickness > -g_ShadowThickness && thickness < g_ShadowThickness )
    shadow = CalculateShadows( … );
```

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The only option is to do a thickness check in the pixel shader

This checks the thickness of the box, which is the height of the box

One thing to be careful about is when the character is on non level surfaces

If there are any pits that lie outside the geometry’s thickness they will be clipped, which might not be expected

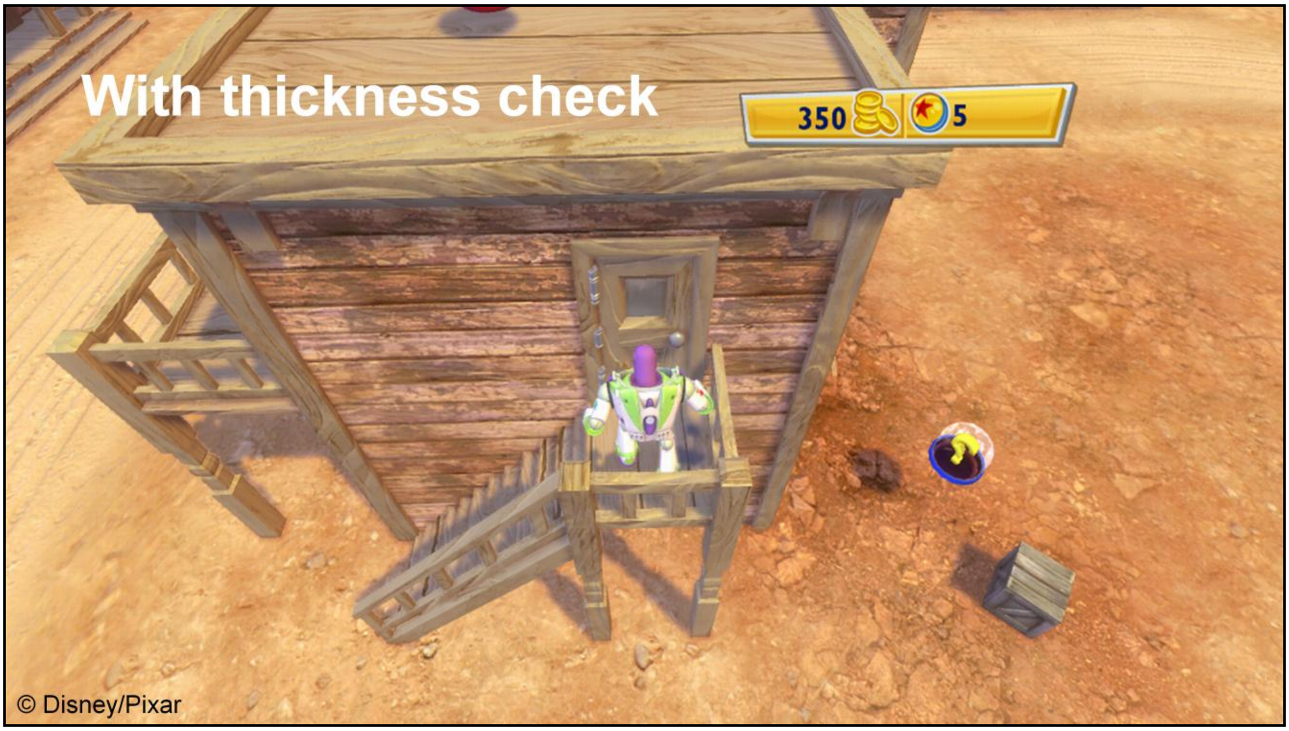
Make sure the box’s thickness is large enough to cover those pits, but not too large

If the box’s thickness is too large then when the geometry clips with a wall, then the wall will have the shadow projected on it

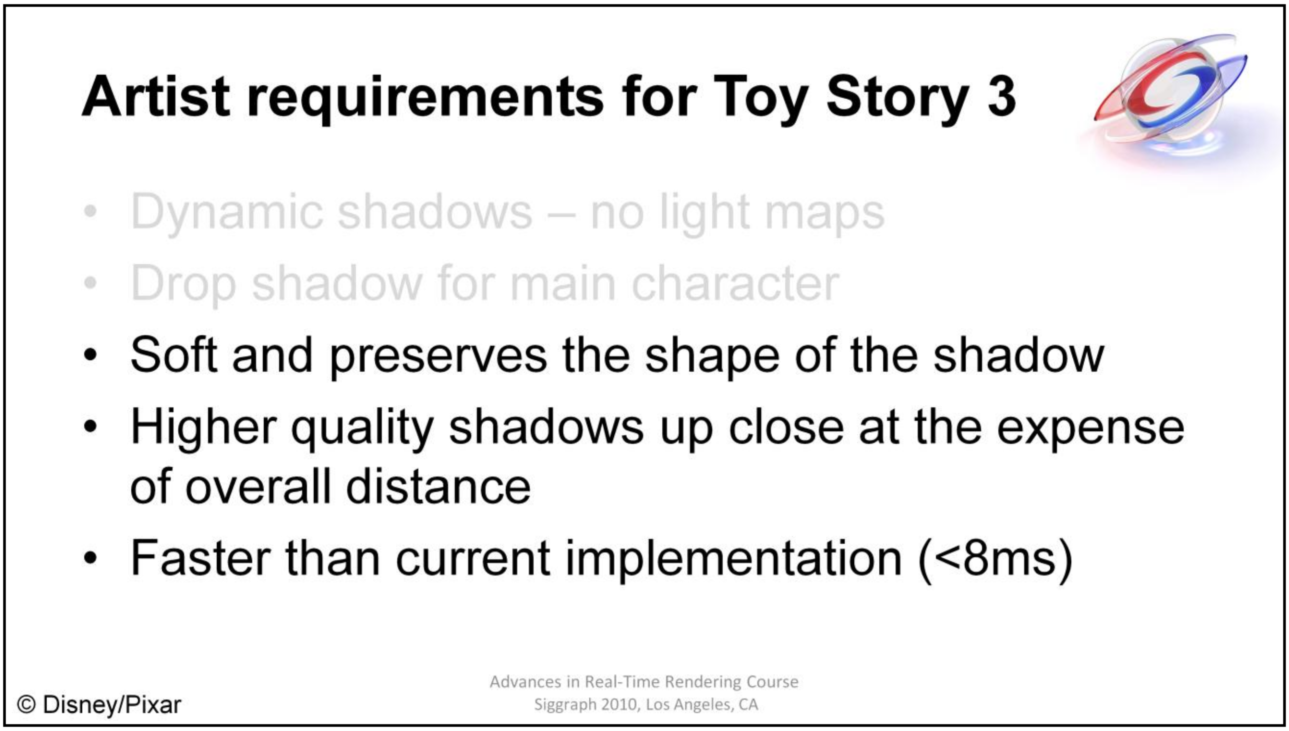
Since the shadow is projected straight down, it will project in the entirety of the box’s bounds

In the picture shown, it is possible that the shadow could project entirely down the columns supporting the stairs

A small thickness prevents this from happening



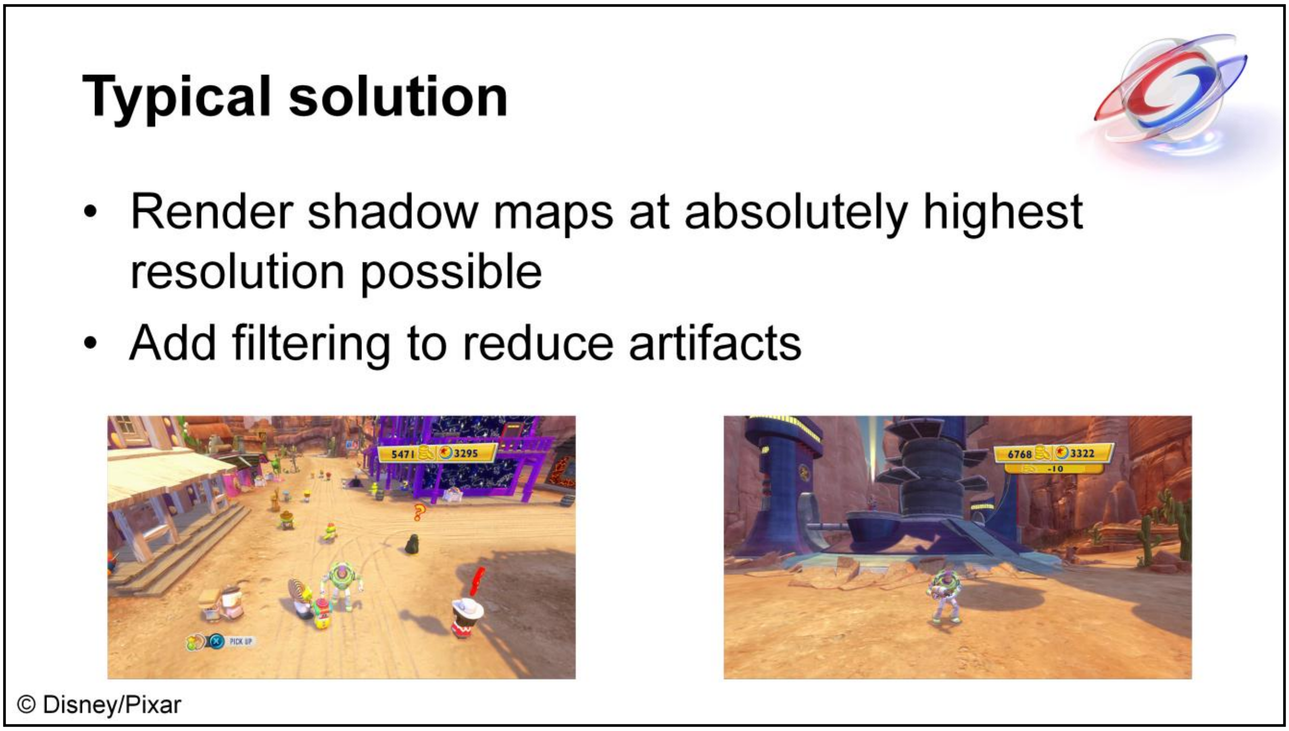
With this thickness check, the reprojection artifact is gone



Artist requirements for Toy Story 3

- Dynamic shadows – no light maps
- Drop shadow for main character
- Soft and preserves the shape of the shadow
- Higher quality shadows up close at the expense of overall distance
- Faster than current implementation (<8ms)

These last three items is part of the world shadows

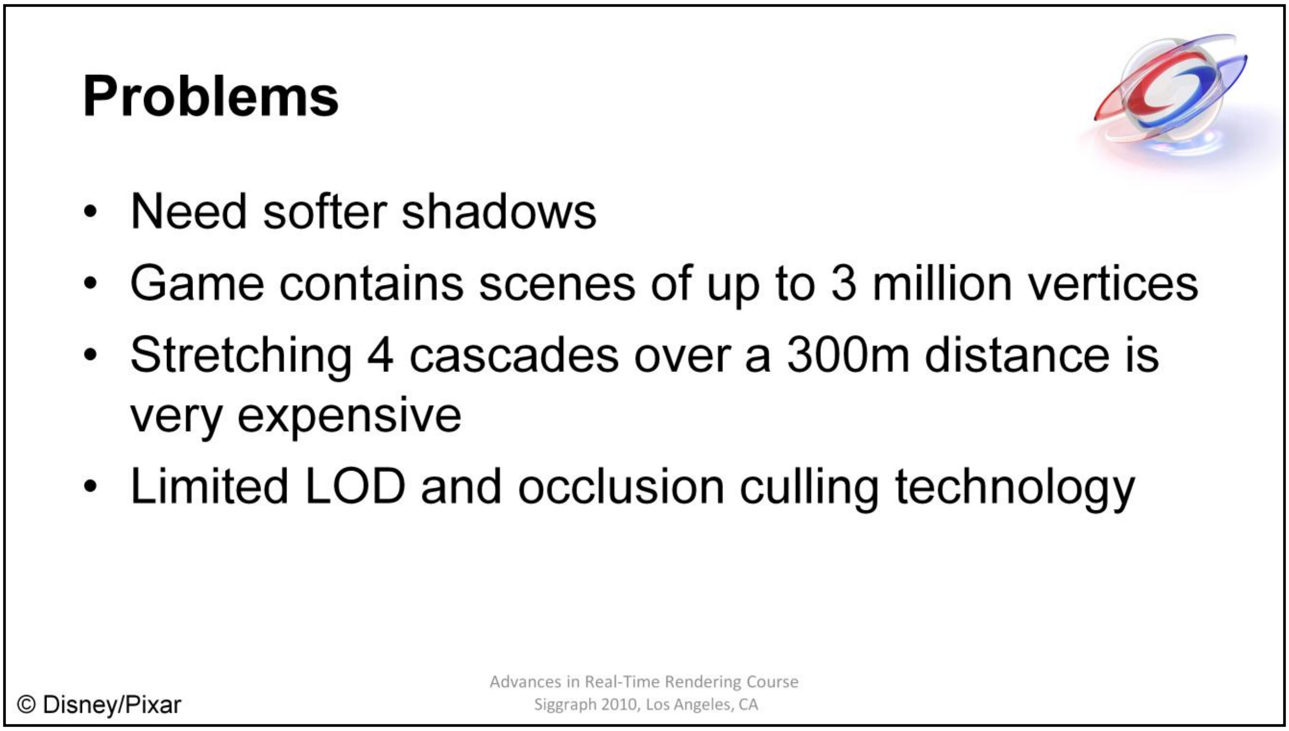


Typical solution

- Render shadow maps at absolutely highest resolution possible
- Add filtering to reduce artifacts

© Disney/Pixar

Many games pick this option, because it provides the best resolution and the most realistic shadows



Problems

- Need softer shadows
- Game contains scenes of up to 3 million vertices
- Stretching 4 cascades over a 300m distance is very expensive
- Limited LOD and occlusion culling technology

We are bound from picking this solution due to performance constraints

It is not possible performance wise to have both high resolution shadow maps and expensive filtering to make it soft

The amount of geometry rendered in each cascade is very expensive when rendered at high resolution

Handling complex scenes

- Solutions?

© Disney/Pixar

The main town area of Toy Story 3 enables the character to build buildings and add customizations, which all add geometry to the scene

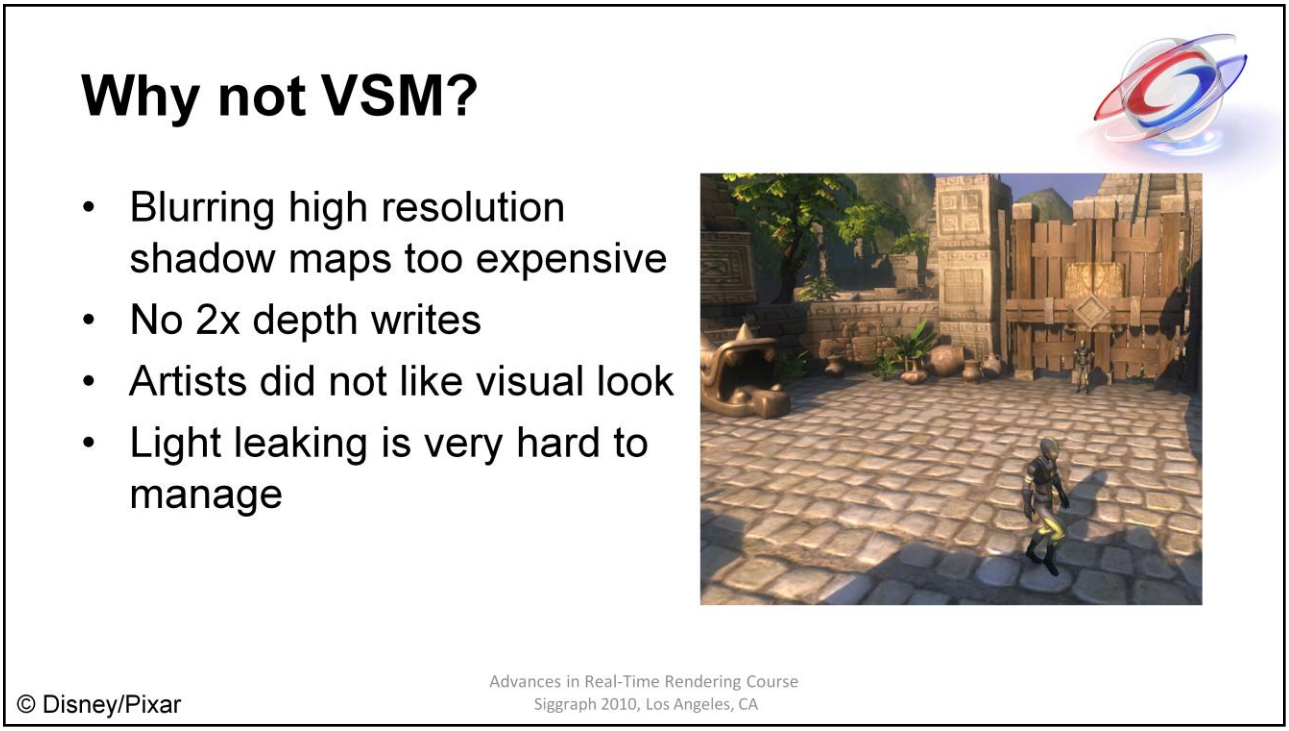
The scene can also contain a handful of rideable characters and cars

Each townsperson is fully customizable making it very difficult to instance

Why not VSM?

- Previously used in Bolt

© Disney/Pixar



Why not VSM?

- Blurring high resolution shadow maps too expensive
- No 2x depth writes
- Artists did not like visual look
- Light leaking is very hard to manage

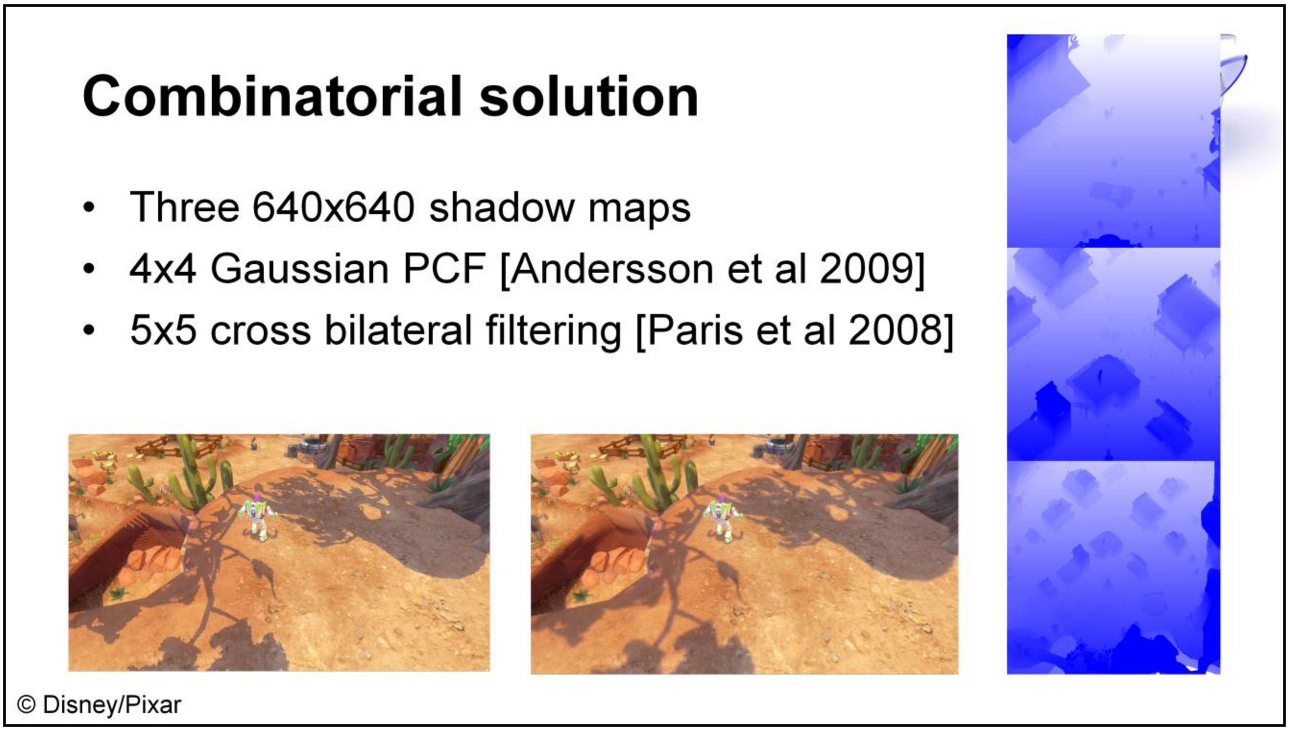
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

In many cases, the worst case performance for shadows in Bolt was over 12ms on the Xbox 360

Artists also did not like the non realistic shadow outlines

Limited precision also prevented accurate grounding of the characters with their shadow



Combinatorial solution

- Three 640x640 shadow maps
- 4x4 Gaussian PCF [Andersson et al 2009]
- 5x5 cross bilateral filtering [Paris et al 2008]

© Disney/Pixar

This solution deliberately renders low resolution shadow maps

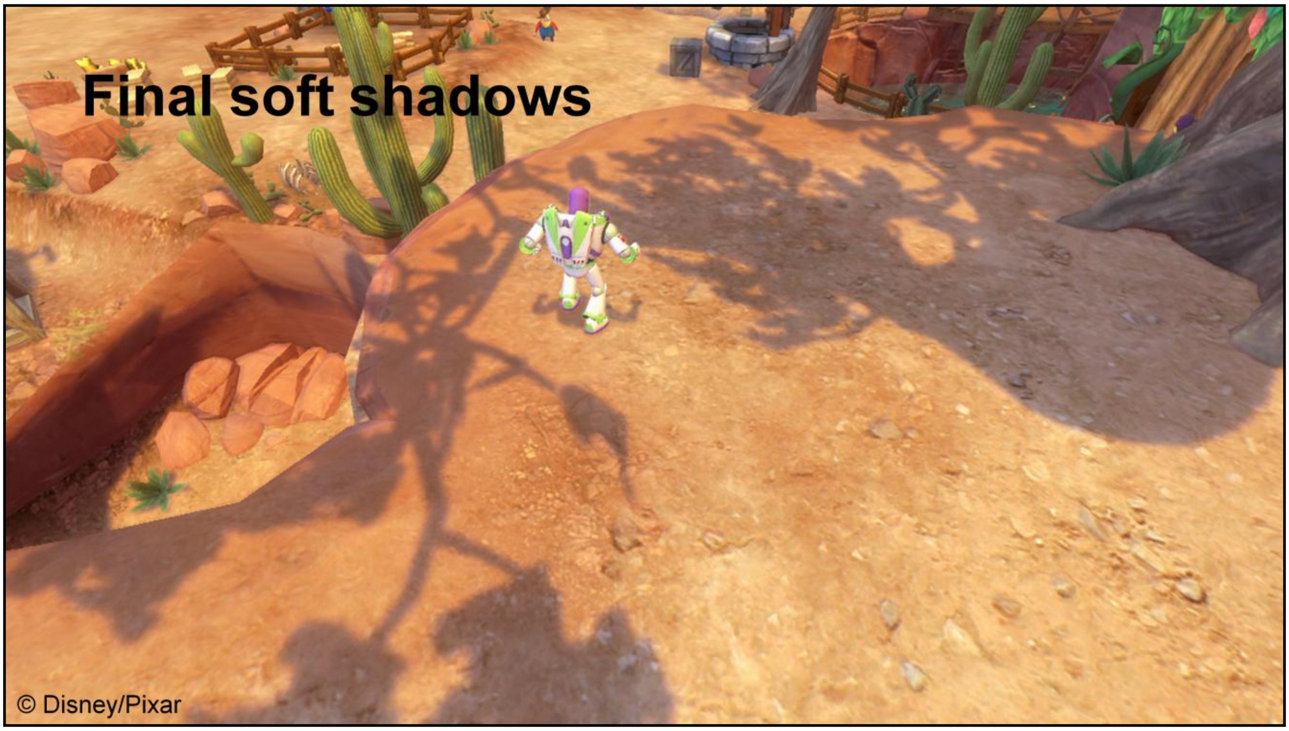
Shadow map generation cost low enough to handle complex scenes

This works due to the high quality filtering on the shadow test and in post

Advantages

- Inexpensive and still looks good!

© Disney/Pixar

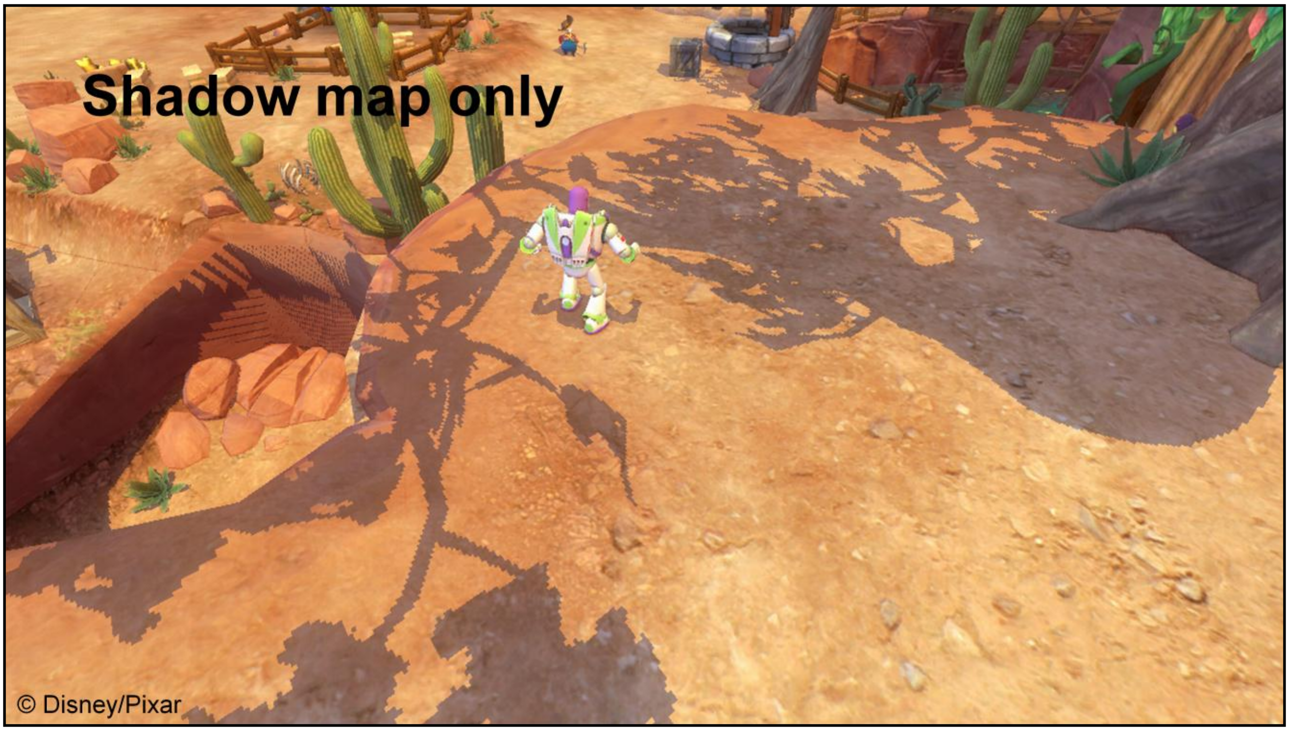


Final soft shadows

© Disney/Pixar

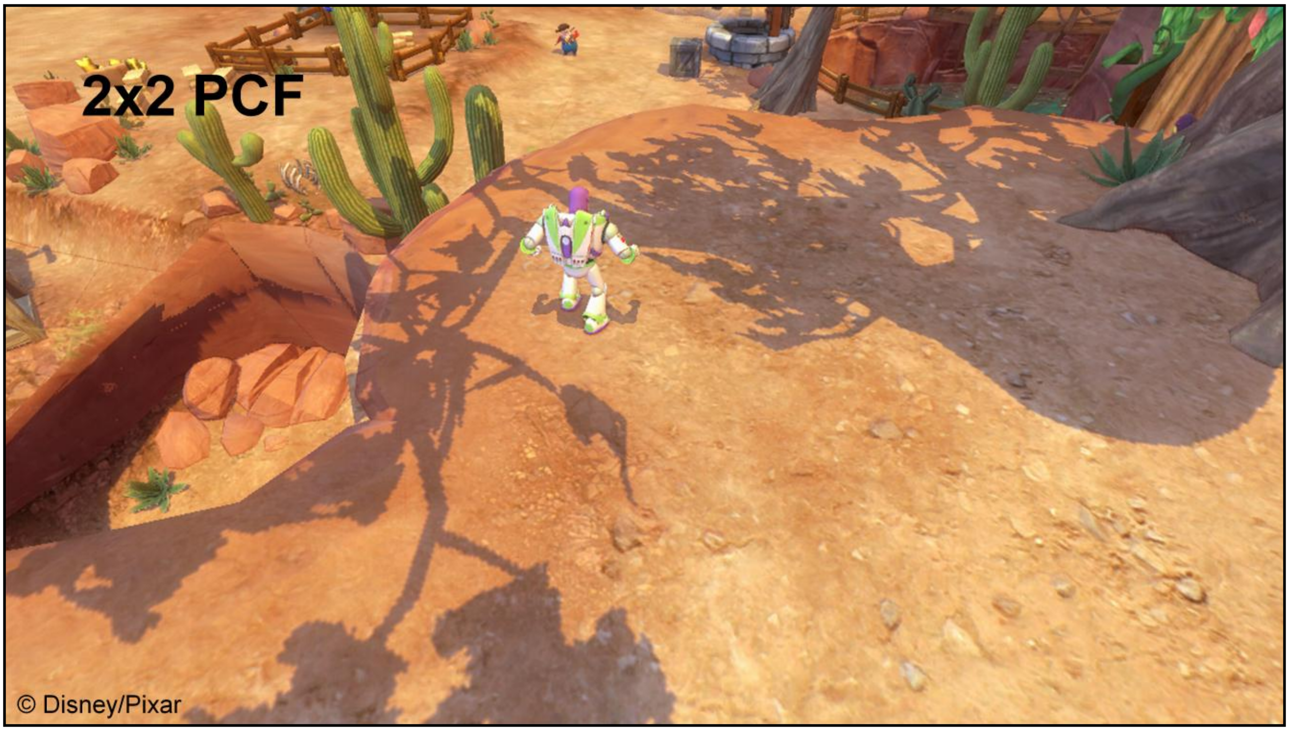
To show how it all works together, we will show a comparison

This is the final soft shadow look

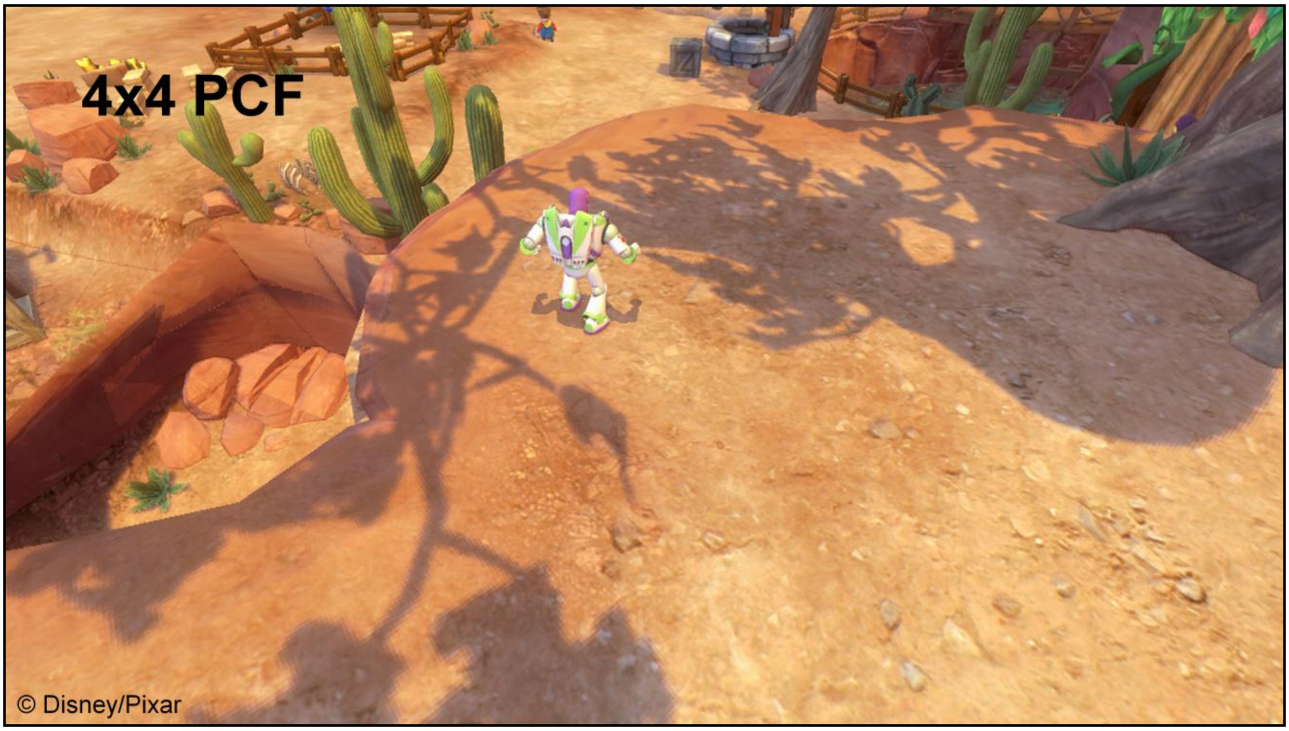


This is what it would look like using no filtering in the shadow map or in post

The shadow map aliasing is very large

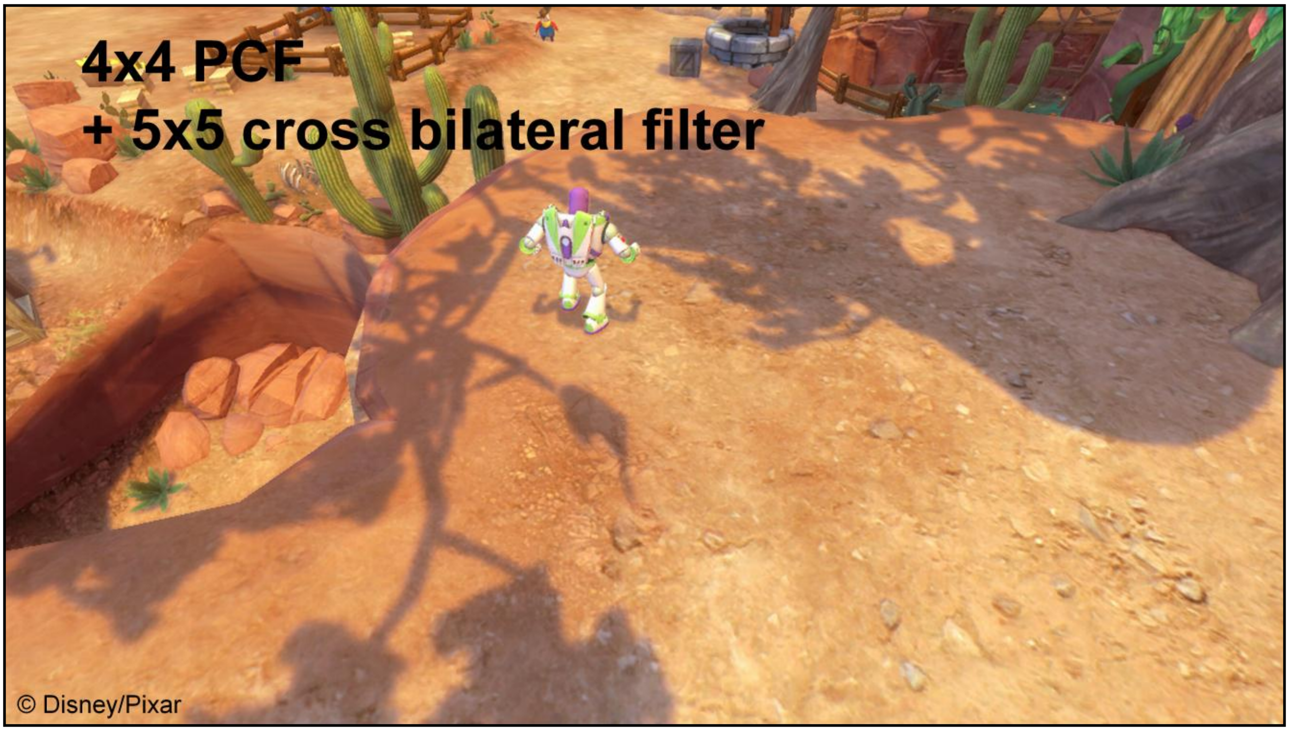


This now smoothes the edges, but the aliasing is still visible



The 4x4 PCF is just a regular 4x4 grid of Gaussian weighted samples

The aliasing is now removed, but the grid is now visible



4x4 PCF
+ 5x5 cross bilateral filter

© Disney/Pixar

The 5x5 cross bilateral filter removes the regular grid and makes the shadows very soft

Since the cross bilateral filter was already being used in SSAO, we just combined the shadow term with it

This means that this pass was added without any additional cost

Additional notes on 5x5 cross bilateral filter:

The filter itself is not very complicated

It is basically like a two pass 5x5 gaussian filter, but with a depth test on each pixel fetched

It rejects pixels that have a depth difference larger than a specified threshold

Since the colors can be bilinearly sampled, you only need three color samples and five depth samples per pass

There can be a mismatch between the color and depth samples, but this can be resolved by gathering the filter weights for the color samples

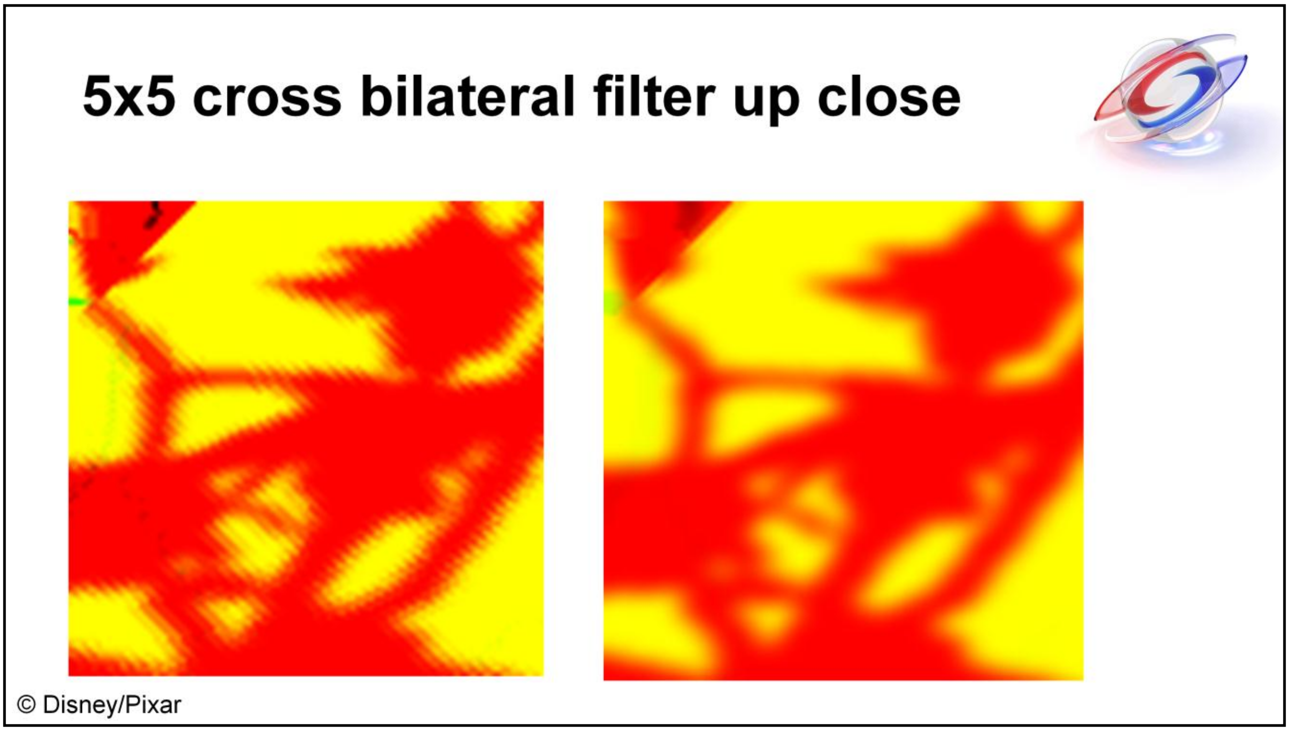
For each depth sample that fails the threshold test the color contribution for that pixel can be accurately removed from the bilinearly sampled color

It is possible to reduce the total number of samples per pass by combining the depth and color buffers in either the shadow or SSAO passes

You would need at a minimum of a 16bit depth and at least 8 bits per shadowed component

We had three components, so it did not fit a 32bit render target

I am sure someone else might find a solution to this in the future

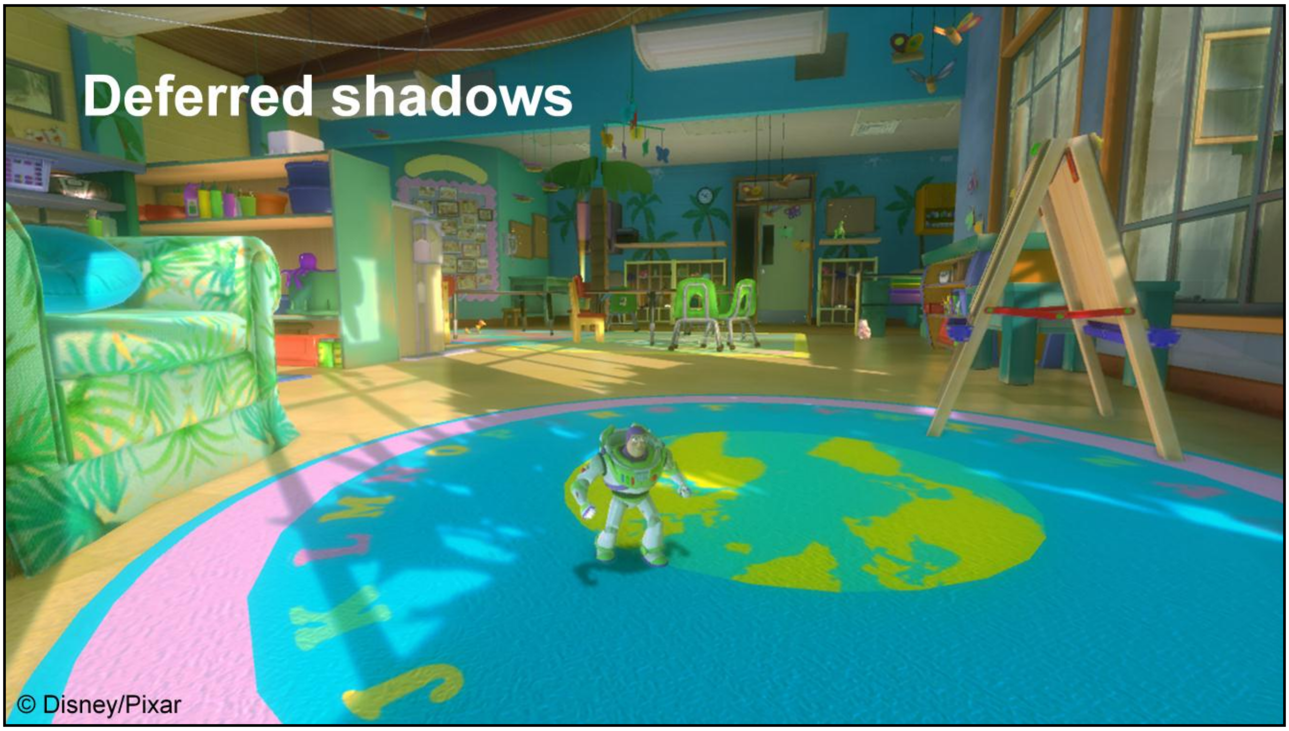


5x5 cross bilateral filter up close

© Disney/Pixar

This is what one part of the shadow mask in the previous screen looks like before and after the filter

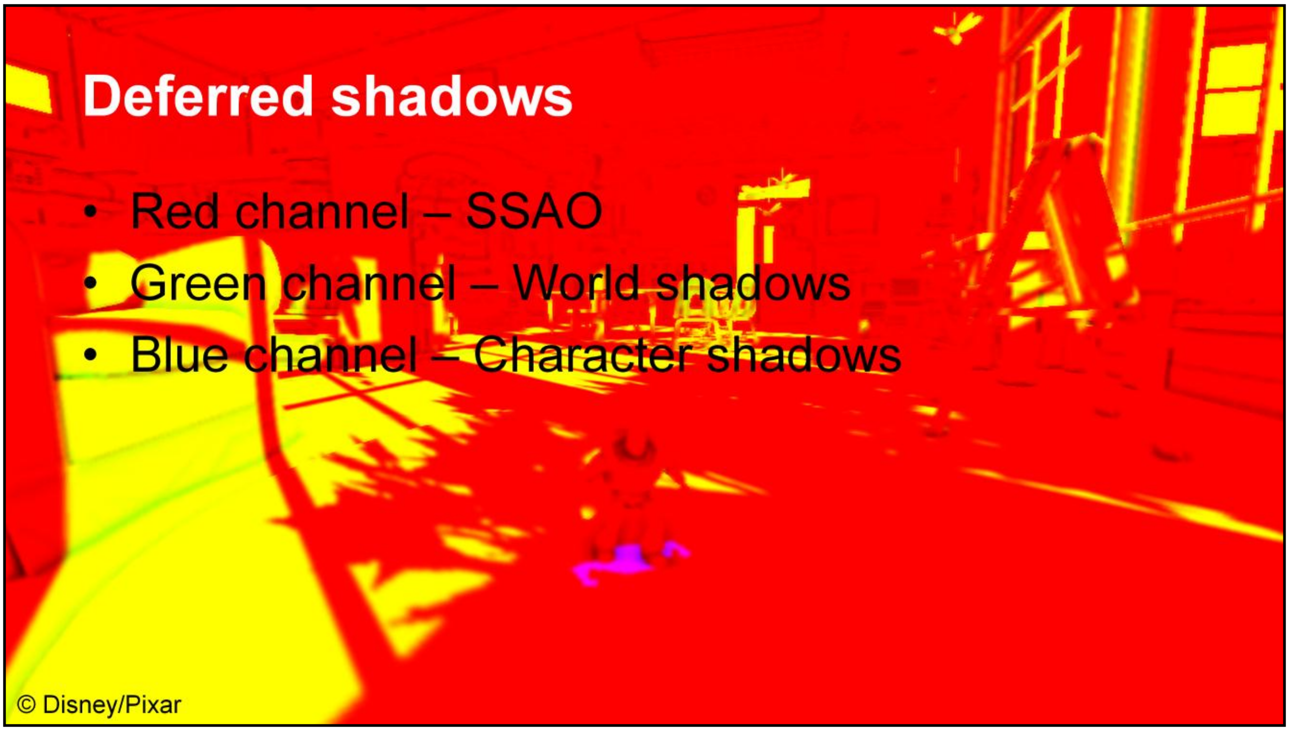
This filter makes the shadows very soft without the aliasing of the shadow map



Deferred shadows

© Disney/Pixar

To enable the post process filter for shadows, it needed to be deferred



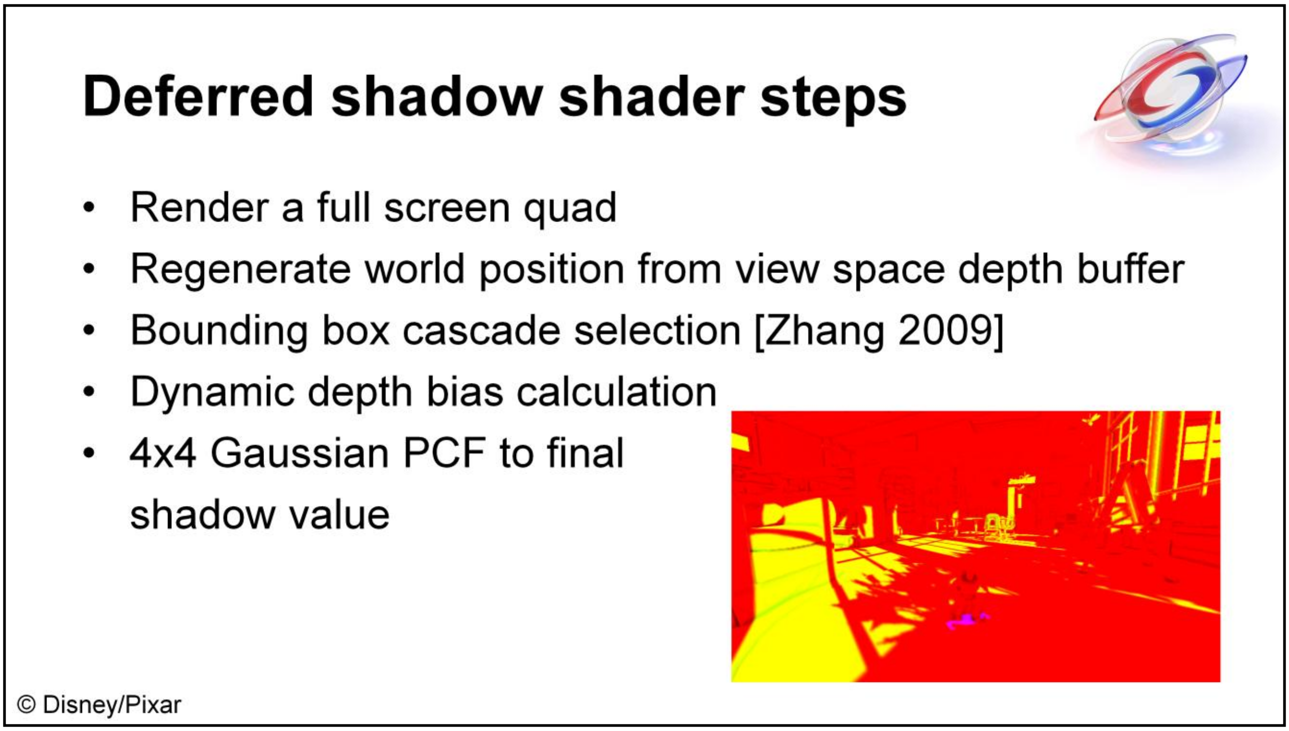
Deferred shadows

- Red channel – SSAO
- Green channel – World shadows
- Blue channel – Character shadows

© Disney/Pixar

All of these components are blurred together

The drop shadow for the main character used no PCF filtering, which looked good enough after the filter



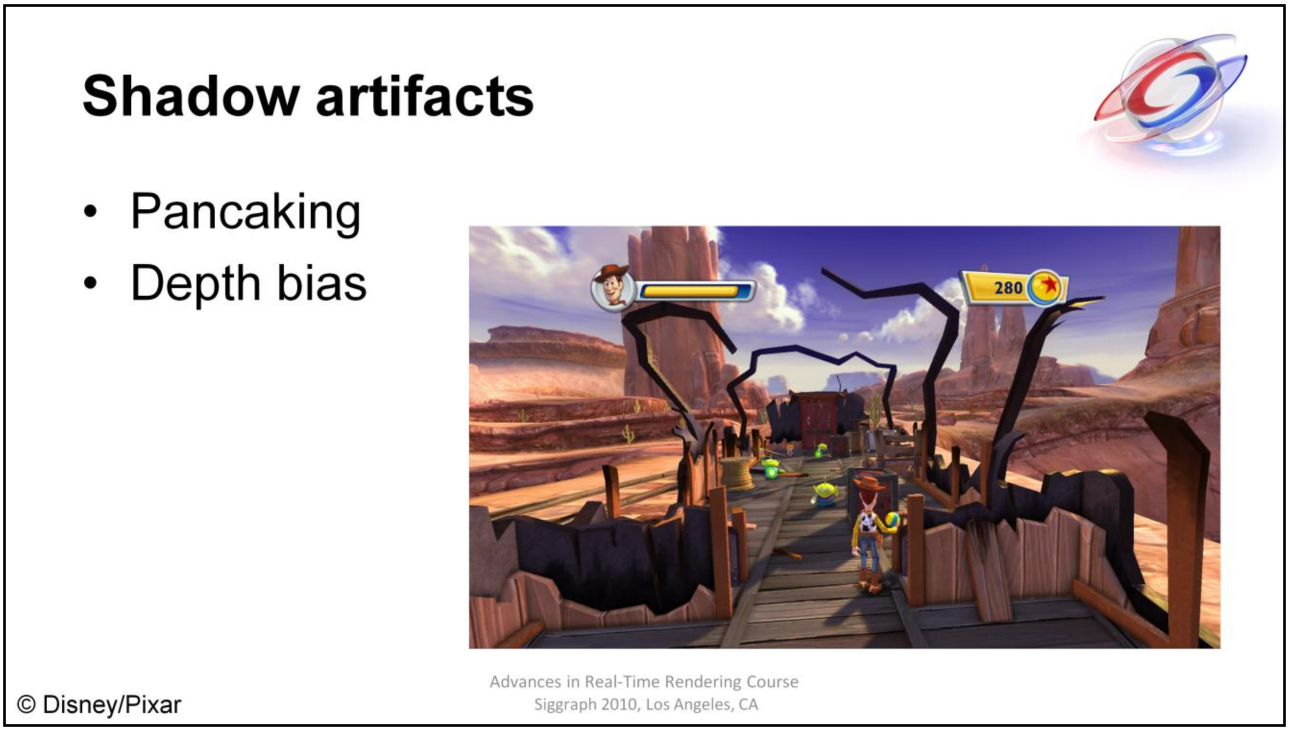
Deferred shadow shader steps

- Render a full screen quad
- Regenerate world position from view space depth buffer
- Bounding box cascade selection [Zhang 2009]
- Dynamic depth bias calculation
- 4x4 Gaussian PCF to final shadow value

These are the steps for the deferred pixel shader

There are both a lot of ALU operations and texture fetches

The texture fetches still took longer than the ALU, but not by much



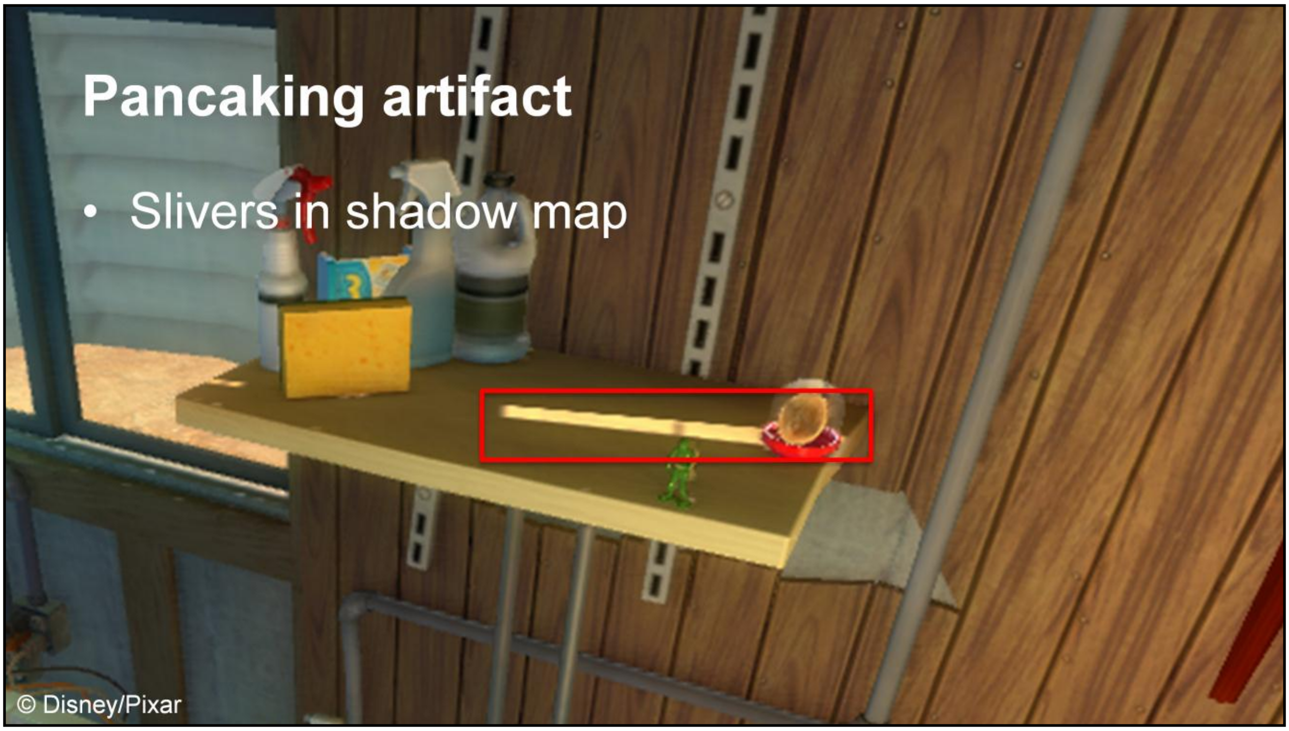
Shadow artifacts

- Pancaking
- Depth bias

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

© Disney/Pixar

Despite how good the shadows now look, there were a few artifacts we experienced



Pancaking artifact

- Slivers in shadow map

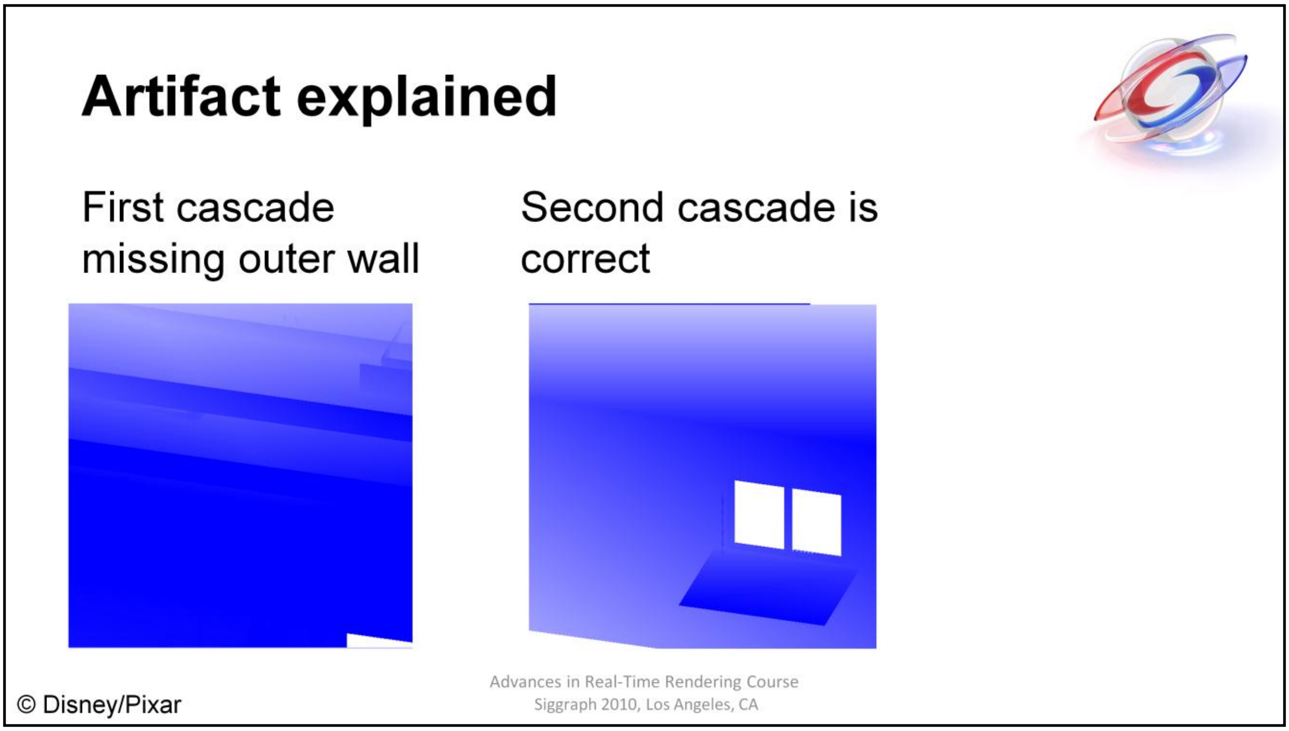
© Disney/Pixar

In many of our indoor scenes, this artifact quickly showed up

After a bit of investigation, we found out that this was caused by how we do pancaking

For an explanation of pancaking, take a look at “The Dark Art of Shadow Mapping” by Tuft

Pancaking enables each shadow cascade to have more precision by clamping the depth range based on the camera frustum



Artifact explained

First cascade missing outer wall

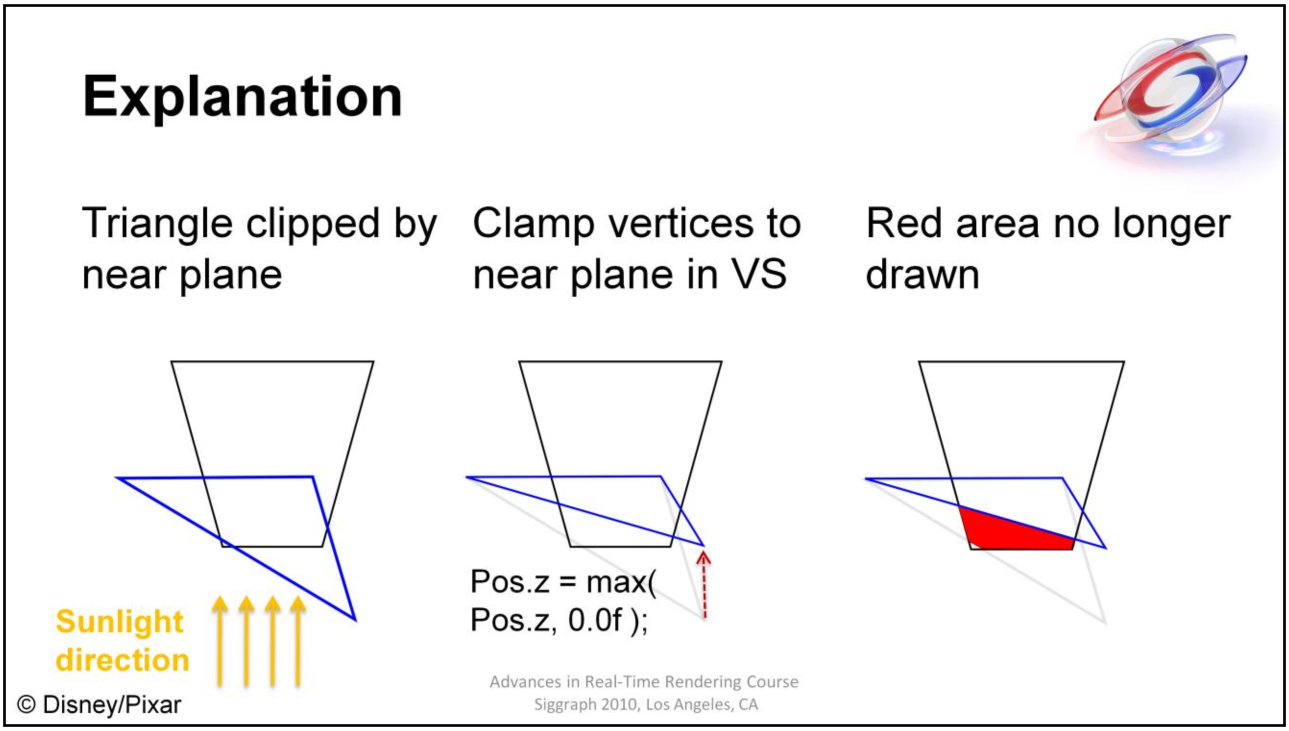
Second cascade is correct

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The sliver shows up because some geometry is getting clipped out in the first cascade

As you can see, the first cascade should show a wall like the second cascade



Explanation

Triangle clipped by near plane

Clamp vertices to near plane in VS

Red area no longer drawn

```
Pos.z = max(
Pos.z, 0.0f );
```

Sunlight direction

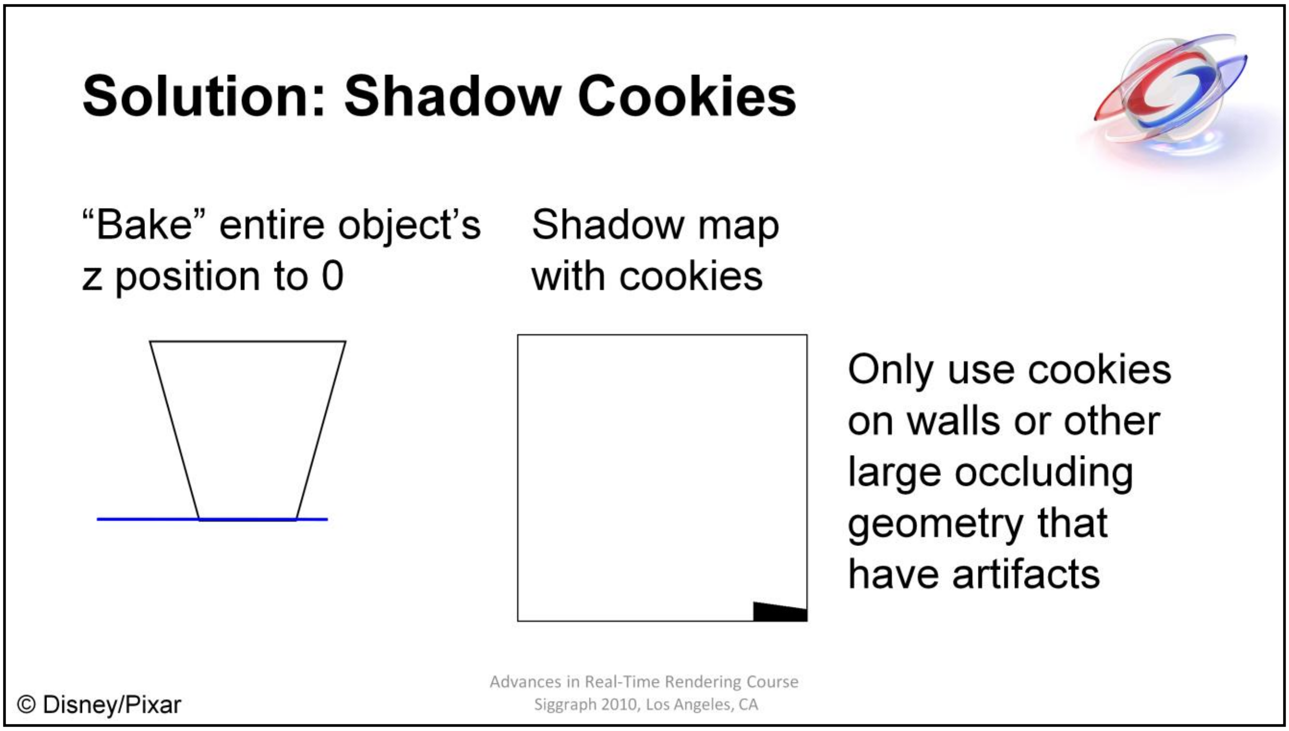
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

Pancaking has a problem where certain objects are clipped by the near plane of the sun light’s frustum

This can be removed by putting in the vertex shader the line shown in the middle figure

However, as you can see on the right figure that although the vertex is no longer clipped, the red region is now not covered by that triangle



Solution: Shadow Cookies

"Bake" entire object's z position to 0

Shadow map with cookies

Only use cookies on walls or other large occluding geometry that have artifacts

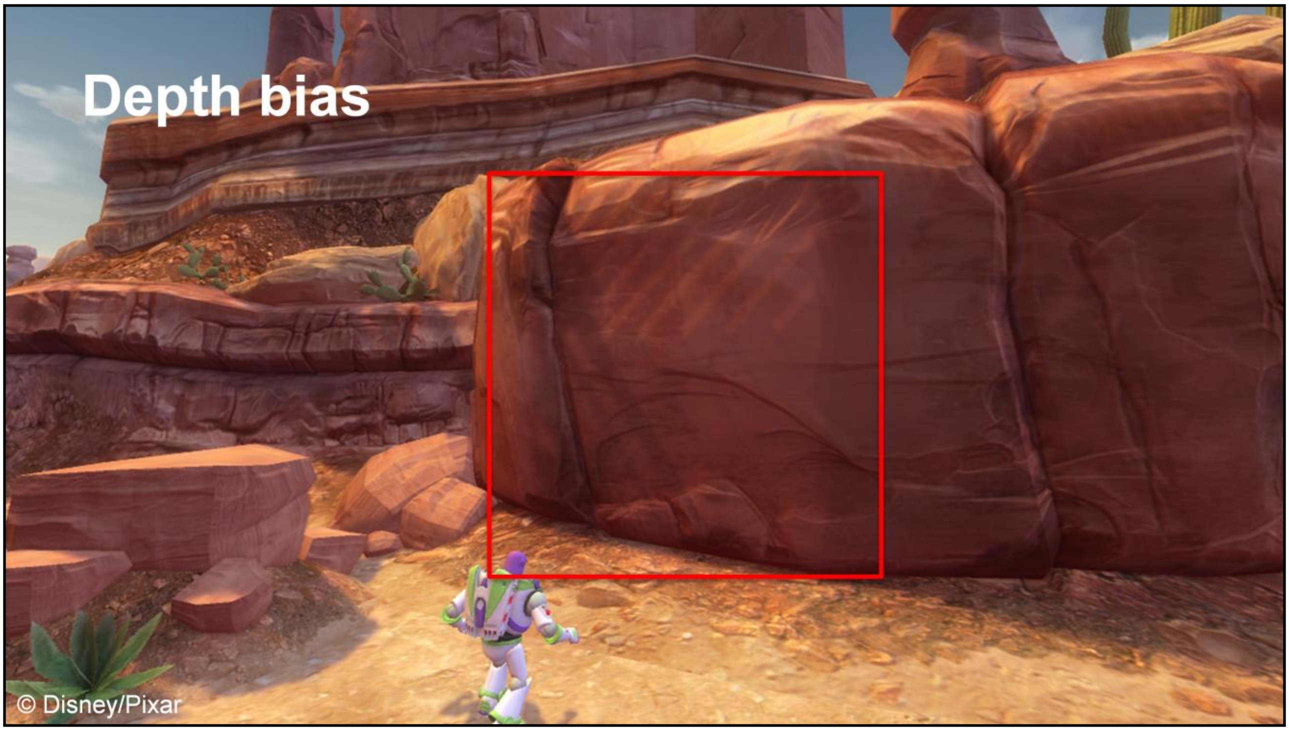
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The solution to this is to assign objects to use a shadow cookie

This "bakes" the entire object's z position to 0

This essentially removes any self shadowing for an object, so I suggest you only use it on walls or other large occluding geometry that have this artifact



Depth bias

© Disney/Pixar

An oft explained artifact is incorrect self shadowing for shadow mapping

Shown here is a face that has a striped pattern that immediately does not look correct

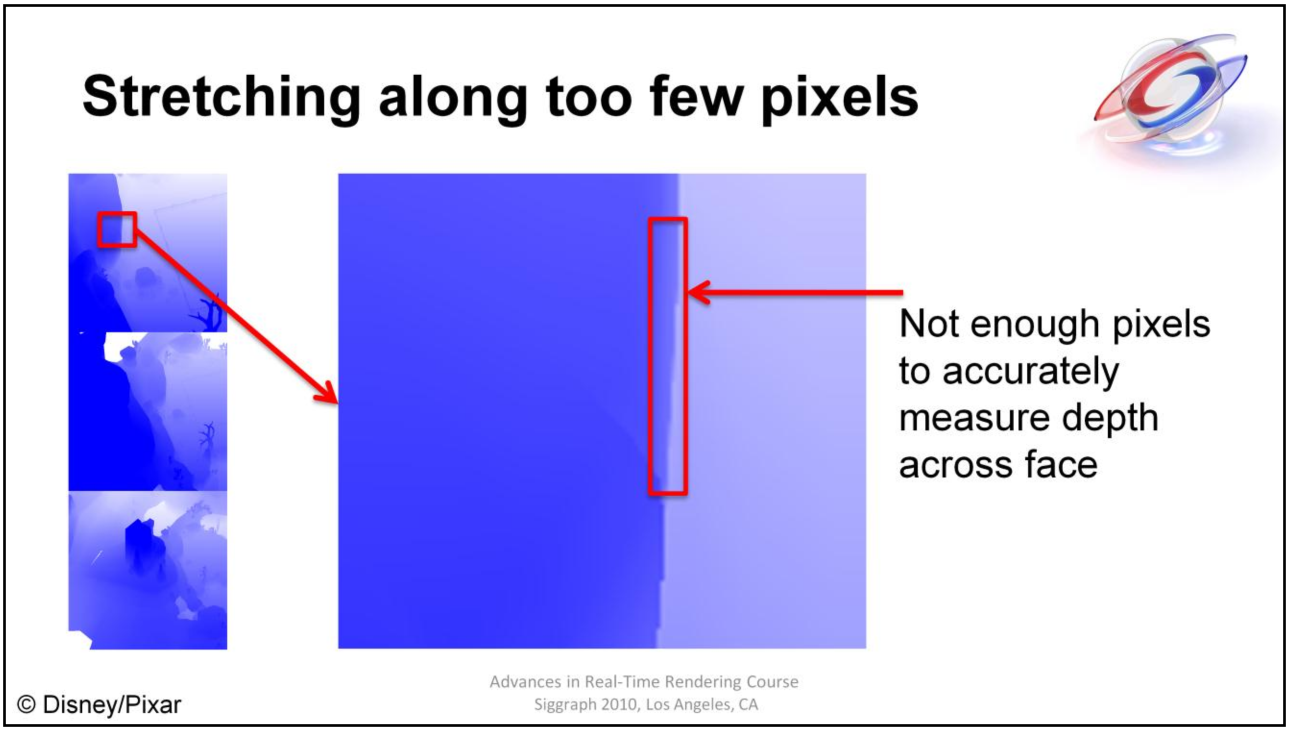
Even with a static depth bias this cannot be eliminated

Notes on slope scale depth bias:

For these surfaces, a slope scale depth bias should solve the problem

Unfortunately, the slope is too extreme to be removed

Console has no way to have a maximum bias, so we cannot use a very large value or else we experience more artifacts when rotating the camera



Stretching along too few pixels

Not enough pixels to accurately measure depth across face

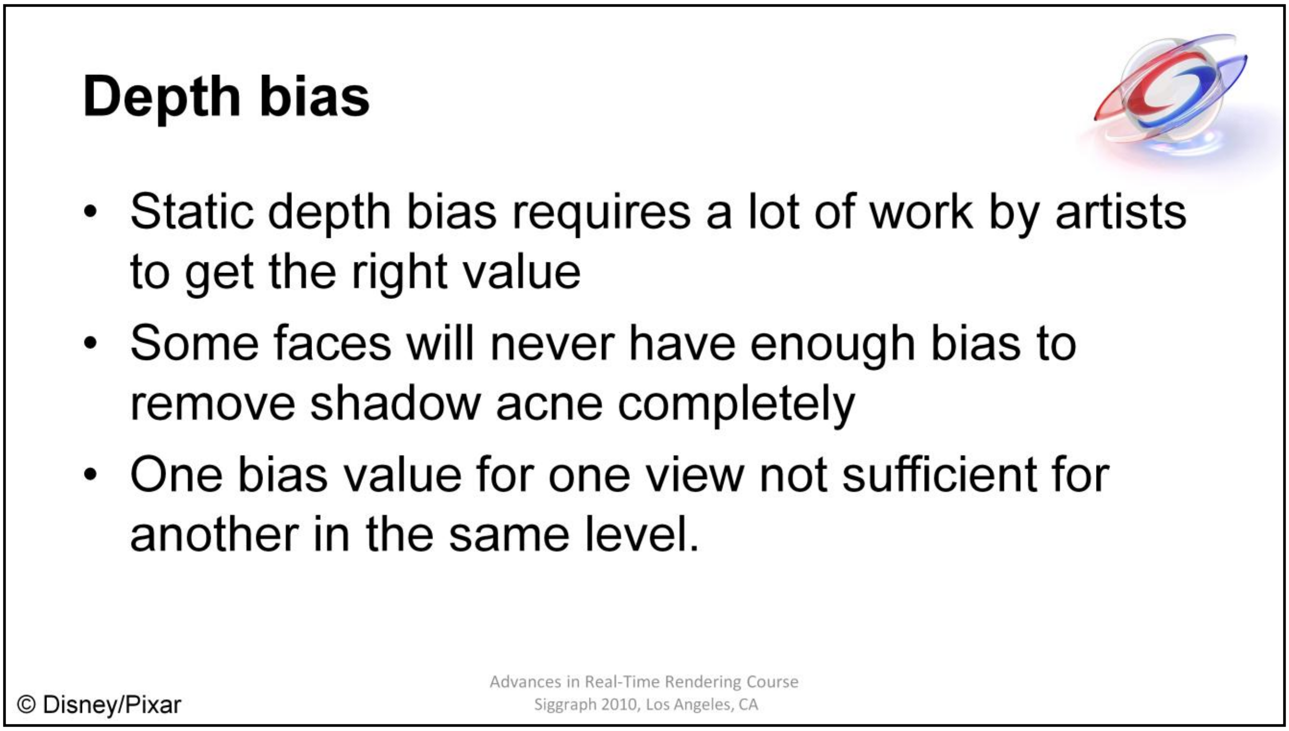
© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

This explains the problem with this face

In the shadow map, it is represented by only a few pixels

Therefore, there is no way to accurately measure depth across this face



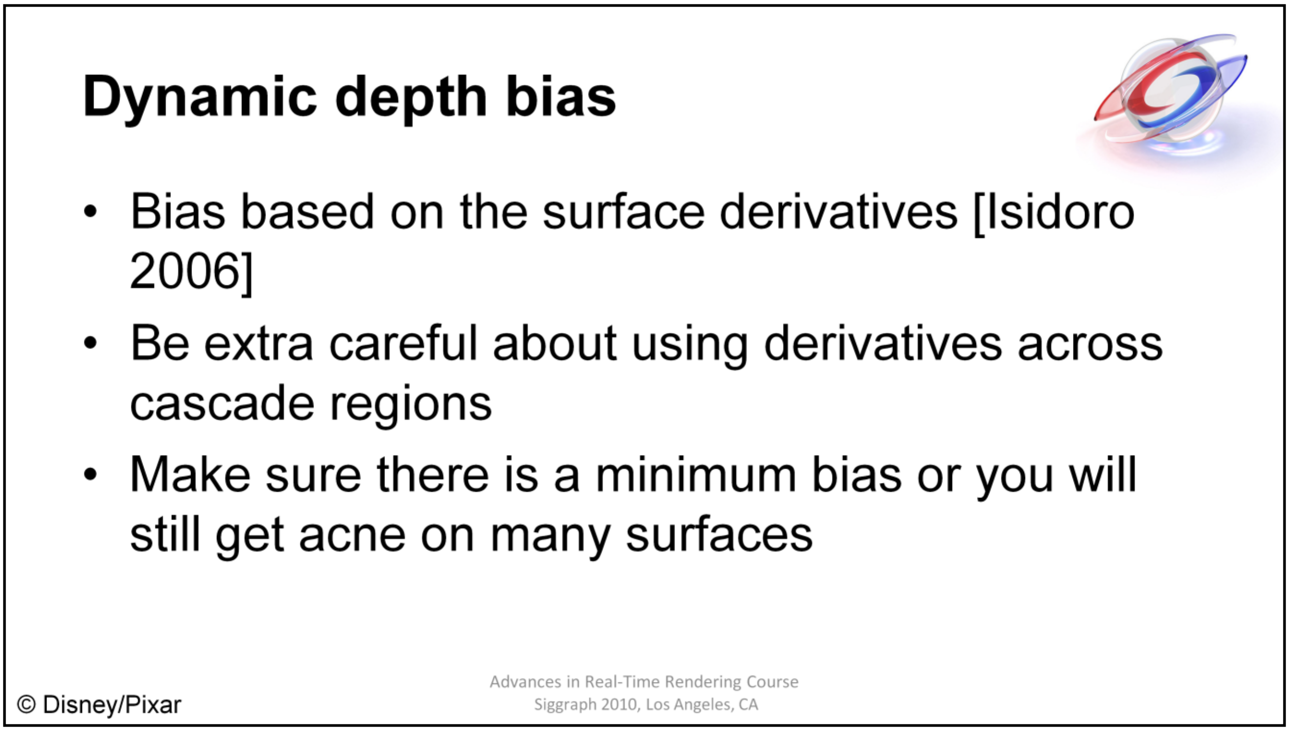
Depth bias

- Static depth bias requires a lot of work by artists to get the right value
- Some faces will never have enough bias to remove shadow acne completely
- One bias value for one view not sufficient for another in the same level.

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The static depth bias is not effective as it will not work for all scenes

Even if you get a good value in the general case, it still won’t work for extreme cases



Dynamic depth bias

- Bias based on the surface derivatives [Isidoro 2006]
- Be extra careful about using derivatives across cascade regions
- Make sure there is a minimum bias or you will still get acne on many surfaces

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

The solution is to have a dynamic depth bias based on the surface derivatives

Isidoro’s paper contains the sample code to get you started

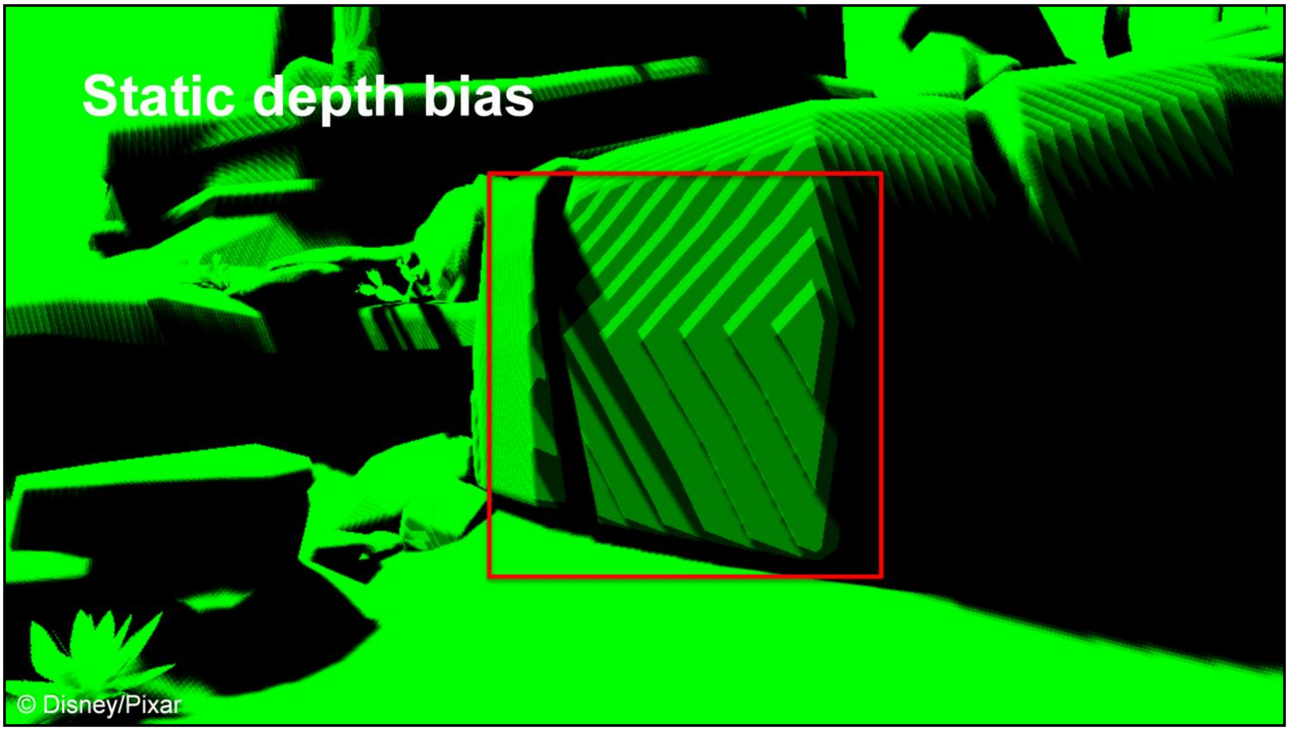
This has an added benefit that we no longer needed to specify a static depth bias so artists had one less thing to worry about

Notes on getting it to work:

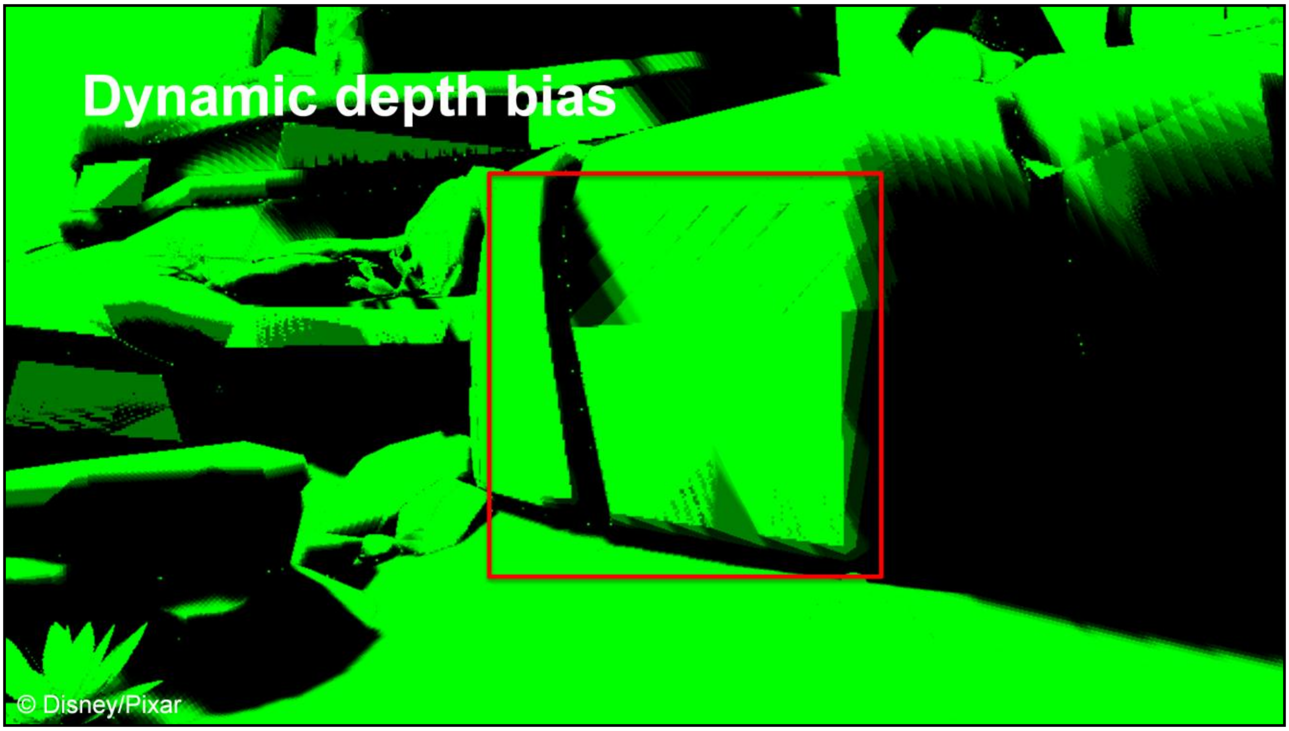
If your derivatives crossed cascade regions, we would invalidate the dynamic bias and instead use a scaled static bias based on the current cascade for that pixel

Also, for very level surfaces, the dynamic bias was still not sufficient, so we clamped the bias to a minimum value

In the opposite case, some extreme angles produced a very large bias, so that was also clamped to a maximum value



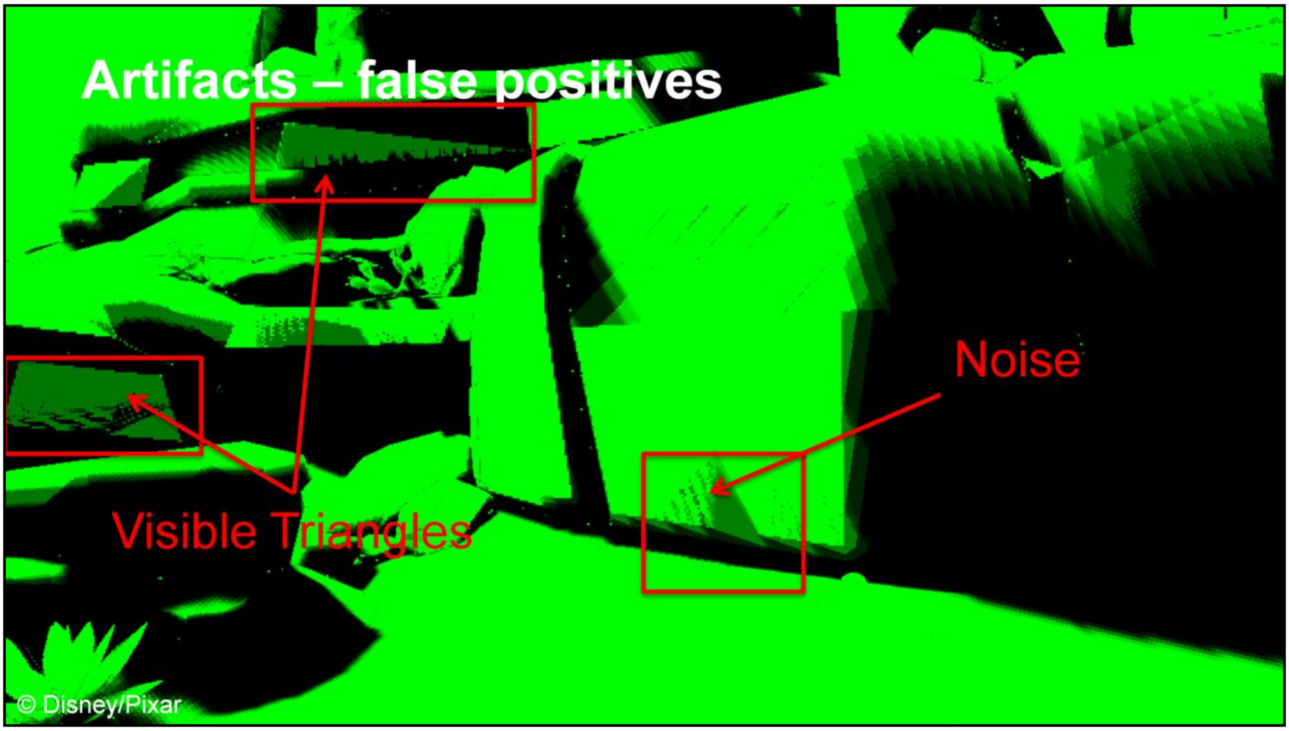
To give you a reference, this is what the shadow mask looks like using the static depth bias



Dynamic depth bias

© Disney/Pixar

This is what it looks like with the dynamic depth bias



Artifacts – false positives

© Disney/Pixar

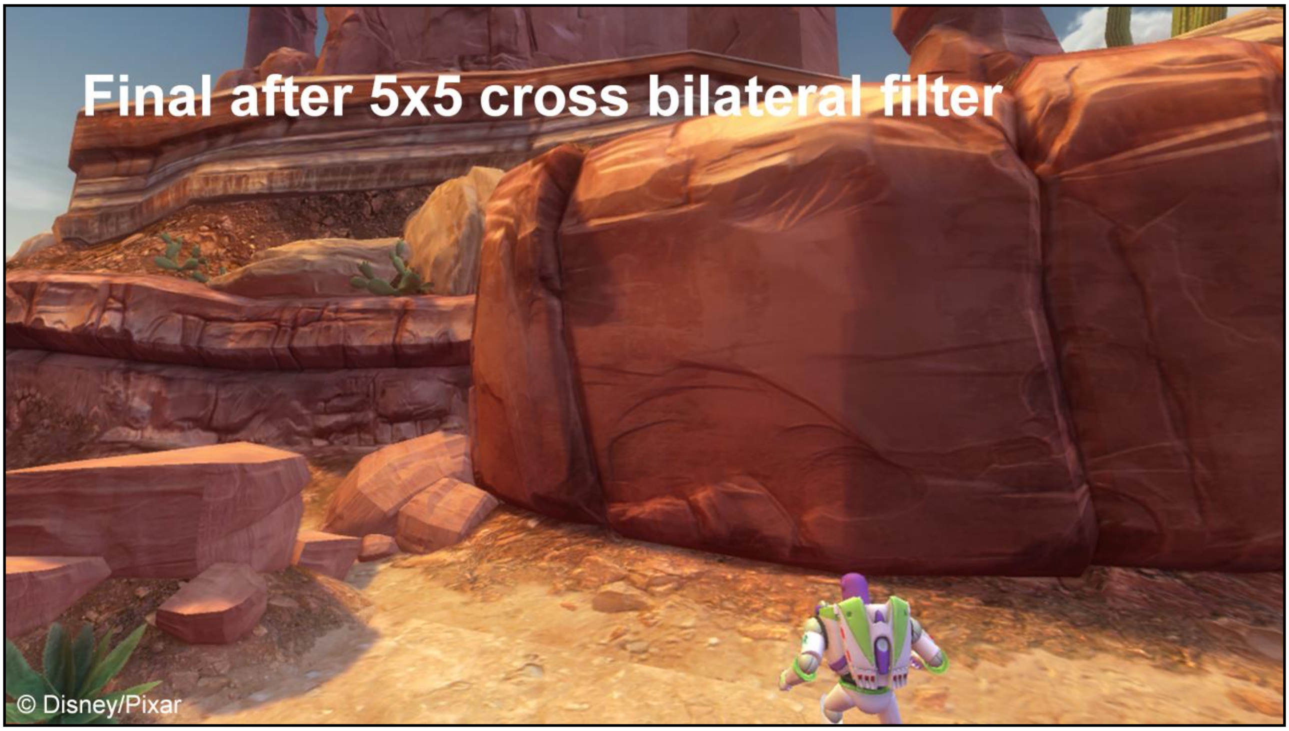
Unfortunately, it does not work perfectly

First, the stiped lines are almost removed, but still visible

Next, noise is introduced on some surfaces

Also, this noise can further affect certain surfaces and make triangles visible that should be completely black

By examining those triangles, the shadow value is still close to black, so it might not be visible in the final scene



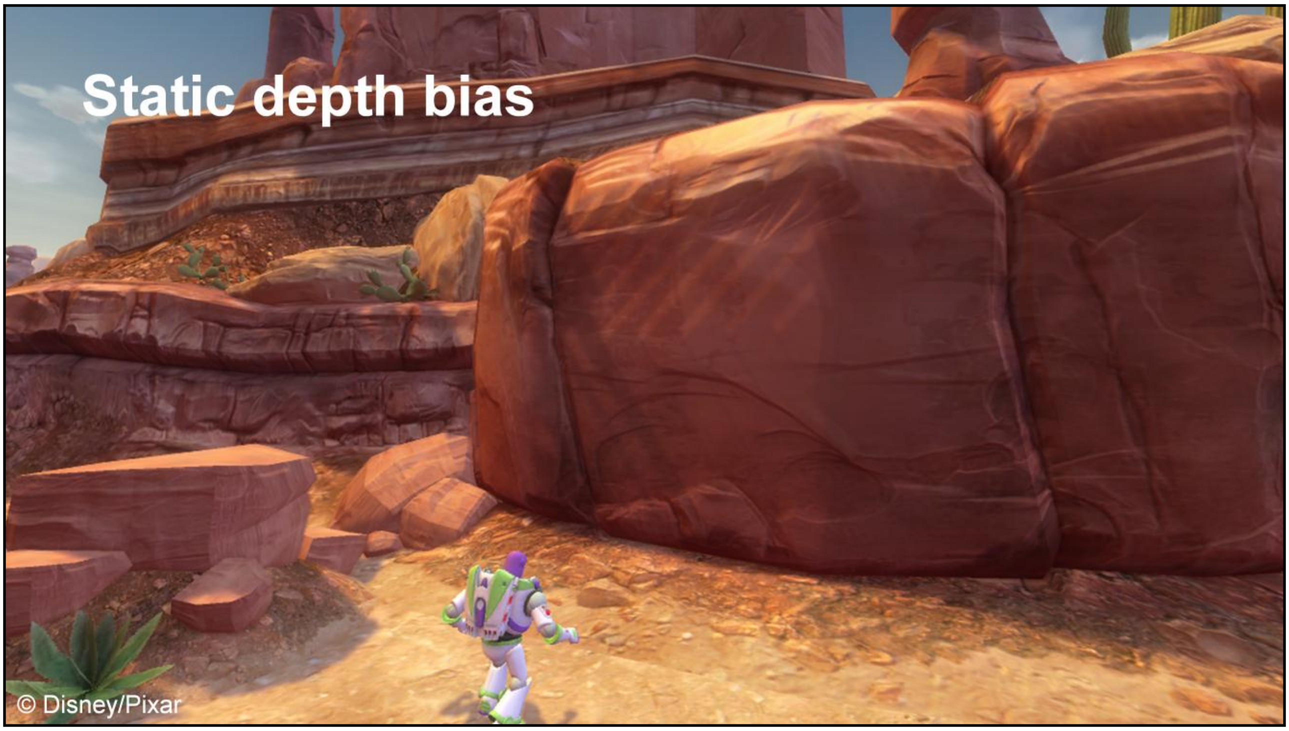
Final after 5x5 cross bilateral filter

© Disney/Pixar

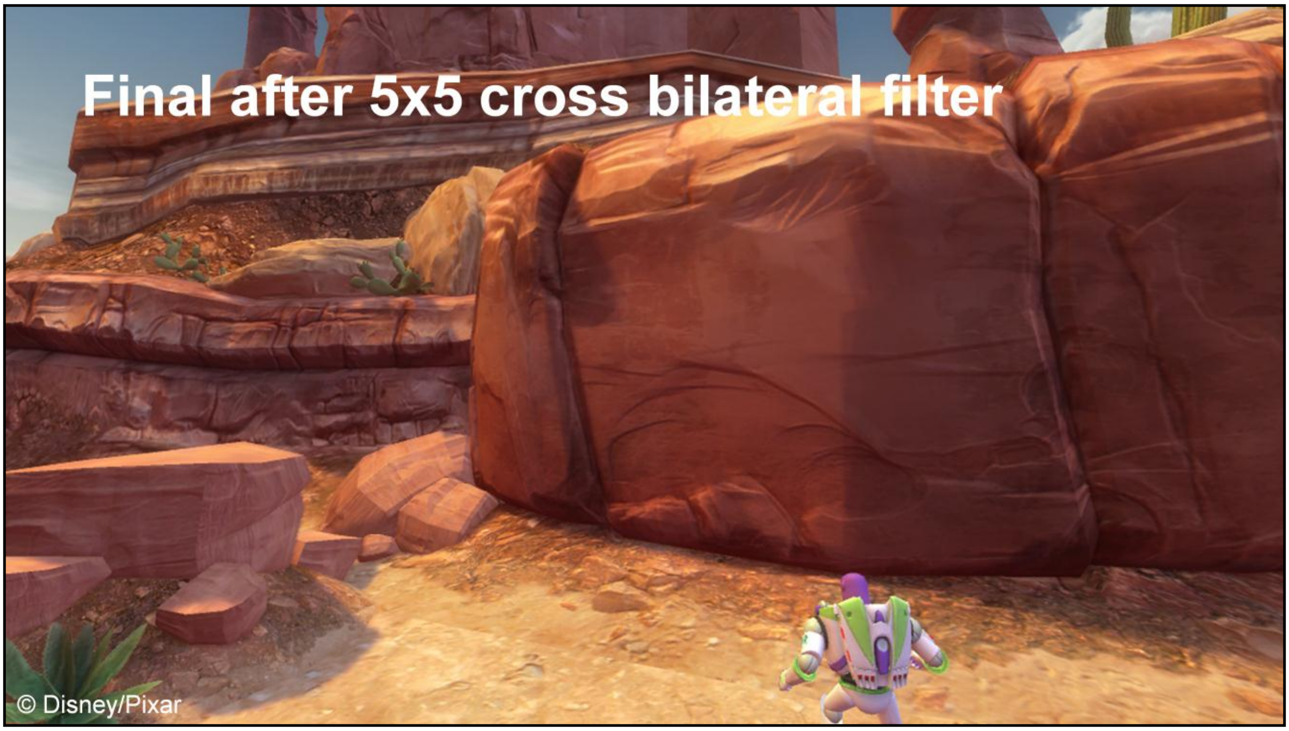
As you can see, those artifacts don’t really show up

The 5x5 cross bilateral filter removes a lot of the noise and the rest of the striped lines

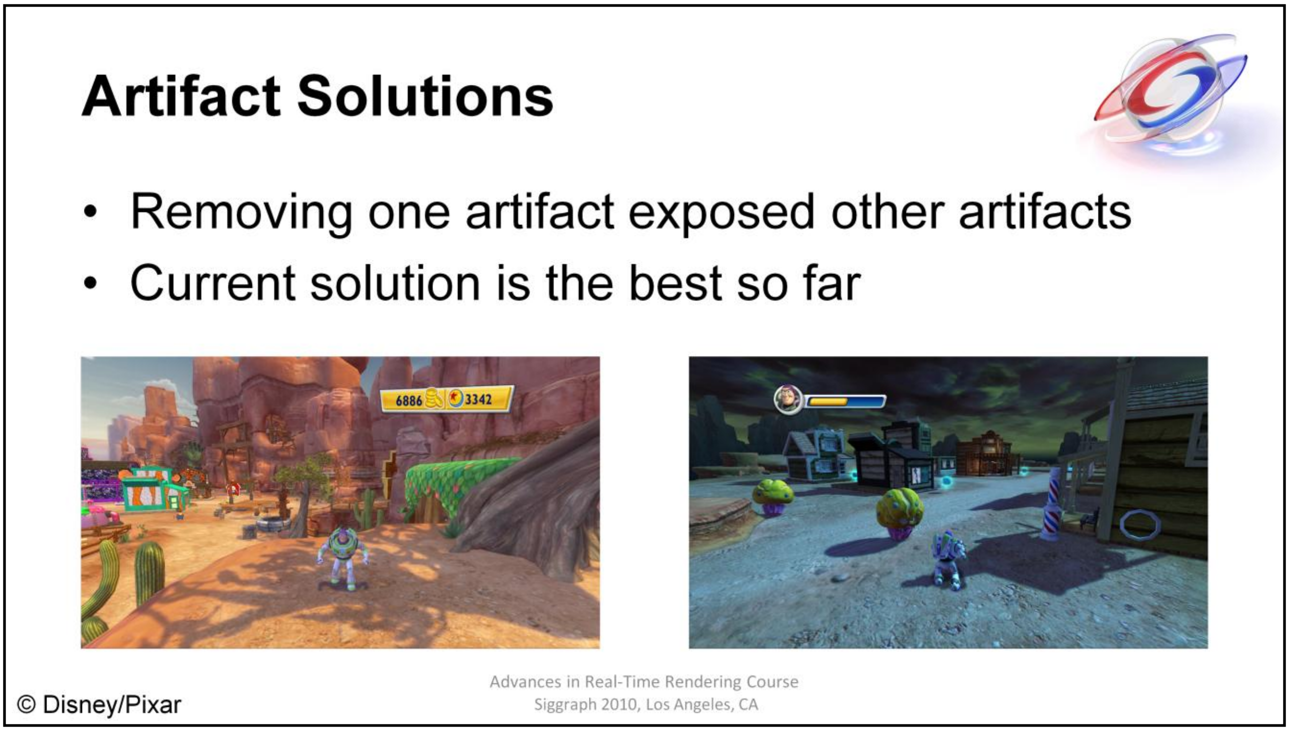
Since the shadow is just a mask, then the visible triangles really don’t show up in the end



To give you reference to what it was before, this is what it looks like with the static depth bias



This is what is looks like with the dynamic depth bias and the 5x5 cross bilateral filter



Artifact Solutions

- Removing one artifact exposed other artifacts
- Current solution is the best so far

© Disney/Pixar

Advances in Real-Time Rendering Course
Siggraph 2010, Los Angeles, CA

There are still cases in which the dynamic bias still doesn’t work well

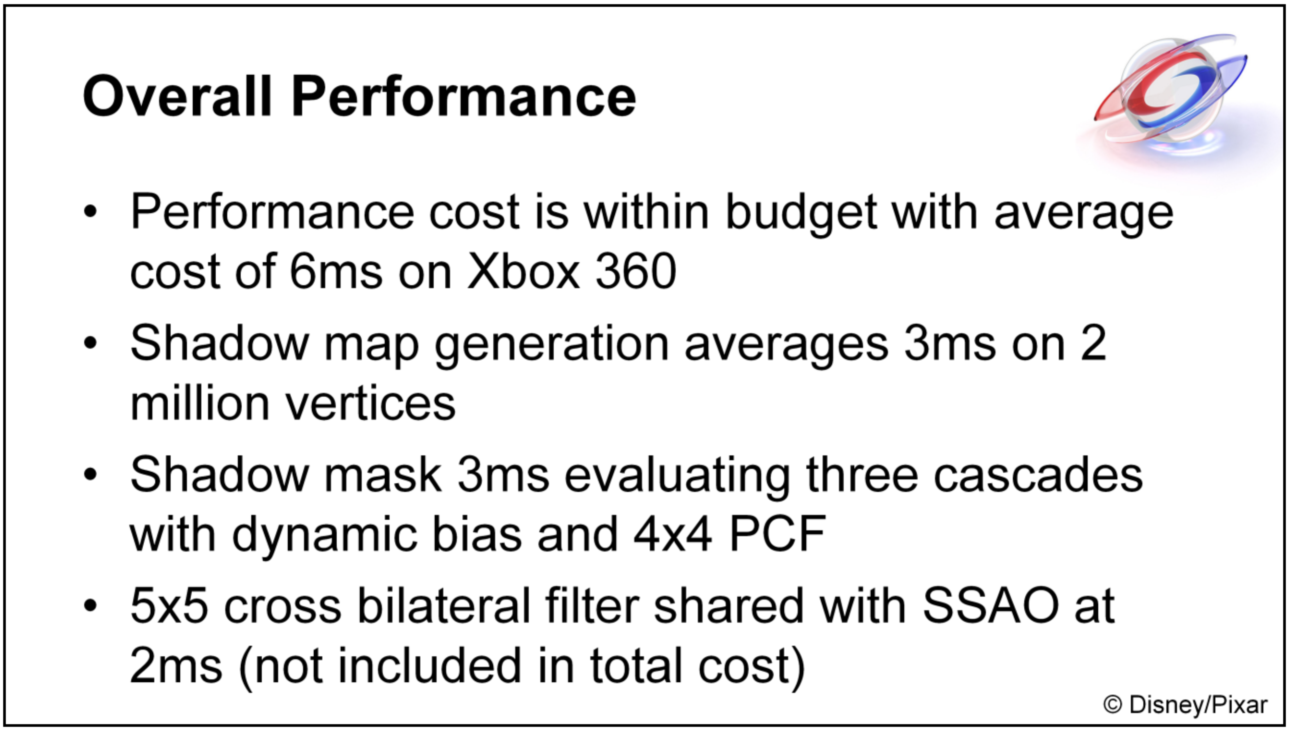
Some of the triangles and noise are visible in extreme cases

It is possible to reduce the noise or the visible triangles, but it would expose the striped lines more

Therefore, this is the best solution so far

It still on average gave better results than the static depth bias

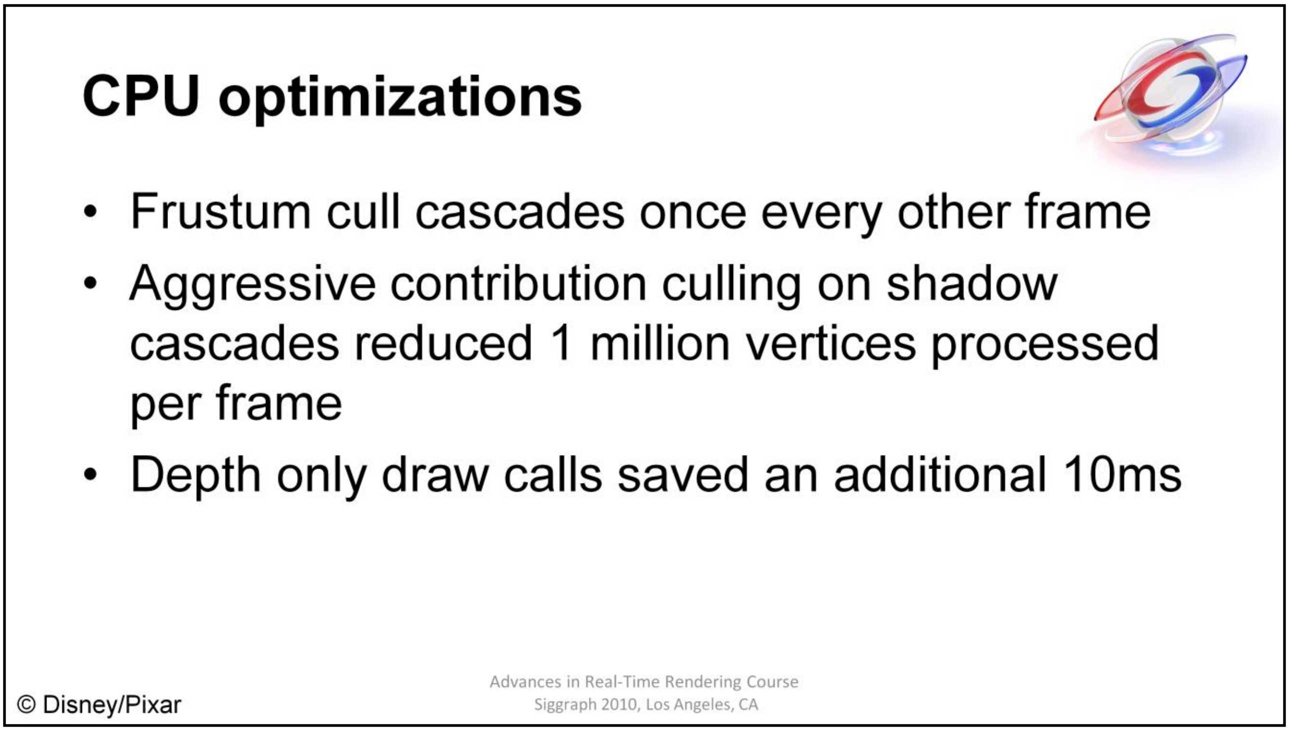
Tweaking the depth bias was no longer necessary, so there was no more peter panning or significant shadow acne



Overall Performance

- Performance cost is within budget with average cost of 6ms on Xbox 360
- Shadow map generation averages 3ms on 2 million vertices
- Shadow mask 3ms evaluating three cascades with dynamic bias and 4x4 PCF
- 5x5 cross bilateral filter shared with SSAO at 2ms (not included in total cost)

© Disney/Pixar



CPU optimizations

- Frustum cull cascades once every other frame
- Aggressive contribution culling on shadow cascades reduced 1 million vertices processed per frame
- Depth only draw calls saved an additional 10ms

Additional notes not mentioned in talk:

Notes on frustum culling:

We only frustum cull cascades 1 and 2 on one frame and then 1 and 3 on the next

Each time a cascade would perform frustum culling, it would cache an array of object pointers to use for the next frame

It is possible that objects could be deleted between frames, so we need to verify the object exists before rendering

The rendering is on another thread, so it is still possible for deleted objects to still be rendered resulting in a crash

The solution was to double buffer the garbage collection list and delete the objects with a one frame delay

The cache needs to be invalidated any time a level changes or when the camera cuts to another location

Notes on contribution culling:

Already used contribution culling for the main camera.

Instead of clipping out if it was smaller than 1 pixel, we clip if it is smaller than 5, 15, and 25 pixels for each successive cascade

Objects would clip out between cascades, but they were already too small anyways to notice

Notes on depth only draw calls:

Most high performance engines should already have this

If you don’t, then this should be very important

In scenes of over 3 million vertices, it saved 10ms on the render thread and 0.5ms on the GPU

Adding Occlusion culling:

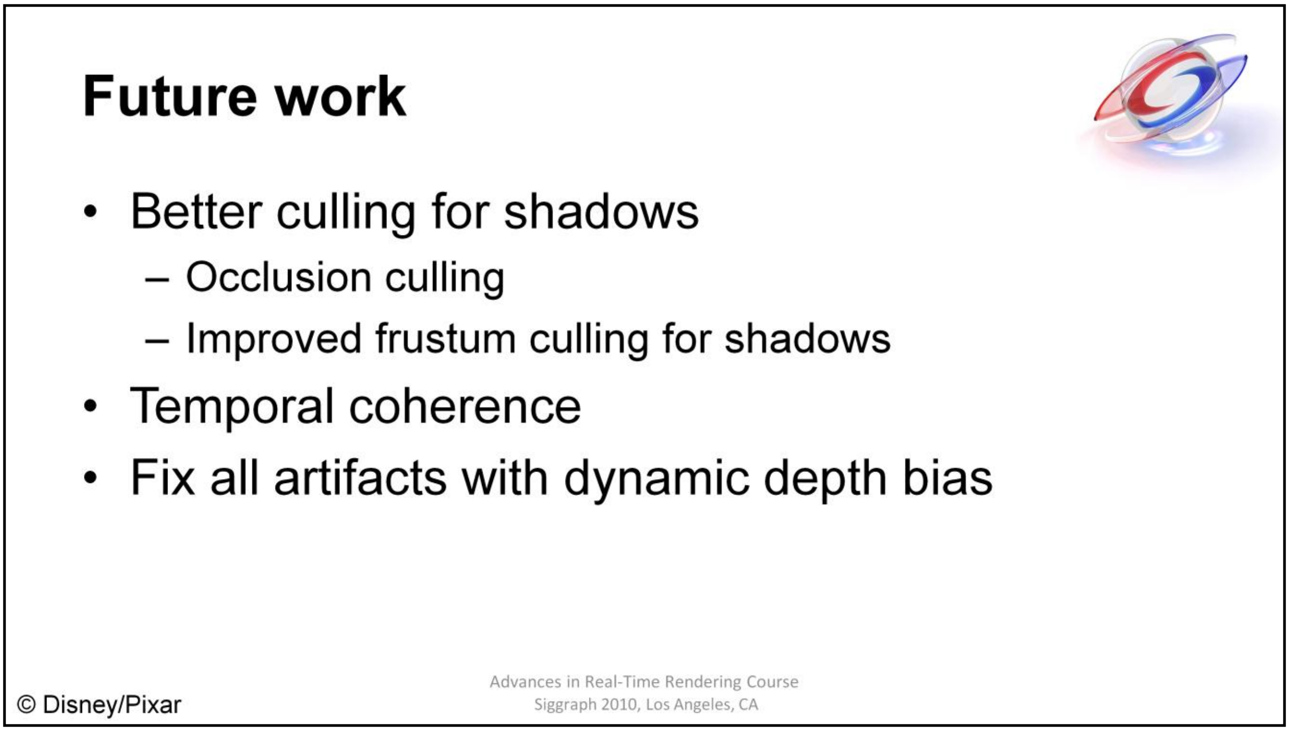
This is also a prime configuration for occlusion culling for the main scene on the GPU

After the depth pass, send out the occlusion queries

After the SSAO and shadow passes run, then the queries should already be returned to use when rendering the main scene

After writing the depth only draw calls, another engineer added occlusion culling in a day

PS3 and 360 handle occlusion queries differently



Future work

- Better culling for shadows
  - Occlusion culling
  - Improved frustum culling for shadows
- Temporal coherence
- Fix all artifacts with dynamic depth bias

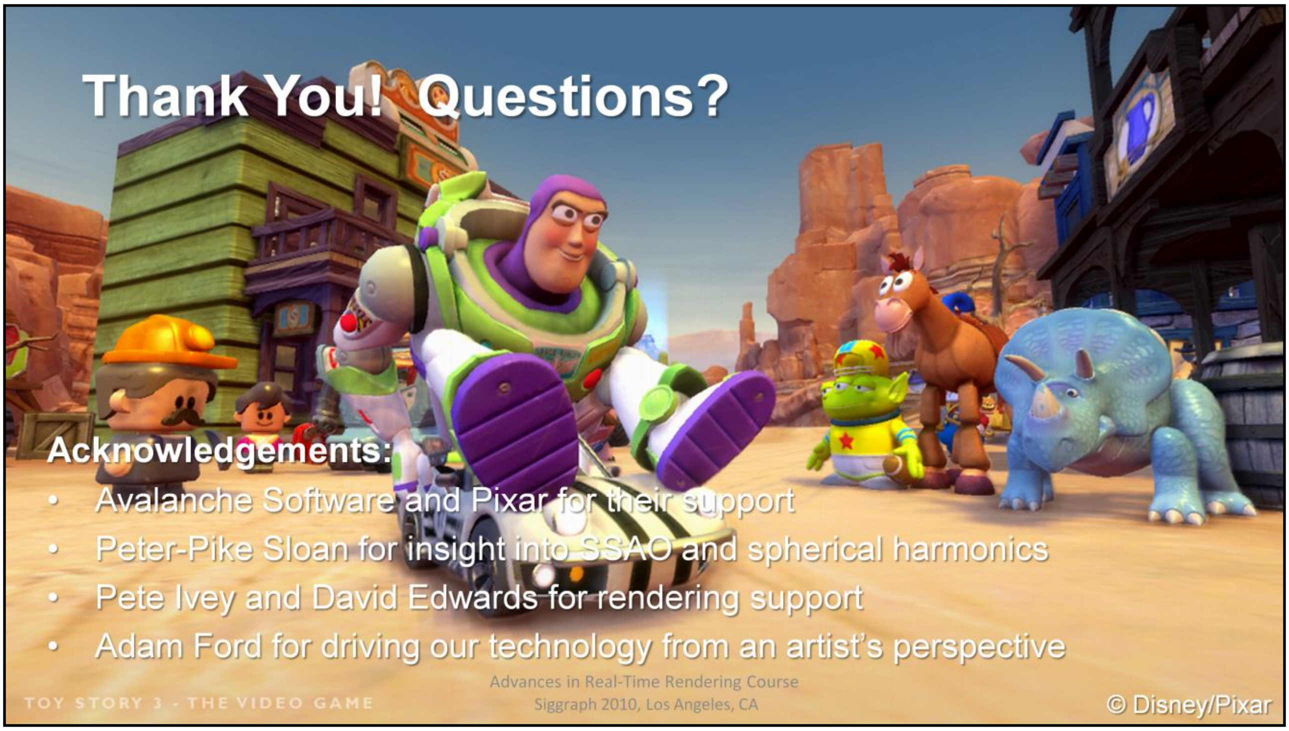
The shadow map cost can be reduced by having better culling for the shadows

Temporal coherence for shadows could be used to reuse data between frames

The dynamic depth bias is not perfect, so more work might improve it

Lastly, It is possible to replace the 4x4 Gaussian PCF with another sampling scheme

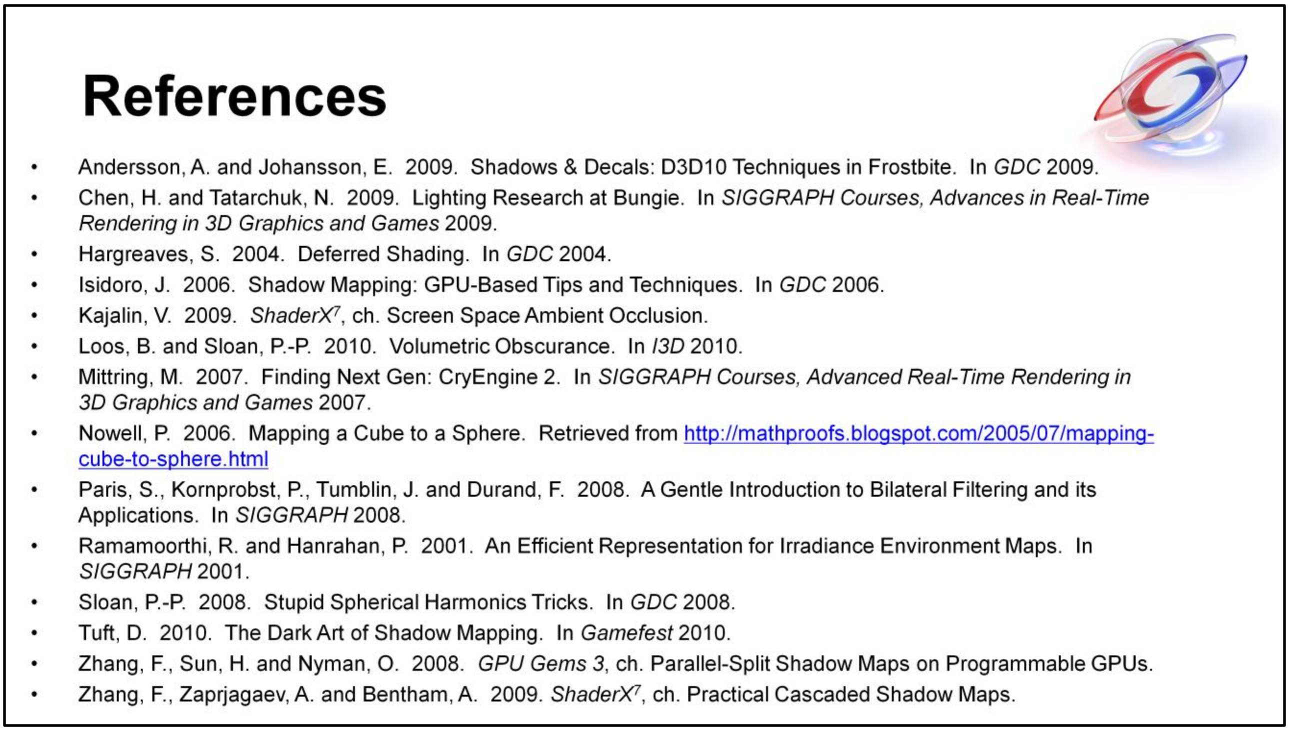
Since the 5x5 cross bilateral filter works great in removing noise, a noisy PCF pattern might give better results



Thank You! Questions?

Acknowledgements:

- Avalanche Software and Pixar for their support
- Peter-Pike Sloan for insight into SSAO and spherical harmonics
- Pete Ivey and David Edwards for rendering support
- Adam Ford for driving our technology from an artist's perspective



References

- Andersson, A. and Johansson, E. 2009. Shadows & Decals: D3D10 Techniques in Frostbite. In *GDC* 2009.
- Chen, H. and Tatarchuk, N. 2009. Lighting Research at Bungie. In *SIGGRAPH Courses, Advances in Real-Time Rendering in 3D Graphics and Games* 2009.
- Hargreaves, S. 2004. Deferred Shading. In *GDC* 2004.
- Isidoro, J. 2006. Shadow Mapping: GPU-Based Tips and Techniques. In *GDC* 2006.
- Kajalin, V. 2009. *ShaderX*^{7}, ch. Screen Space Ambient Occlusion.
- Loos, B. and Sloan, P.-P. 2010. Volumetric Obscurance. In *I3D* 2010.
- Mittring, M. 2007. Finding Next Gen: CryEngine 2. In *SIGGRAPH Courses, Advanced Real-Time Rendering in 3D Graphics and Games* 2007.
- Nowell, P. 2006. Mapping a Cube to a Sphere. Retrieved from http://mathproofs.blogspot.com/2005/07/mapping-cube-to-sphere.html
- Paris, S., Kornprobst, P., Tumblin, J. and Durand, F. 2008. A Gentle Introduction to Bilateral Filtering and its Applications. In *SIGGRAPH* 2008.
- Ramamoorthi, R. and Hanrahan, P. 2001. An Efficient Representation for Irradiance Environment Maps. In *SIGGRAPH* 2001.
- Sloan, P.-P. 2008. Stupid Spherical Harmonics Tricks. In *GDC* 2008.
- Tuft, D. 2010. The Dark Art of Shadow Mapping. In *Gamefest* 2010.
- Zhang, F., Sun, H. and Nyman, O. 2008. *GPU Gems 3*, ch. Parallel-Split Shadow Maps on Programmable GPUs.
- Zhang, F., Zaprjagaev, A. and Bentham, A. 2009. *ShaderX*^{7}, ch. Practical Cascaded Shadow Maps.