
Rendering Fur in
*Life of Pi*
Ivan Neulander
Google
Toshi Kato
Kevin Beason
Rhythm & Hues Studios


Rendering Fur in
Life of Pi
Ivan Neulander
Google
Toshi Kato
Kevin Beason
Rhythm & Hues Studios

# Rendering Fur in *Life of Pi*

Ivan Neulander
Google

Toshi Kato
Kevin Beason
Rhythm & Hues Studios

# Over the years...

# Over the years...

Coca Cola Polar Bears (1993-1996)

# Over the years...

# Over the years...

Cats & Dogs (2001)

# Over the years...

# Over the years...

Garfield, Garfield 2 (2004, 2006)

Rhythm&Hues

# Over the years...

# Over the years...

Chronicles of Narnia (2005)


Over the years...
Rhythm&Hues

# Over the years...

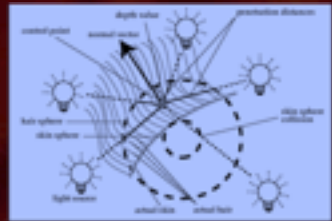
Life of Pi (2012)

# Technical Advancements for Life of Pi

# Technical Advancements for Life of Pi

1. Hair Shading
   - Extensive use of area lights, ray tracing

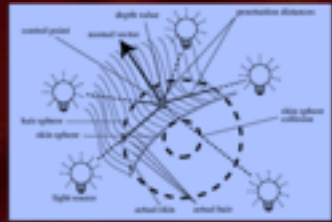
# Technical Advancements for Life of Pi

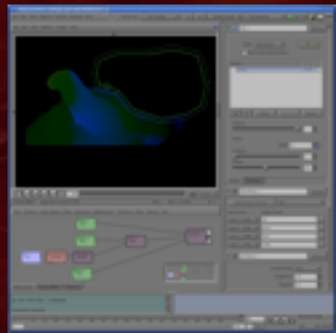


1. Hair Shading
   - Extensive use of area lights, ray tracing
2. Renderer Optimizations
   - Reduced render times & maintained quality

# Technical Advancements for Life of Pi

1. Hair Shading
   - Extensive use of area lights, ray tracing
2. Renderer Optimizations
   - Reduced render times & maintained quality
3. Postprocessing
   - Moved operations from renderer into 2D

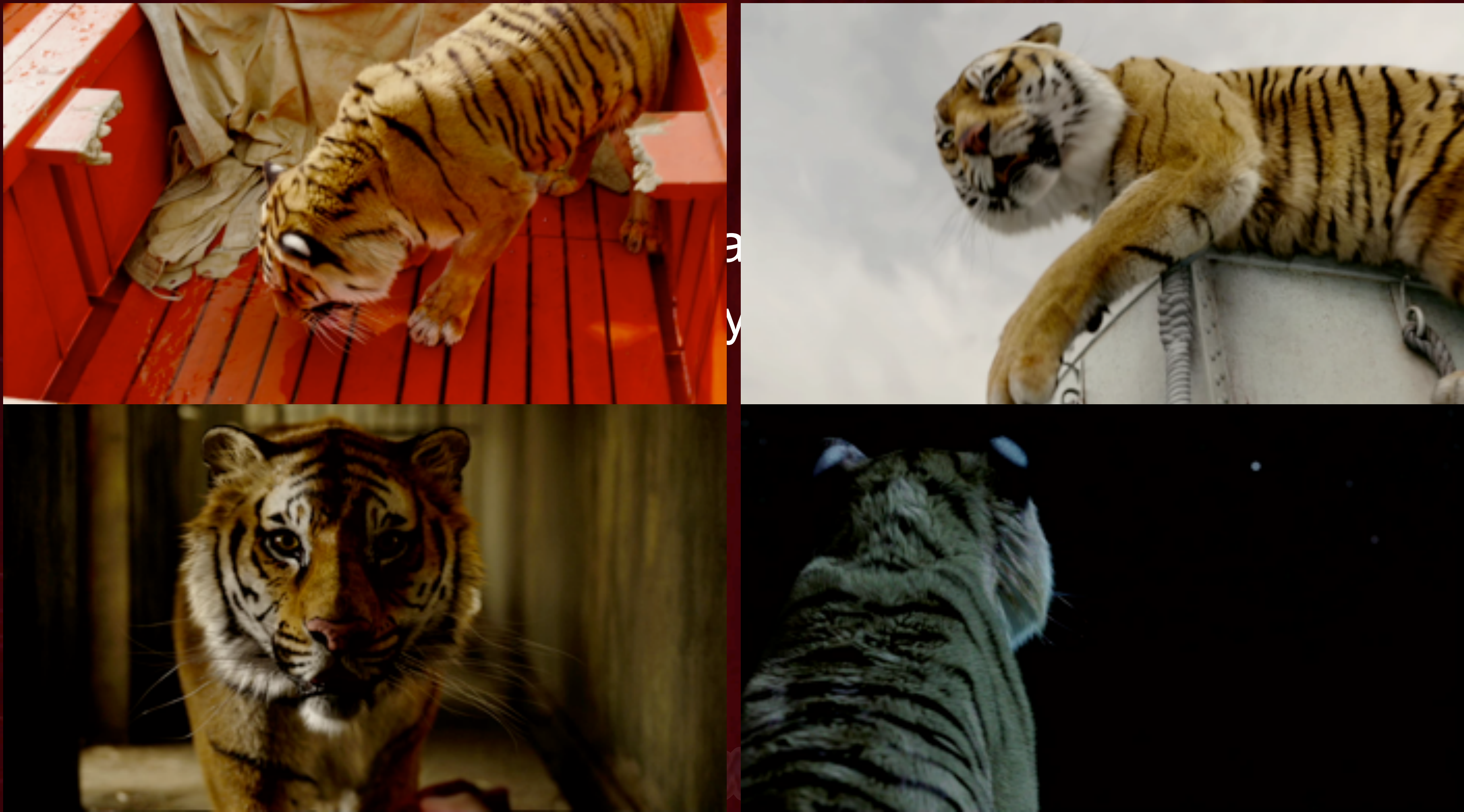




# 1) Hair Shading: Area Lights

# 1) Hair Shading:
# Area Lights

- First show to use area lights almost exclusively



# 1) Hair Shading:
# Area Lights

# 1) Hair Shading: Area Lights

- First show to use area lights almost exclusively
  - Blends realistically with live-action footage

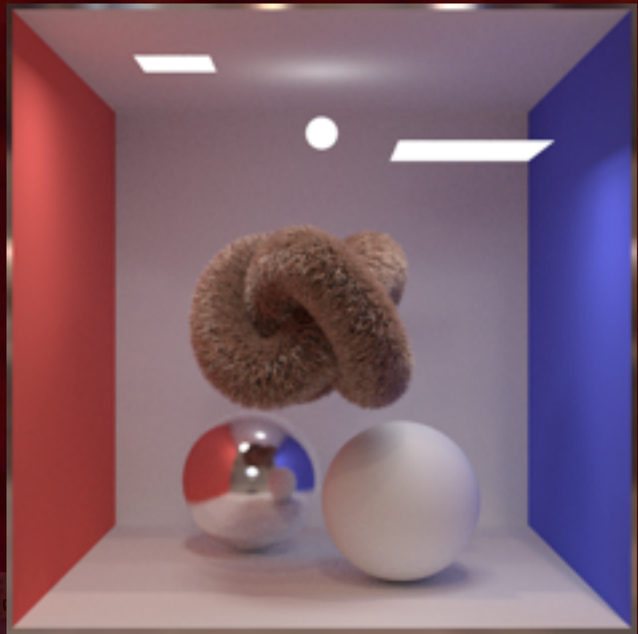
# 1) Hair Shading: Area Lights

- First show to use area lights almost exclusively
  - Blends realistically with live-action footage

# 1) Hair Shading: Area Lights

# 1) Hair Shading:
# Area Lights

- How to deal with them efficiently
  - Good Importance Sampling:

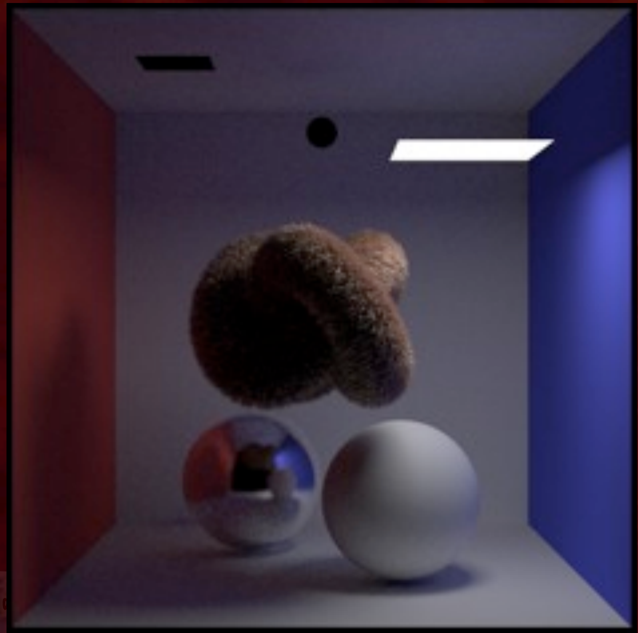


# 1) Hair Shading: Area Lights

- How to deal with them efficiently
  - Good Importance Sampling:
    - Rectangles

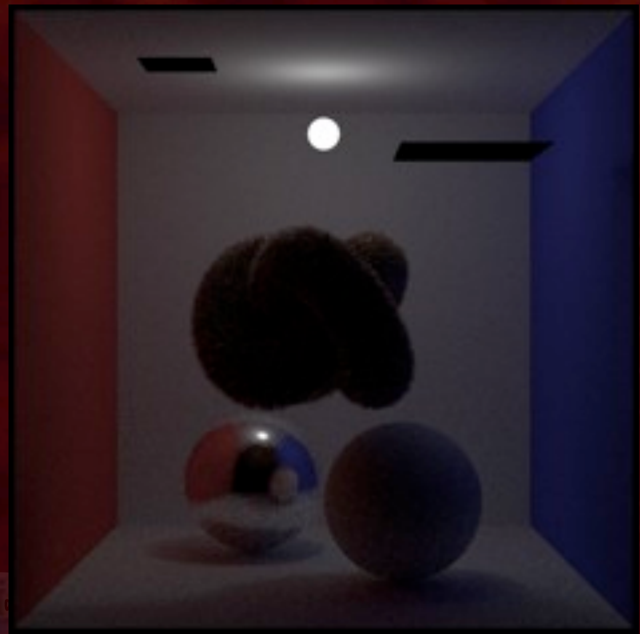
# 1) Hair Shading:
# Area Lights

- How to deal with them efficiently
  - Good Importance Sampling:
    - Rectangles



# 1) Hair Shading:
# Area Lights

- How to deal with them efficiently
  - Good Importance Sampling:
    - Rectangles
    - Spheres

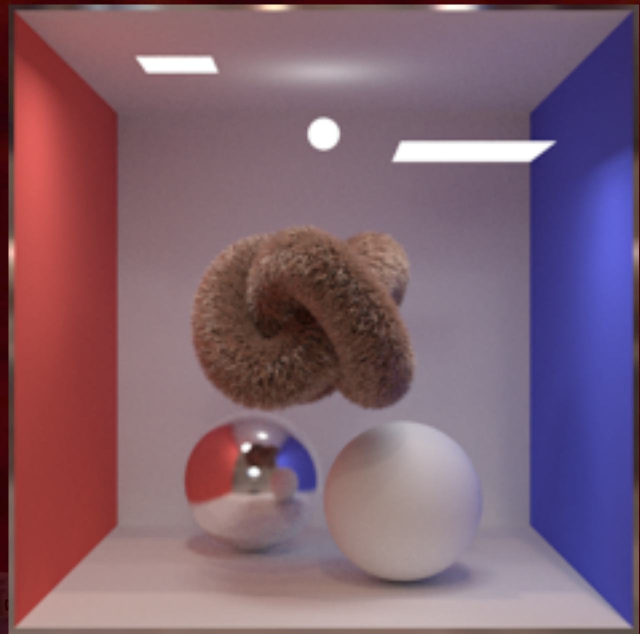


# 1) Hair Shading:
## Area Lights

- How to deal with them efficiently
  - Good Importance Sampling:
    - Rectangles
    - Spheres
    - Environment lights

# 1) Hair Shading: Area Lights

- How to deal with them efficiently
  - Good Importance Sampling:
    - Rectangles
    - Spheres
    - Environment lights
    - Ray Magnets
      - shapes that attract light rays to geometry



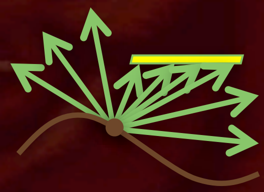
# 1) Hair Shading:
# Area Lights

# 1) Hair Shading:
# Area Lights

- Multiple Importance Sampling (MIS) [Veach97]:

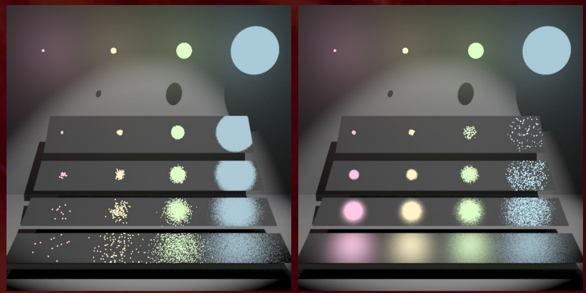
# 1) Hair Shading: Area Lights

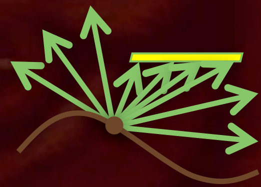
- Multiple Importance Sampling (MIS) [Veach97]:
  - BSDF vs Light Importance



# 1) Hair Shading: Area Lights

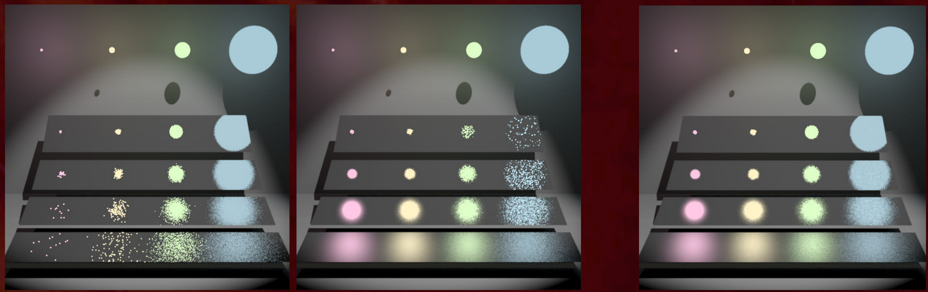
- Multiple Importance Sampling (MIS) [Veach97]:
  - BSDF vs Light Importance

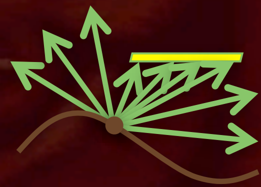




# 1) Hair Shading: Area Lights

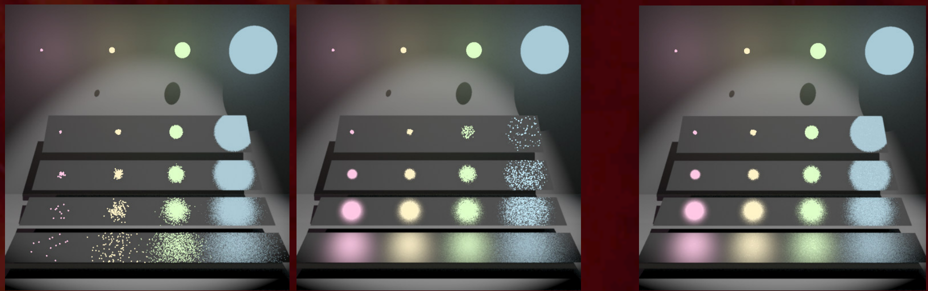
- Multiple Importance Sampling (MIS) [Veach97]:
  - BSDF vs Light Importance





# 1) Hair Shading:
# Area Lights

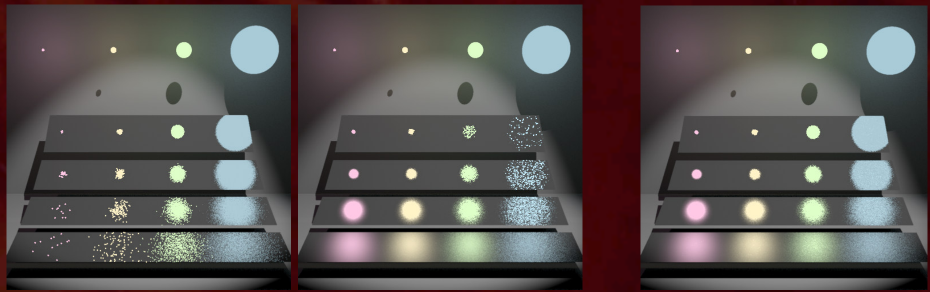
- Multiple Importance Sampling (MIS) [Veach97]:
  - BSDF vs Light Importance



- Stochastic light selection

# 1) Hair Shading: Area Lights

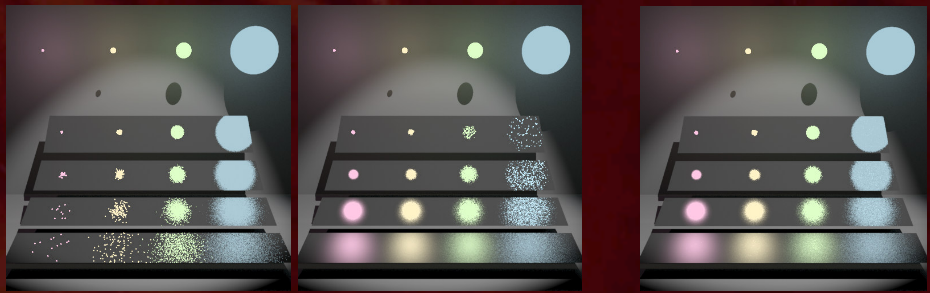
- Multiple Importance Sampling (MIS) [Veach97]:
  - BSDF vs Light Importance



- Stochastic light selection
  - based on solid angle, average radiance

# 1) Hair Shading: Area Lights

- Multiple Importance Sampling (MIS) [Veach97]:
  - BSDF vs Light Importance

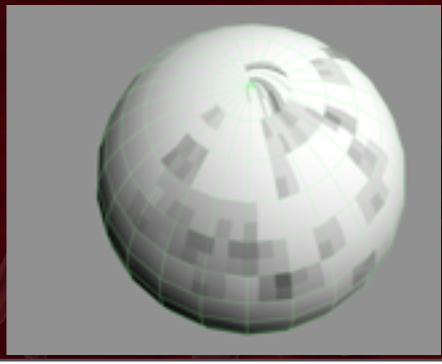


- Stochastic light selection
  - based on solid angle, average radiance
  - also uses MIS

# 1) Hair Shading:
# Area Lights

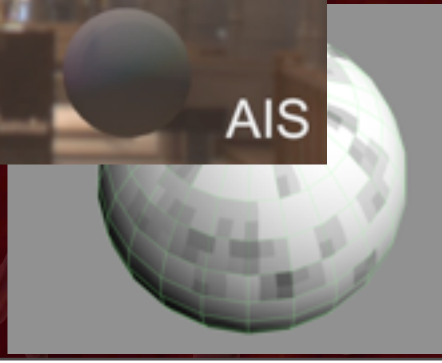
# 1) Hair Shading:
# Area Lights

- Adaptive Importance Sampling [Neulander11]
  - Sampled ray directions are rated for contribution
  - Poorly rated directions are rejected in the future



# 1) Hair Shading: Area Lights

- Ada… Neulander11]
  - S… for contribution
  - P… ed in the future
  - R…

# 1) Hair Shading:
# Area Lights

# 1) Hair Shading:
# Area Lights

- Adaptive Importance Sampling [Neulander11]
- Well suited to fur

# 1) Hair Shading: Area Lights

- Adaptive Importance Sampling [Neulander11]
- Well suited to fur
  - lots of occlusion

# 1) Hair Shading: Area Lights

- Adaptive Importance Sampling [Neulander11]
- Well suited to fur
  - lots of occlusion

# 1) Hair Shading: Area Lights

- Adaptive Importance Sampling [Neulander11]
- Well suited to fur
  - lots of occlusion

# 1) Hair Shading: Area Lights

- Adaptive Importance Sampling [Neulander11]
- Well suited to fur
  - lots of occlusion

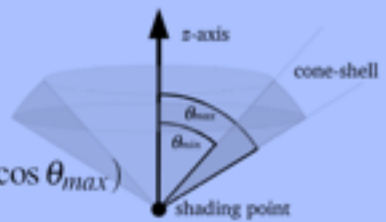
# 1) Hair Shading:

## BSDF

# 1) Hair Shading:
# BSDF

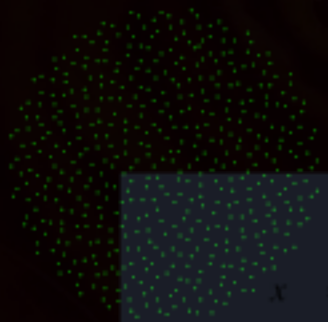
- Cone-Shell BSDF [Neulander10]

$$
\begin{aligned}
x &= \sqrt{1-z^2}\cos(2\pi\xi_2) \\
y &= \sqrt{1-z^2}\sin(2\pi\xi_2) \\
z &= \cos\theta_{max} + \xi_1\left(\cos\theta_{min} - \cos\theta_{max}\right) \\
weight &= z\cos\theta_{mid} + \sqrt{1-z^2}\sin\theta_{mid}
\end{aligned}
$$

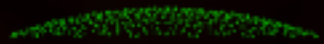


# 1) Hair Shading: BSDF

- Cone-Shell BSDF [Neulander10]



$$
\begin{aligned}
x &= \sqrt{1-z^2}\cos(2\pi\xi_2) \\
y &= \sqrt{1-z^2}\sin(2\pi\xi_2) \\
z &= \cos\theta_{max} + \xi_1\left(\cos\theta_{min} - \cos\theta_{max}\right) \\
weight &= z\cos\theta_{mid} + \sqrt{1-z^2}\sin\theta_{mid}
\end{aligned}
$$

uniform sampling ; overhead view

uniform sampling ; side view

# 1) Hair Shading:

## BSDF

# 1) Hair Shading:
## BSDF

- Cone-Shell BSDF [Neulander10]

# 1) Hair Shading:
# BSDF

- Cone-Shell BSDF [Neulander10]
  - Dual highlights (inspired by Marschner)
    - shift parameter *t* when computing spline tangents
    - randomize *t* to break up highlight

# 1) Hair Shading: BSDF

- Cone-Shell BSDF [Neulander10]
  - Dual highlights (inspired by Marschner)
    - shift parameter *t* when computing spline tangents
    - randomize *t* to break up highlight

# 1) Hair Shading: BSDF

- Cone-Shell BSDF [Neulander10]
  - Dual highlights (inspired by Marschner)
    - shift parameter *t* when computing spline tangents
    - randomize *t* to break up highlight

# 1) Hair Shading: BSDF

- Cone-Shell BSDF [Neulander10]
  - Dual highlights (inspired by Marschner)
    - shift parameter *t* when computing spline tangents
    - randomize *t* to break up highlight

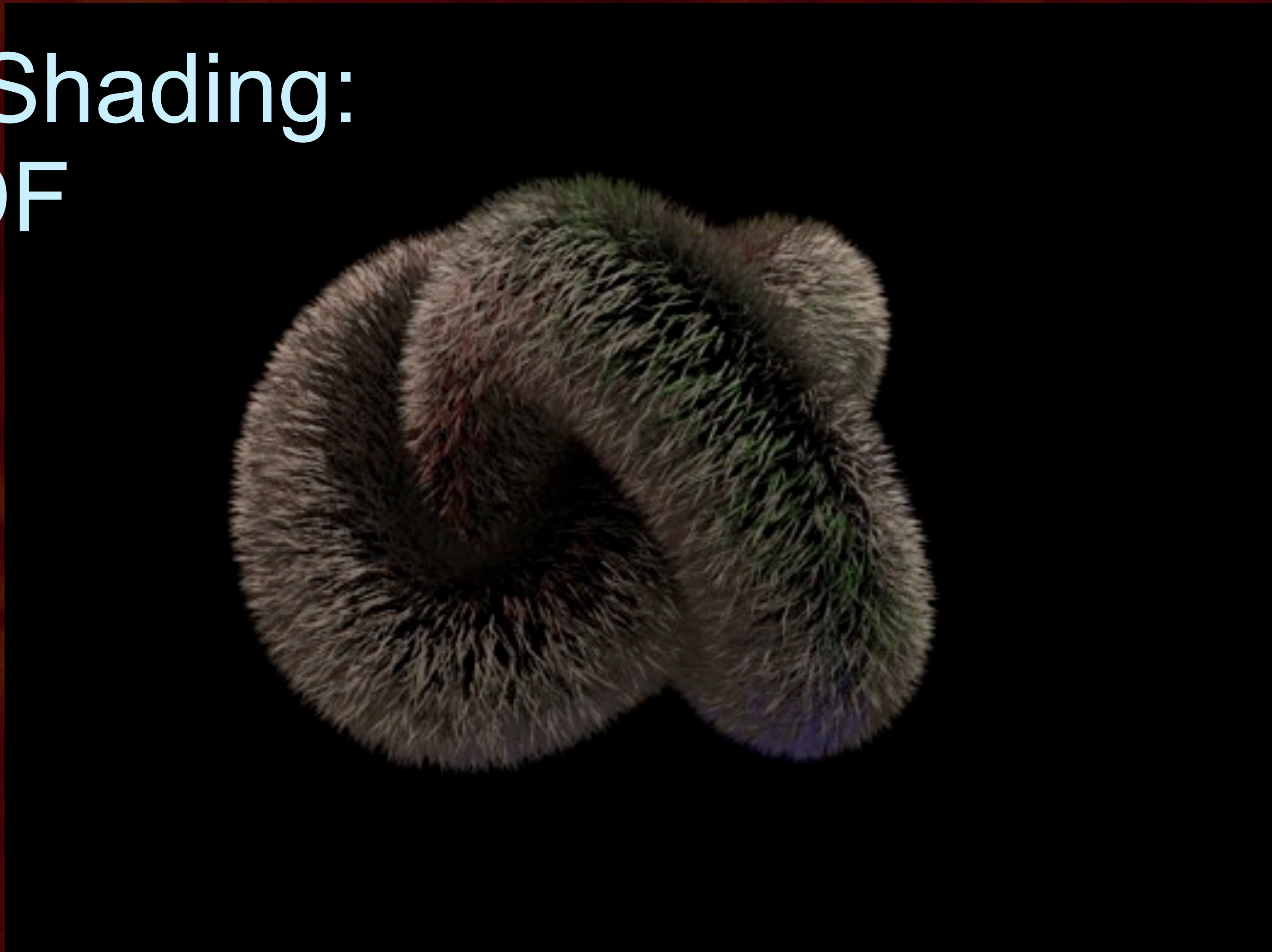


# 1) Hair Shading:
# BSDF

# 1) Hair Shading: BSDF

# 1) Hair Shading: BSDF

# 1) Hair Shading: BSDF

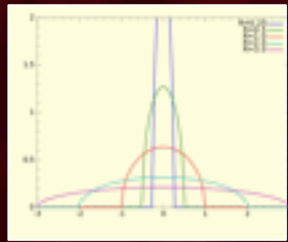


# 1) Hair Shading: BSDF

# 1) Hair Shading: BSDF

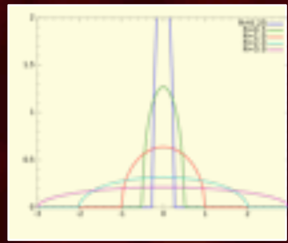
# 1) Hair Shading: BSDF

- Wigner Semicircle Importance Sampler



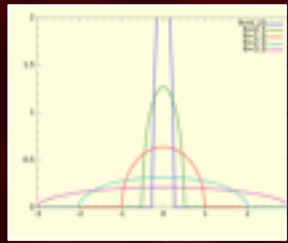
# 1) Hair Shading:
# BSDF

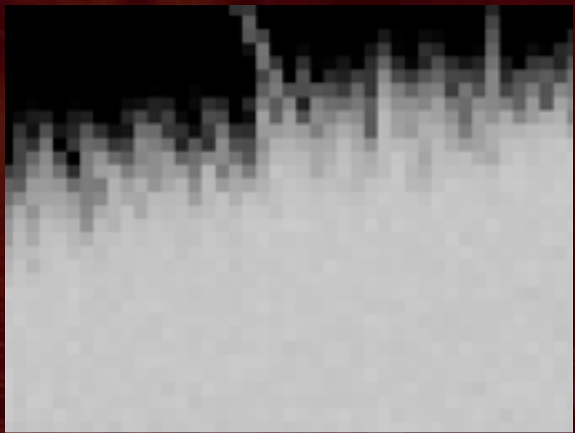
- Wigner Semicircle Importance Sampler
  - Closer to optimal than previous model



# 1) Hair Shading: BSDF

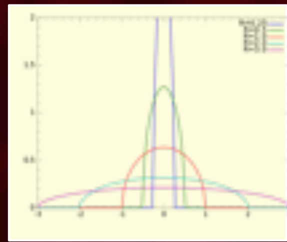
- Wigner Semicircle Importance Sampler
  - Closer to optimal than previous model

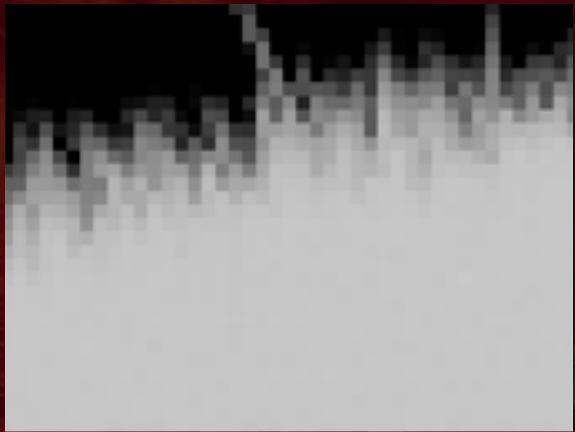




# 1) Hair Shading: BSDF

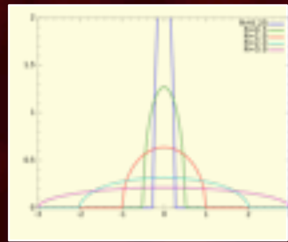
- Wigner Semicircle Importance Sampler
  - Closer to optimal than previous model

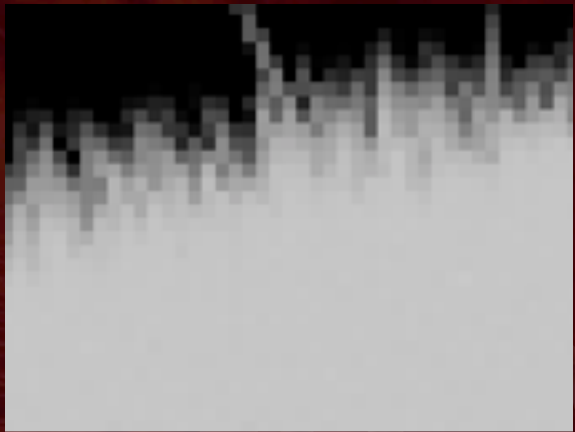


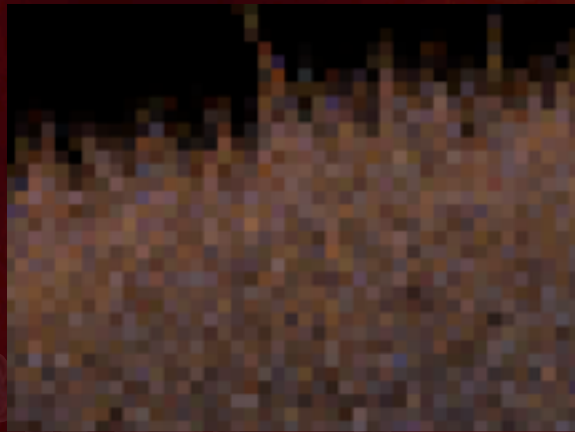


# 1) Hair Shading:
# BSDF

- Wigner Semicircle Importance Sampler
  - Closer to optimal than previous model

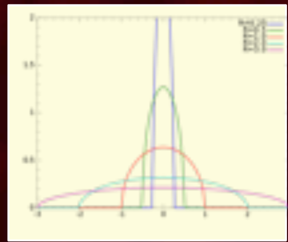


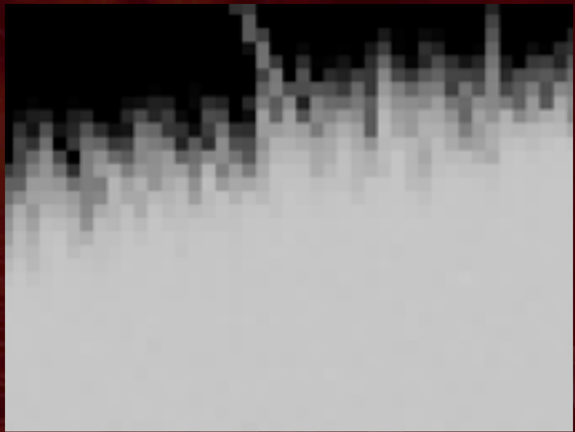


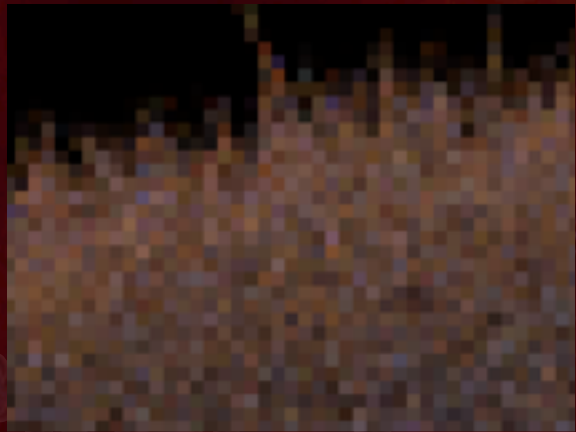


# 1) Hair Shading: BSDF

- Wigner Semicircle Importance Sampler
  - Closer to optimal than previous model

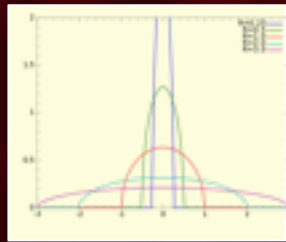






# 1) Hair Shading:
# BSDF

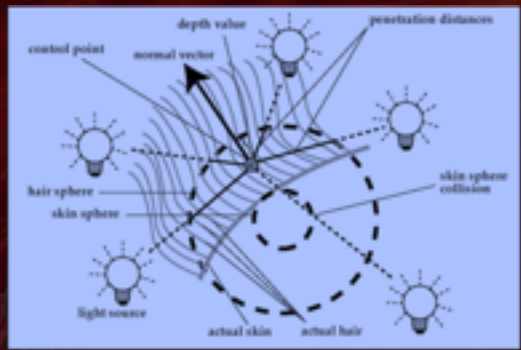
- Wigner Semicircle Importance Sampler
  - Closer to optimal than previous model
  - Implementation:
    - inverse CDF table, interpolate between entries



# 2) Renderer Optimizations: Skin Occlusion

# 2) Renderer Optimizations: Skin Occlusion

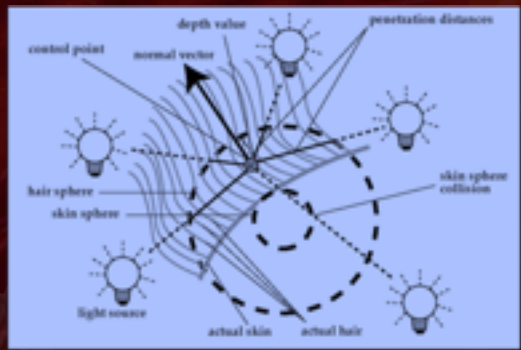
- Based on volumetric occlusion model




Rhythm&Hues

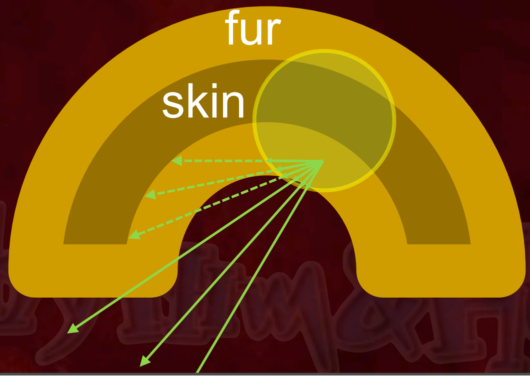
# 2) Renderer Optimizations: Skin Occlusion

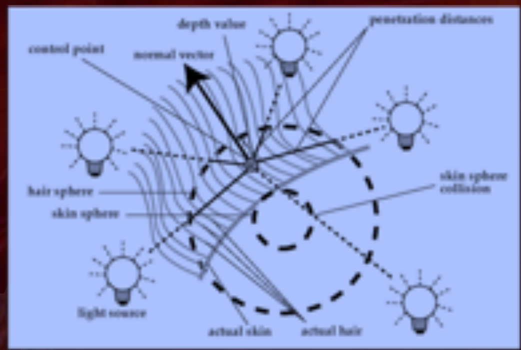
- Based on volumetric occlusion model
  - First introduced in [Neulander04]



# 2) Renderer Optimizations: Skin Occlusion

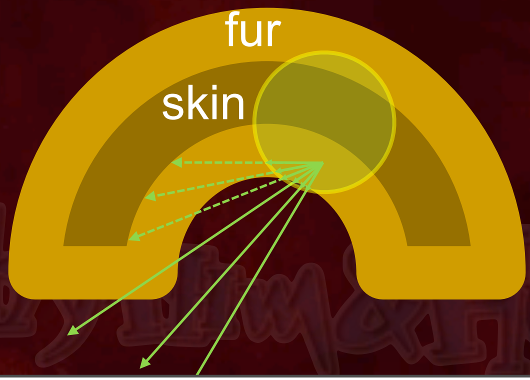
- Based on volumetric occlusion model
  - First introduced in [Neulander04]
    - approximates fractional ray occlusion by fur & skin

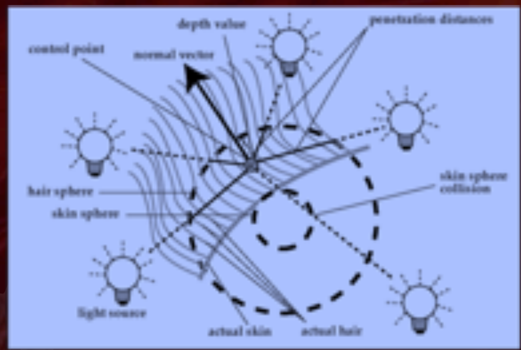


# 2) Renderer Optimizations: Skin Occlusion

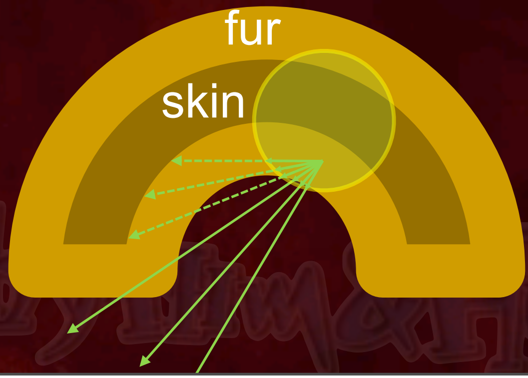
- Based on volumetric occlusion model
  - First introduced in [Neulander04]
    - approximates fractional ray occlusion by fur & skin

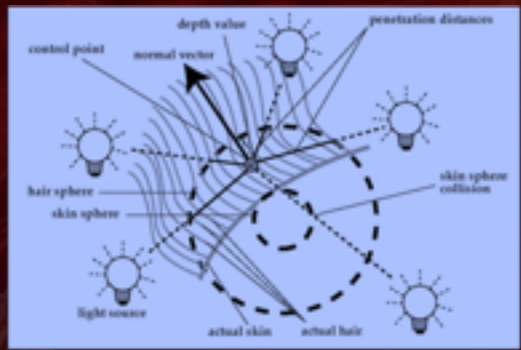


# 2) Renderer Optimizations: Skin Occlusion

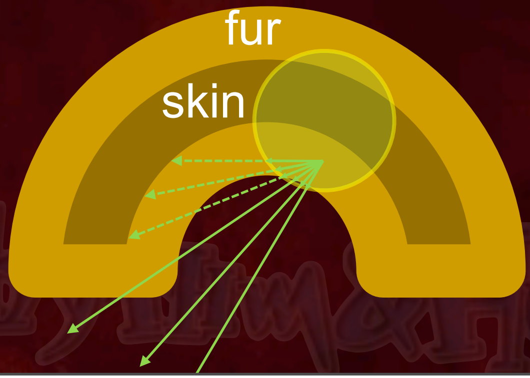
- Based on volumetric occlusion model
  - First introduced in [Neulander04]
    - approximates fractional ray occlusion by fur & skin

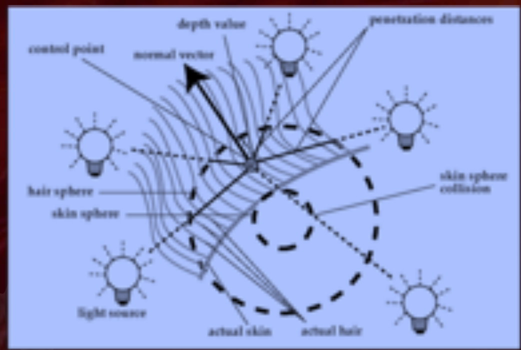


# 2) Renderer Optimizations: Skin Occlusion

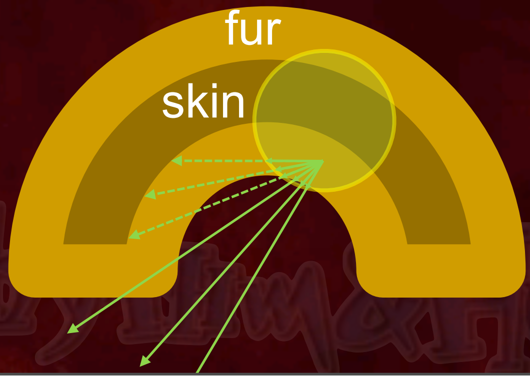
- Based on volumetric occlusion model
  - First introduced in [Neulander04]
    - approximates fractional ray occlusion by fur & skin
  - We use only skin sphere for full/no occlusion

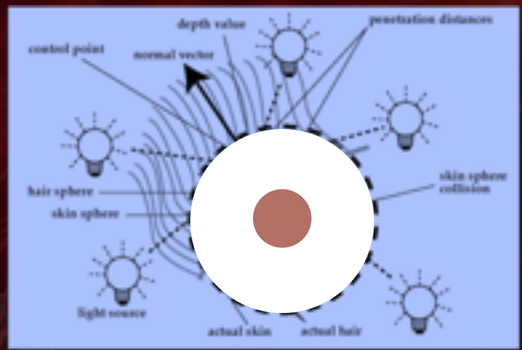


# 2) Renderer Optimizations: Skin Occlusion

- Based on volumetric occlusion model
  - First introduced in [Neulander04]
    - approximates fractional ray occlusion by fur & skin
  - We use only skin sphere for full/no occlusion



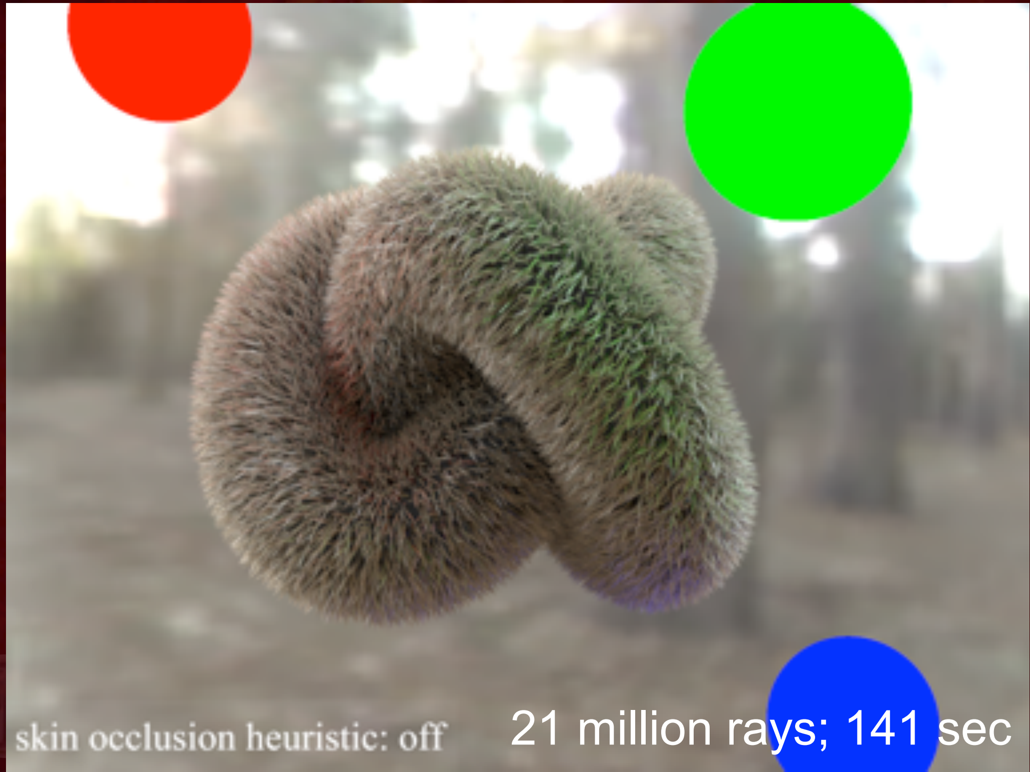



# 2) Renderer Optimizations: Skin Occlusion

- Significant speedup (~50%)
- Minimal image difference
- Controllable speed/quality

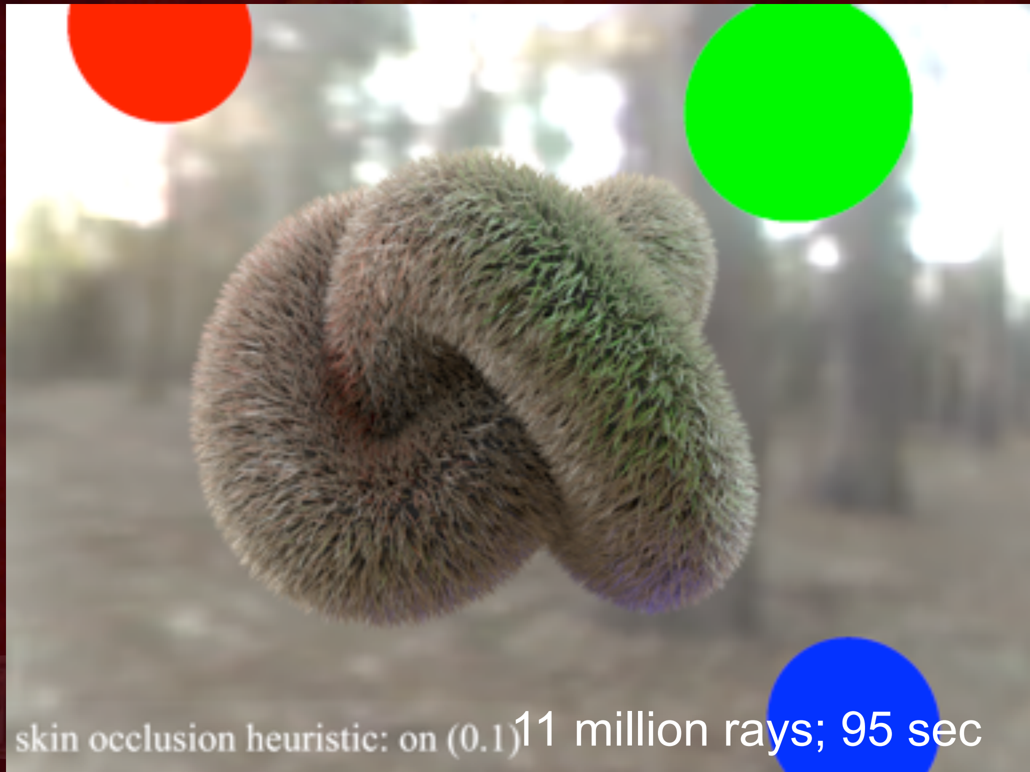
# 2) Renderer Optimizations: Skin Occlusion

- Significant speedup (~50%)
- Minimal image difference
- Controllable speed/quality



# 2) Renderer Optimizations: Skin Occlusion

- Significant speedup (~50%)
- Minimal image difference
- Controllable speed/quality



# 2) Renderer Optimizations:

## Screen Door Transparency

# 2) Renderer Optimizations:
## Screen Door Transparency

- Hybrid renderer:

# 2) Renderer Optimizations:
## Screen Door Transparency

- Hybrid renderer:
  - Scanline mode:
    - thick, semitransparent strands

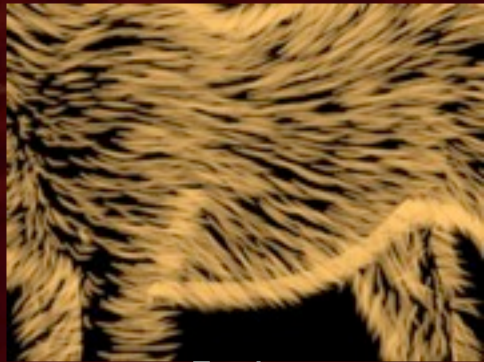
Rhythm&Hues

# 2) Renderer Optimizations:
## Screen Door Transparency

- Hybrid renderer:
  - Scanline mode:
    - thick, semitransparent strands
  - Raytraced occlusion:
    - thinned, opaque strands (of equal coverage)
    - thickness, opacity can vary along strand

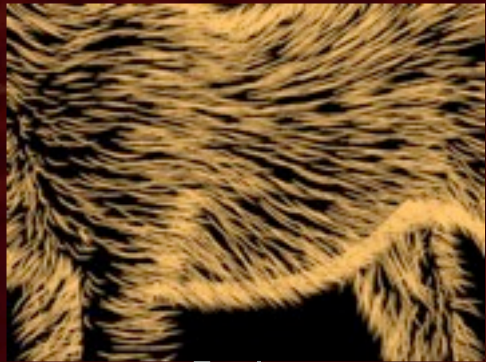
# 2) Renderer Optimizations: Screen Door Transparency

- Hybrid renderer:
  - Scanline mode:
    - thick, semitransparent strands
  - Raytraced occlusion:
    - thinned, opaque strands (of equal
    - thickness, opacity can vary along strand



# 2) Renderer Optimizations:
# Screen Door Transparency

- Hybrid renderer:
  - Scanline mode:
    - thick, semitransparent strands
  - Raytraced occlusion:
    - thinned, opaque strands (of equal
    - thickness, opacity can vary along strand



# 2) Renderer Optimizations: Screen Door Transparency

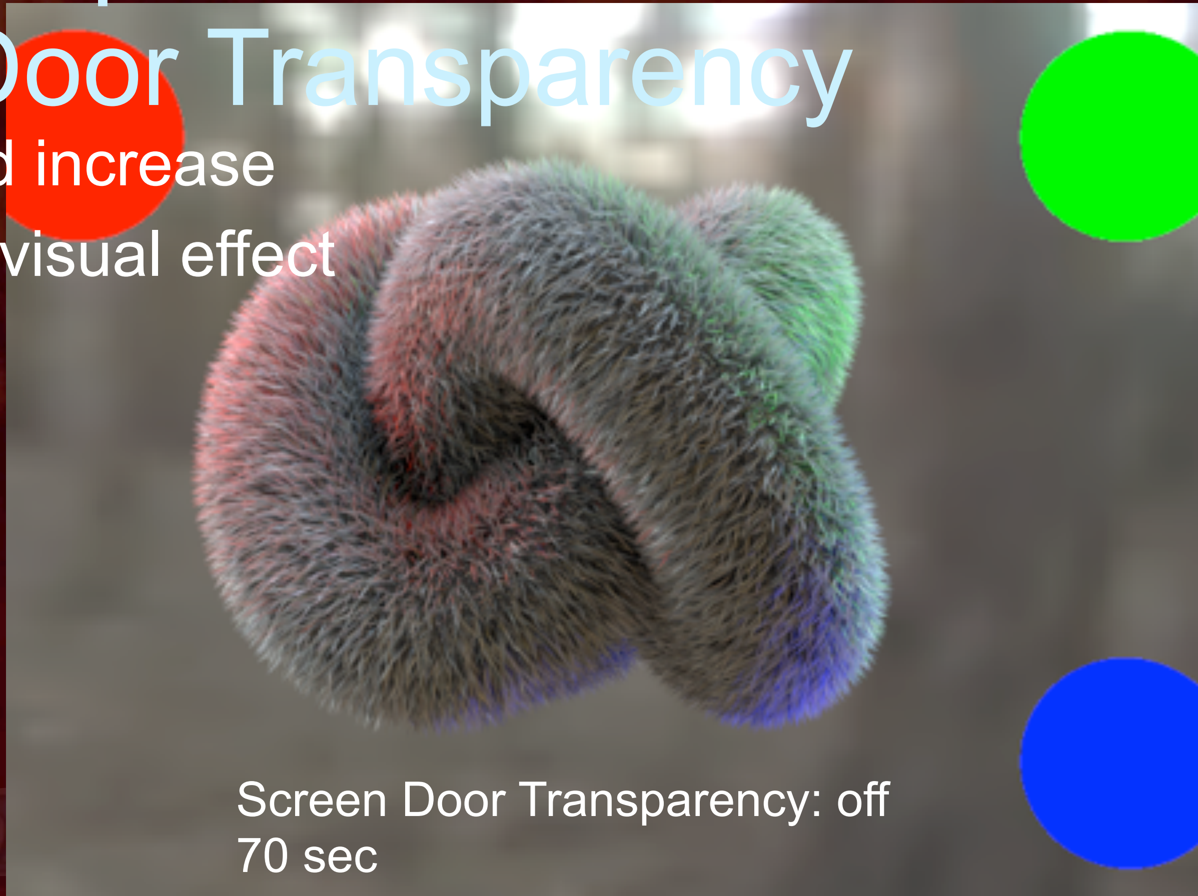
- Hybrid renderer:
  - Scanline mode:
    - thick, semitransparent strands
  - Raytraced occlusion:
    - thinned, opaque strands (of equal coverage)
    - thickness, opacity can vary along strand
  - Fewer ray hits, no further transparency rays

# 2) Renderer Optimizations: Screen Door Transparency

- Large speed increase
- Only subtle visual effect

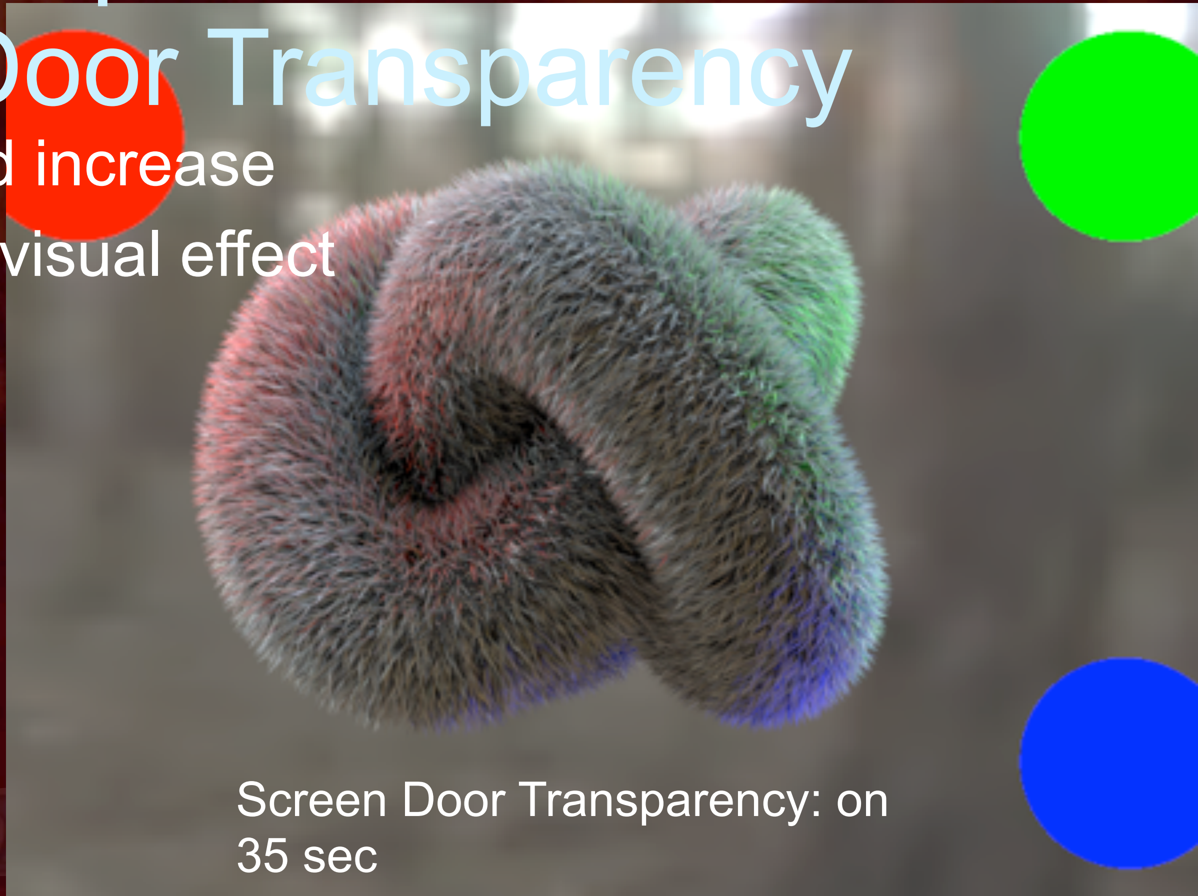
# 2) Renderer Optimizations: Screen Door Transparency

- Large speed increase
- Only subtle visual effect



# 2) Renderer Optimizations: Screen Door Transparency

- Large speed increase
- Only subtle visual effect



# 2) Renderer Optimizations:
## BVH Ray Tracer

# 2) Renderer Optimizations: BVH Ray Tracer

- Quad BVH architecture
  - tries to process up to 4 hair segments at once
  - SSE optimizations
  - memory arena via anonymous mmap

# 2) Renderer Optimizations:
# BVH Ray Tracer

- Quad BVH architecture
  - tries to process up to 4 hair segments at once
  - SSE optimizations
  - memory arena via anonymous mmap
- Ray-hair intersection based on *Ray Tracing for Curves Primites* [Nakamaru, Ohno WSCG 2002]
  - hair CP-segment-based bbox construction
  - Surface Area Heuristic evaluation

# 2) Renderer Optimizations:
# BVH Ray Tracer

# 2) Renderer Optimizations: BVH Ray Tracer

- Recent development
  - Disk-Based storage of complete BVH
    - user-defined RAM footprint
    - computed once and stored on disk

# 2) Renderer Optimizations:
## Reflection Cache

# 2) Renderer Optimizations:

## Reflection Cache

- Introduced in [Neulander10]

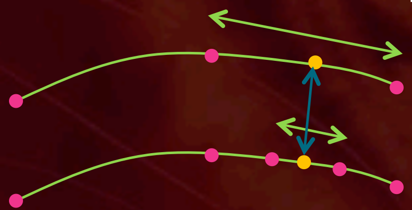
# 2) Renderer Optimizations:

## Reflection Cache

- Introduced in [Neulander10]

# 2) Renderer Optimizations: Reflection Cache

- Introduced in [Neulander10]
  - caches reflected radiance at primary rays along strand



# 2) Renderer Optimizations: Reflection Cache

# 2) Renderer Optimizations: Reflection Cache

- Enhancements

# 2) Renderer Optimizations: Reflection Cache

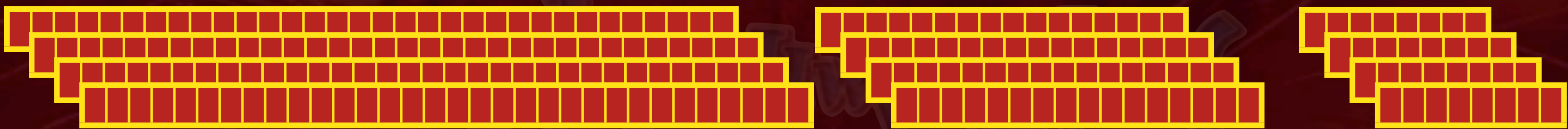
- Enhancements
  - Cache can now store
    - diffuse reflection
    - primary specular reflection
    - secondary specular reflectio

# 2) Renderer Optimizations: Reflection Cache

- Enhancements
  - Cache can now store
    - diffuse reflection
    - primary specular reflection
    - secondary specular reflectio
    - various light paths for above

# 2) Renderer Optimizations: Reflection Cache

- Enhancements
  - Cache can now store
    - diffuse reflection
    - primary specular reflection
    - secondary specular reflectio
    - various light paths for above
  - Clustered allocation improves memory access

# 2) Renderer Optimizations:
## Multithreading

# 2) Renderer Optimizations: Multithreading

- Improved performance of hair reflection cache

# 2) Renderer Optimizations: Multithreading

- Improved performance of hair reflection cache
  - Reads are not blocked by cache updates

# 2) Renderer Optimizations: Multithreading

- Improved performance of hair reflection cache
  - Reads are not blocked by cache updates
  - Writes use Read-Copy-Update (RCU) for synchronization

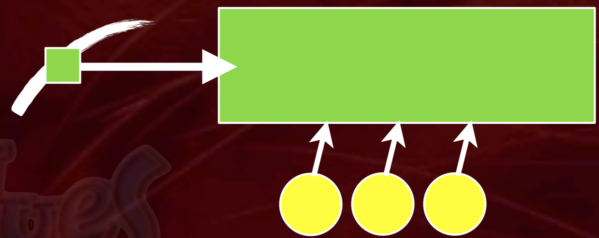
# 2) Renderer Optimizations: Multithreading

- Improved performance of hair reflection cache
  - Reads are not blocked by cache updates
  - Writes use Read-Copy-Update (RCU) for synchronization
    - RCU is used extensively in the Linux kernel
    - Allows lock-free cache reads

# 2) Renderer Optimizations: Multithreading

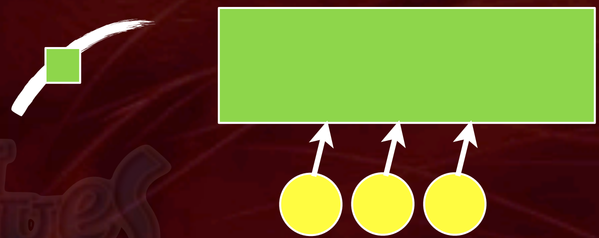
# 2) Renderer Optimizations: Multithreading

- Cache replacement policy with RCU:



# 2) Renderer Optimizations: Multithreading

- Cache replacement policy with RCU:
  - Remove index but keep data while readers exist



# 2) Renderer Optimizations: Multithreading

- Cache replacement policy with RCU:
  - Remove index but keep data while readers exist
  - After some period, readers must finish

# 2) Renderer Optimizations: Multithreading

- Cache replacement policy with RCU:
  - Remove index but keep data while readers exist
  - After some period, readers must finish
  - At that point, remove data from cache

# 2) Renderer Optimizations: Multithreading

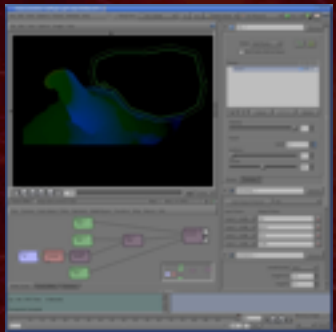
- Cache replacement policy with RCU:
  - Remove index but keep data while readers exist
  - After some period, readers must finish
  - At that point, remove data from cache
- Improved concurrency:
  - near-linear speed (8 threads)
  - slight memory increase

# 3) Postprocessing: Motion Blur

- ***pixmotor***: pixel motion integrator [Neulander07]
    - Screen-space motion vectors, depth values output by renderer
    - Integrated as a plugin into compositing software

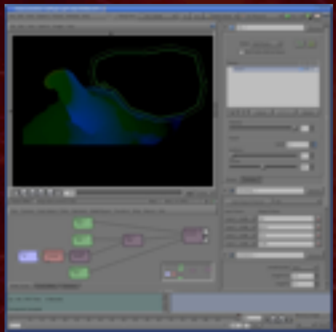
# 3) Postprocessing: Motion Blur

- ***pixmotor***: pixel motion integrator [Neulander07]
  - Screen-space motion vectors, depth values output by renderer
  - Integrated as a plugin into compositing software



# 3) Postprocessing: Motion Blur

- ***pixmotor***: pixel motion integrator [Neulander07]
  - Screen-space motion vectors, depth values output by renderer
  - Integrated as a plugin into compositing software



# 3) Postprocessing:
# Motion Blur

# 3) Postprocessing: Motion Blur

# 3) Postprocessing:
## Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

- Synthesize right-eye image from left-eye image

# 3) Postprocessing:

## Stereo Synthesis

- Synthesize right-eye image from left-eye image
- ***pixstereo***: modified form of pixmotor

# 3) Postprocessing: Stereo Synthesis

- Synthesize right-eye image from left-eye image
- ***pixstereo***: modified form of pixmotor
  - We have:
    - camera-projected image

# 3) Postprocessing:
# Stereo Synthesis

- Synthesize right-eye image from left-eye image
- ***pixstereo***: modified form of pixmotor
  - We have:
    - camera-projected image
    - depth values

# 3) Postprocessing: Stereo Synthesis

- Synthesize right-eye image from left-eye image
- ***pixstereo***: modified form of pixmotor
  - We have:
    - camera-projected image
    - depth values
    - camera parameters

# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

- We can construct 3D "surface" of each pixel and reproject to other camera

# 3) Postprocessing: Stereo Synthesis

- We can construct 3D “surface” of each pixel and reproject to other camera
- Use this to compute screen-space motion vectors



# 3) Postprocessing:
# Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

- Recipe:

# 3) Postprocessing: Stereo Synthesis

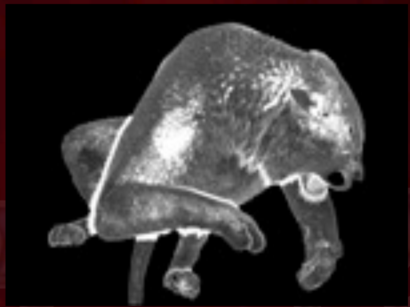
- Recipe:
  - Compute parallax-based motion vectors

# 3) Postprocessing: Stereo Synthesis

- Recipe:
  - Compute parallax-based motion vectors
  - Compute motion gradient image

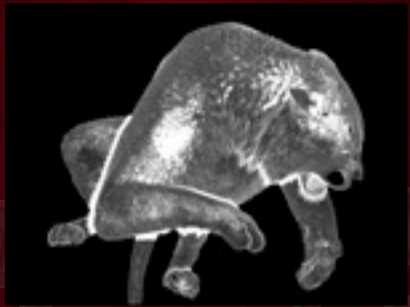
# 3) Postprocessing: Stereo Synthesis

- Recipe:
  - Compute parallax-based motion vectors
  - Compute motion gradient image
  - Fill holes using heuristics

# 3) Postprocessing: Stereo Synthesis

- Recipe:
  - Compute parallax-based motion vectors
  - Compute motion gradient image
  - Fill holes using heuristics
  - Build result at 4x+ resolution, then downsample


# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

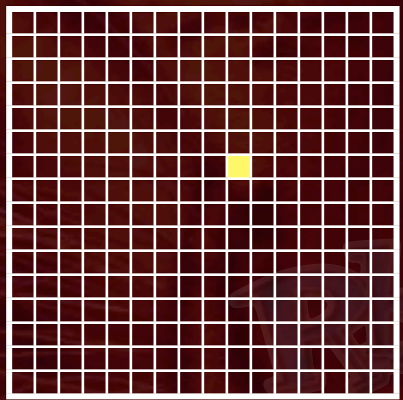
# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing: Stereo Synthesis

# 3) Postprocessing:
# Pixmotor/Pixstereo Optimization

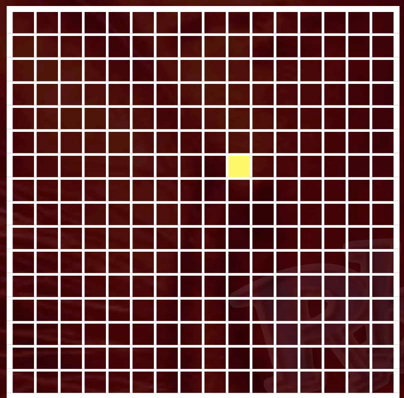
# 3) Postprocessing: Pixmotor/Pixstereo Optimization

- High-res work buffer stores only pixel coords



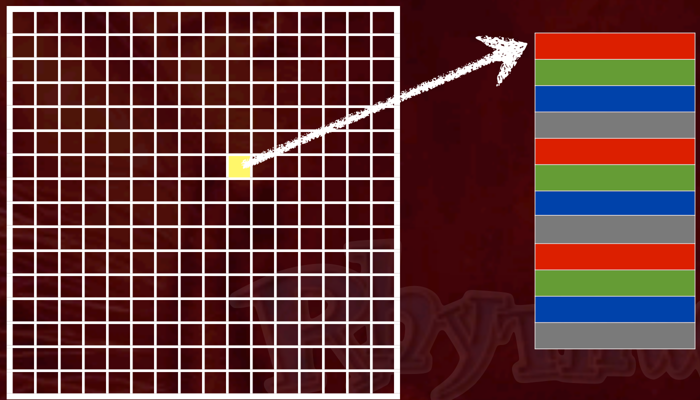
# 3) Postprocessing: Pixmotor/Pixstereo Optimization

- High-res work buffer stores only pixel coords
  - pair of 16-bit coords instead of many floats (plus one float for depth)



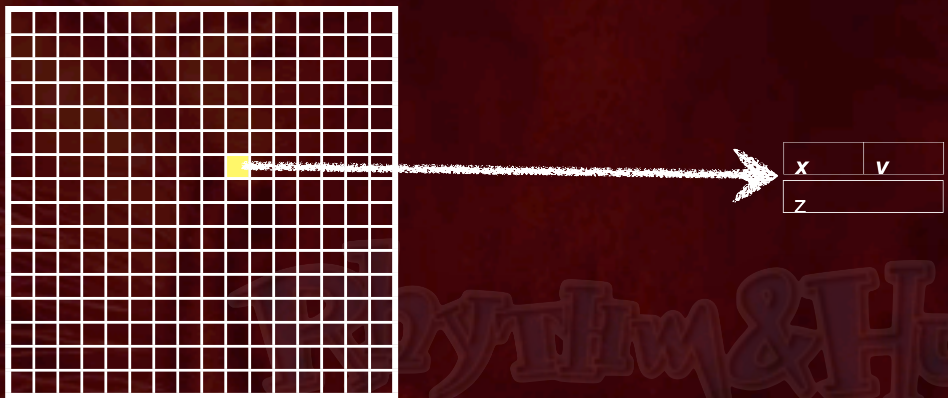
# 3) Postprocessing: Pixmotor/Pixstereo Optimization

- High-res work buffer stores only pixel coords
  - pair of 16-bit coords instead of many floats (plus one float for depth)



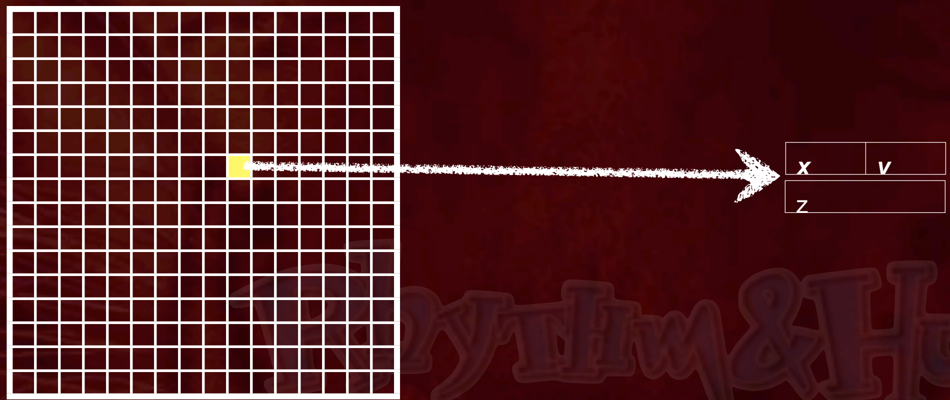
# 3) Postprocessing: Pixmotor/Pixstereo Optimization

- High-res work buffer stores only pixel coords
  - pair of 16-bit coords instead of many floats (plus one float for depth)



# 3) Postprocessing: Pixmotor/Pixstereo Optimization

- High-res work buffer stores only pixel coords
  - pair of 16-bit coords instead of many floats (plus one float for depth)
  - faster due to lower memory bandwidth



# 3) Postprocessing:
## Pixstereo Quality

# 3) Postprocessing:
## Pixstereo Quality

- Improved output filtering for pixstereo

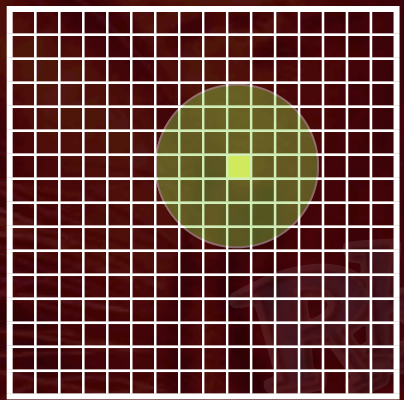
# 3) Postprocessing:

# Pixstereo Quality

- Improved output filtering for pixstereo
  - need to preserve sharpness of input image

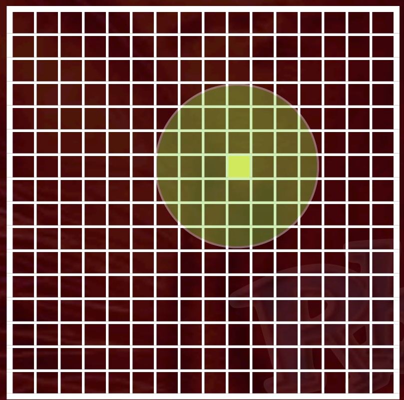
# 3) Postprocessing:
# Pixstereo Quality

- Improved output filtering for pixstereo
  - need to preserve sharpness of input image
    - negative lobed filter (Lanczos-windowed sinc)



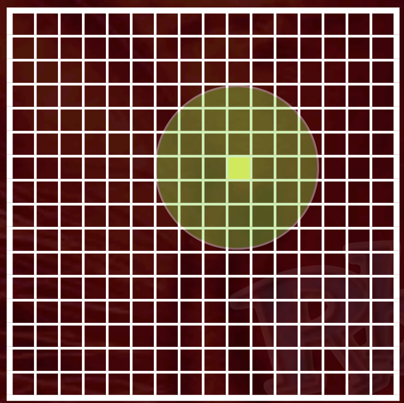
# 3) Postprocessing: Pixstereo Quality

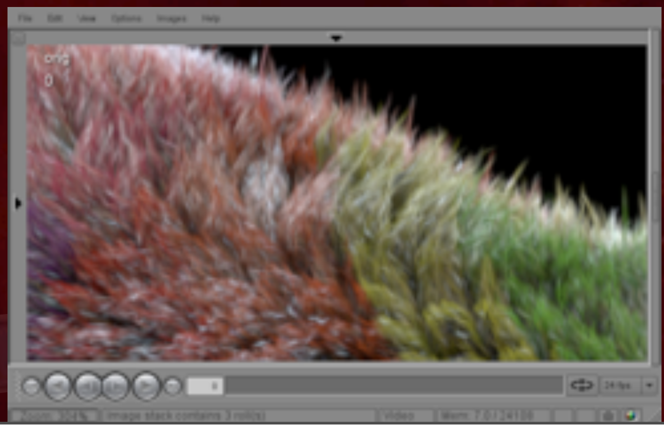
- Improved output filtering for pixstereo
  - need to preserve sharpness of input image
    - negative lobed filter (Lanczos-windowed sinc)
  - not useful for MB but helps for stereo



# 3) Postprocessing:
## Pixstereo Quality

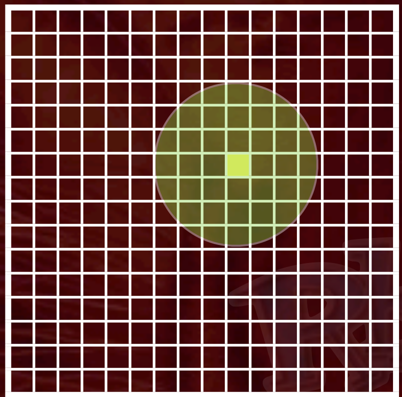
- Improved output filtering for pixstereo
  - need to preserve sharpness of input image
    - negative lobed filter (Lanczos-windowed sinc)
  - not useful for MB but helps for stereo

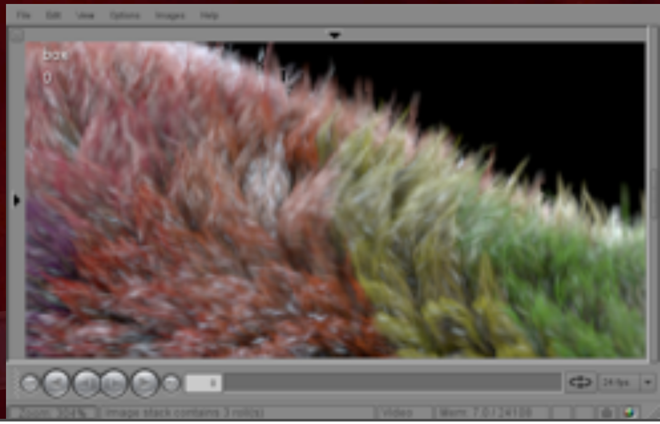




# 3) Postprocessing: Pixstereo Quality

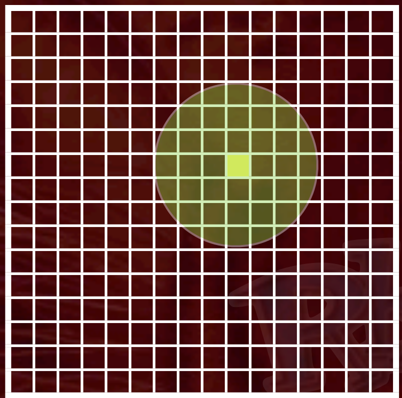
- Improved output filtering for pixstereo
  - need to preserve sharpness of input image
    - negative lobed filter (Lanczos-windowed sinc)
  - not useful for MB but helps for stereo

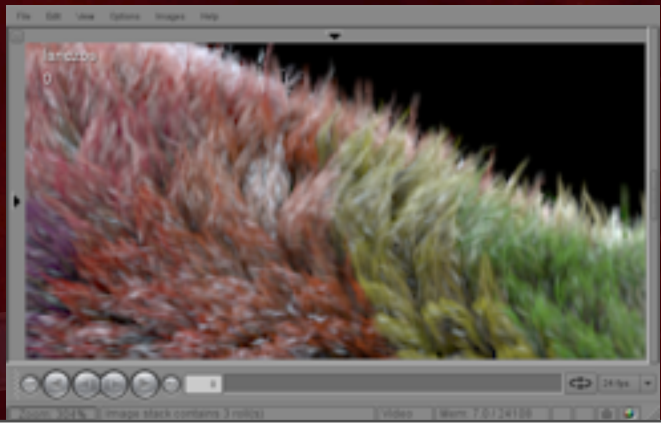




# 3) Postprocessing:
# Pixstereo Quality

- Improved output filtering for pixstereo
  - need to preserve sharpness of input image
    - negative lobed filter (Lanczos-windowed sinc)
  - not useful for MB but helps for stereo





# Conclusions & Future Work

# Conclusions & Future Work

- Most of credit for the look of the fur in *Life of Pi* goes to the digital artists

# Conclusions & Future Work

- Most of credit for the look of the fur in *Life of Pi* goes to the digital artists
- Main contributions of our rendering software:
  - A good level of realism is achievable

# Conclusions & Future Work

- Most of credit for the look of the fur in *Life of Pi* goes to the digital artists
- Main contributions of our rendering software:
  - A good level of realism is achievable
  - Results are highly art-directable

# Conclusions & Future Work

- Most of credit for the look of the fur in *Life of Pi* goes to the digital artists
- Main contributions of our rendering software:
  - A good level of realism is achievable
  - Results are highly art-directable
  - Rendering is fast enough for many lighting iterations

# Conclusions & Future Work

- Most of credit for the look of the fur in *Life of Pi* goes to the digital artists
- Main contributions of our rendering software:
  - A good level of realism is achievable
  - Results are highly art-directable
  - Rendering is fast enough for many lighting iterations
- Future work:
  - Improve hair scattering, including multiple scatter


Rendering Fur in
Life of Pi
Ivan Neulander
Google
Toshi Kato
Kevin Beason
Rhythm & Hues Studios


Rendering Fur in
Life of Pi
Ivan Neulander
Google
Toshi Kato
Kevin Beason
Rhythm & Hues Studios

# Rendering Fur in *Life of Pi*

Ivan Neulander
Google

Toshi Kato
Kevin Beason
Rhythm & Hues Studios