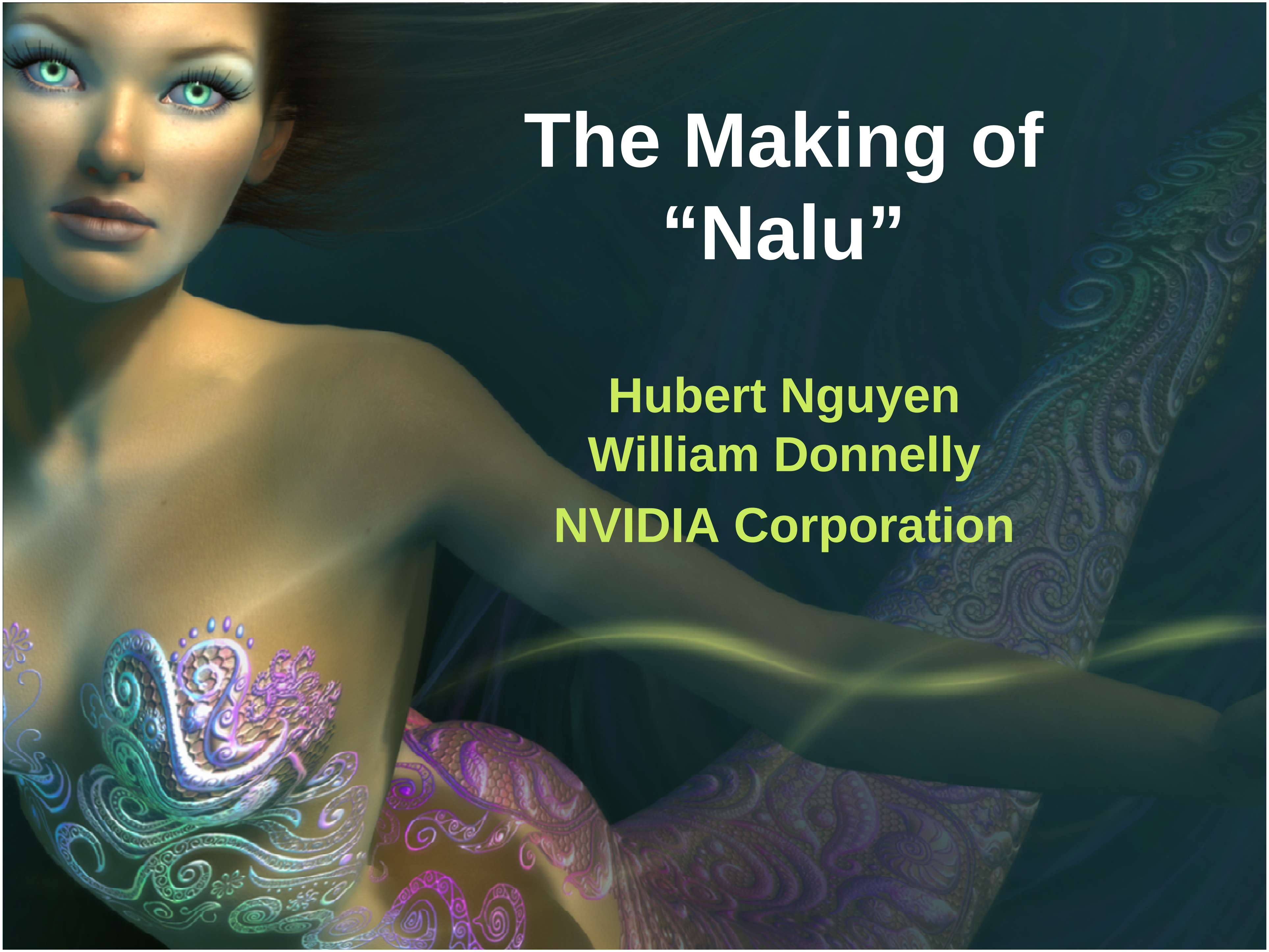
Secrets of the NVIDIA Demo Team

GeForce 6800

The NVIDIA Demo Team


The Making of “Nalu”
Hubert Nguyen
William Donnelly
NVIDIA Corporation

Long, Blonde Hair Rendering

Long, Blonde Hair Rendering

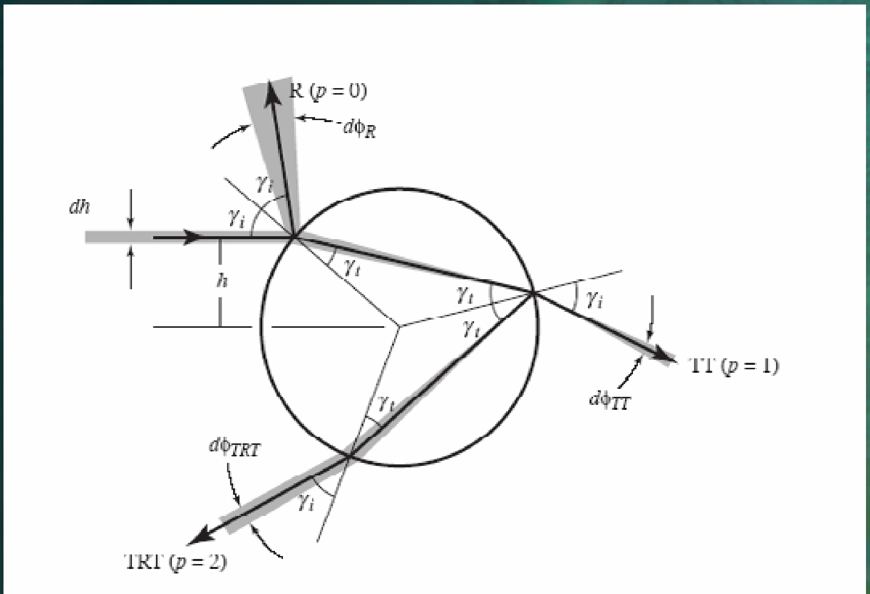
- **Long**
  - Requires dynamic animation
    - Thus cannot bake lighting
  - Requires lots of it
    - Thus shading has to be fast
- **Blonde**
  - Three visible highlights, black only has one
  - Shadows much more visible

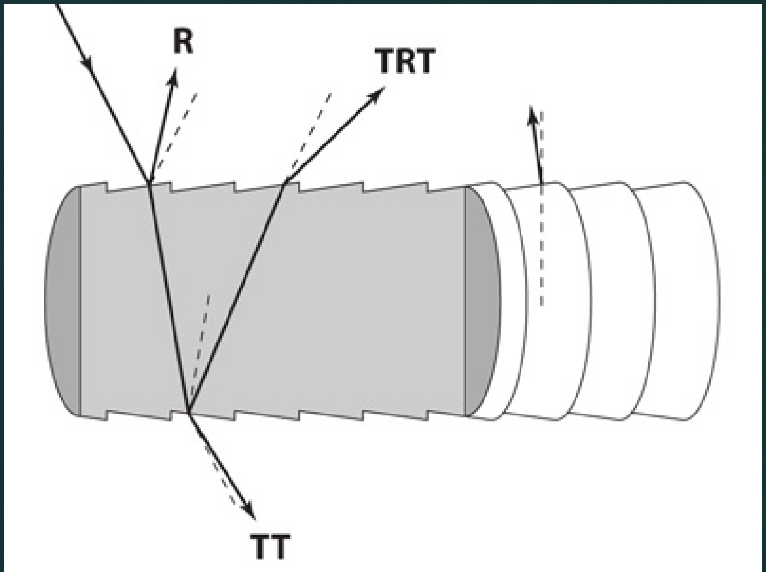
Acknowledgements

- "Light Scattering from Human Hair Fibers"
- By Steve Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan
- SIGGRAPH 2003

Paper Models three Distinct Highlights

- Consider only 3 most significant terms
  - R, TT, TRT





- Uses path notation
  - R is reflection
  - T is transmission

TT Highlight

- TT – strong forward scattering component
  - Important for underwater hair

The Reflectance Model

- **Hair model is a 4-dimensional function**
  - **2 light angles + 2 eye angles**
- **Factor into lower dimensional terms**
  - ```
       M_R (thetaH) * N_R  (thetaD, phiD)
    + M_TT (thetaH) * N_TT (thetaD, phiD)
    + M_TRT(thetaH) * N_TRT(thetaD, phiD)
    ```
- **2D functions are encoded in textures**
  - Texture maps are faster than heavy math
  - Use of mip maps eliminates “shader aliasing”
    ```

Shadowing

- Based on "Opacity Shadow Maps" (OSM)
- By Tae-Yong Kim and Ulrich Neumann SIGGRAPH 2001

Why Opacity Shadow Maps?

- **Opacity Shadow Maps vs Shadow Maps**

  "What percentage of light is blocked from here?"

  vs.

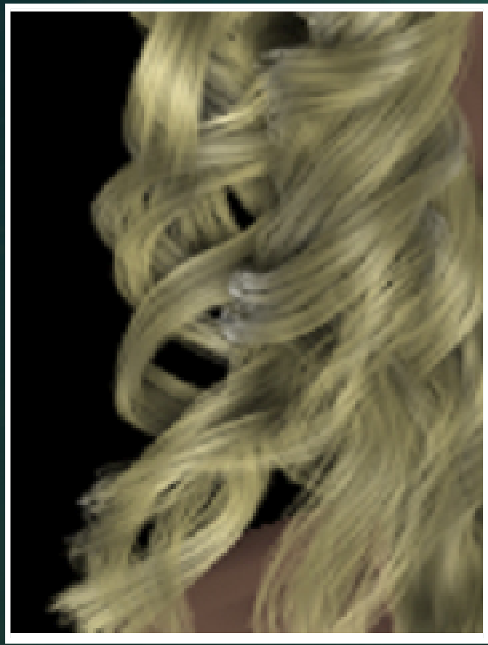
  "Is the light blocked from here?"

- **Thus supports AA edges and volumetric rendering**

- **Regular shadow maps alias around edges**
  - Hair is 100% edges!

Results from Kim & Neumann

No Shadows



15 slices

Opacity Equation

$$T(z) = e^{-\int_0^z k(z')dz'}$$

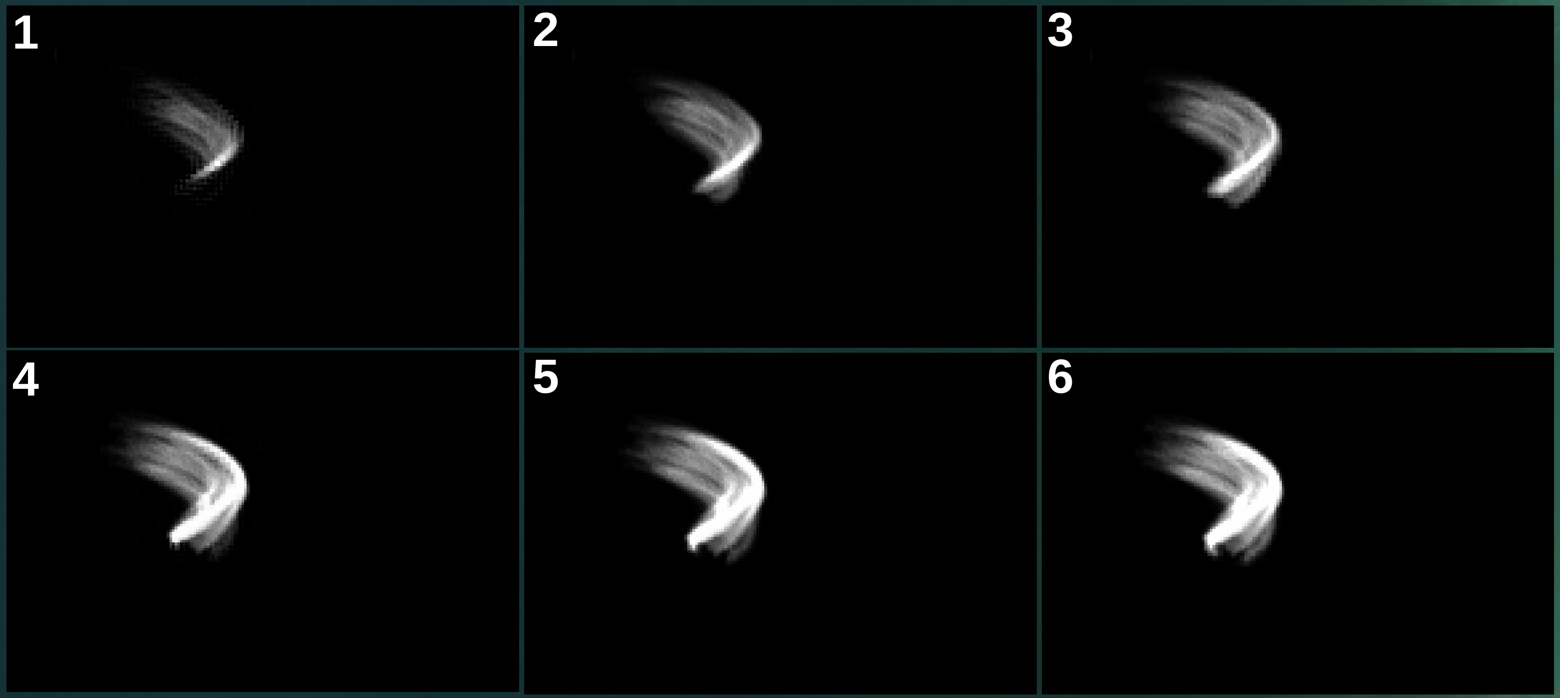
- **T(z): amount of light penetrating to depth z**
- **For discrete case (hair):**
  - Integral is sum over all strands between light and point being shadowed
- **Compute sum via additive blending**
  - "Extinction coefficient" K controls darkness of shadows

Creating the Opacity Maps

- **Choose 16 slicing planes in hair**
  - Uniform distribution based on hair bounding sphere
- **For each hair-pixel and for each plane**
  - Is hair-pixel closer to light than plane?
    - Yes: add hair to contribution (plane)
    - No: do nothing

OSM Creation

- **Render hairs to 16 slices**
  - Original implementation : 16 render passes (RP)
- **Can use lower hair LOD**



… up to 16

Vertex Shader Implementation

- **Compute light space position of the (Hair) fragment**
  - Add z-bias to counter limited z-resolution
- **Hair-pixel position in light space determines:**
  - Which opacity maps to look in (z)
  - Where in opacity map to look in (x,y)

Pixel Shader Implementation

$$T(z) = e^{-\int_0^z k(z')dz'}$$

We know the value of the integral at each plane

- Compute in-between values by linear interpolation
- Interpolated value is a linear combination of plane values

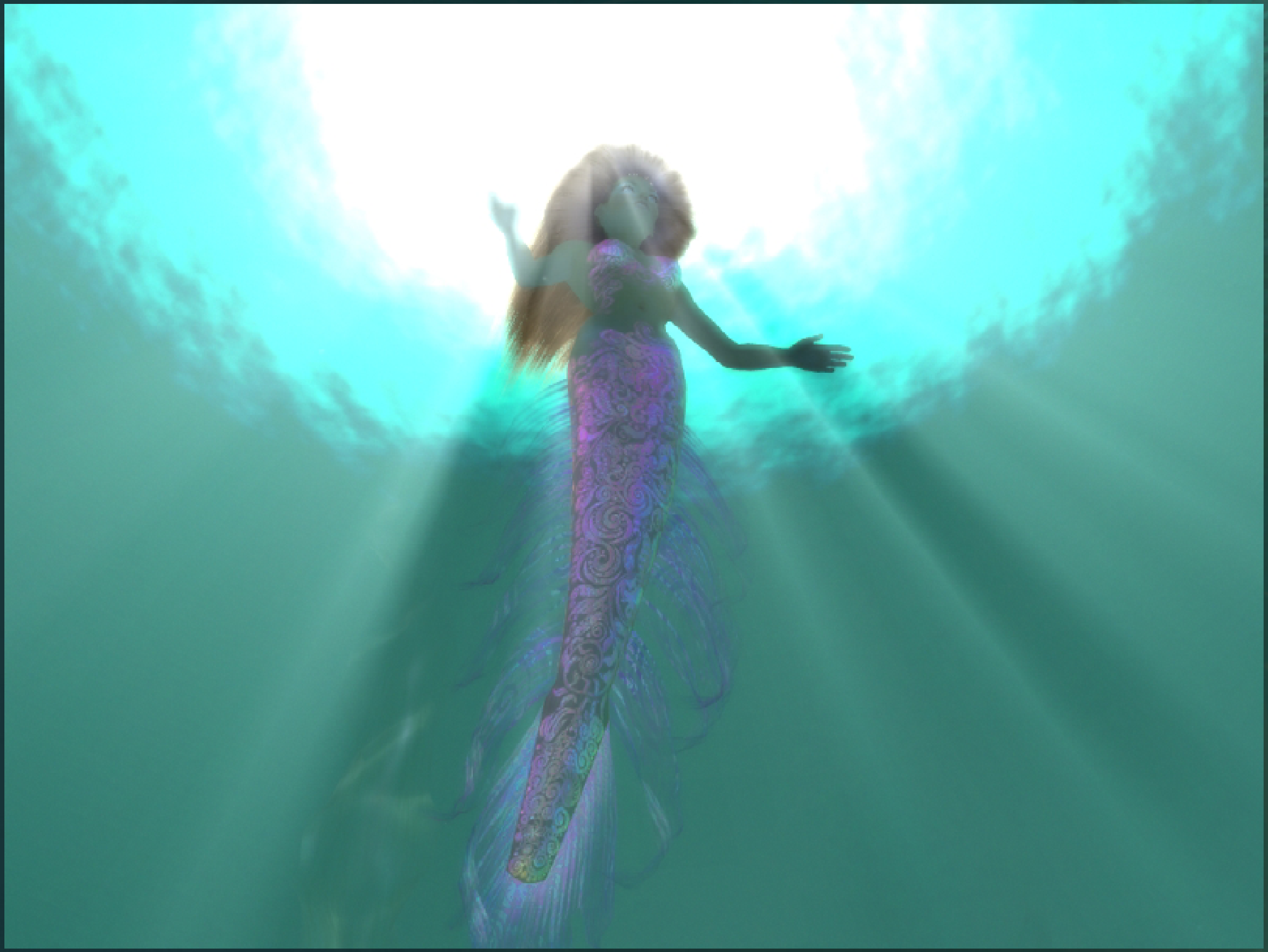
- **Compute opacity by exponentiation**

In the demo…

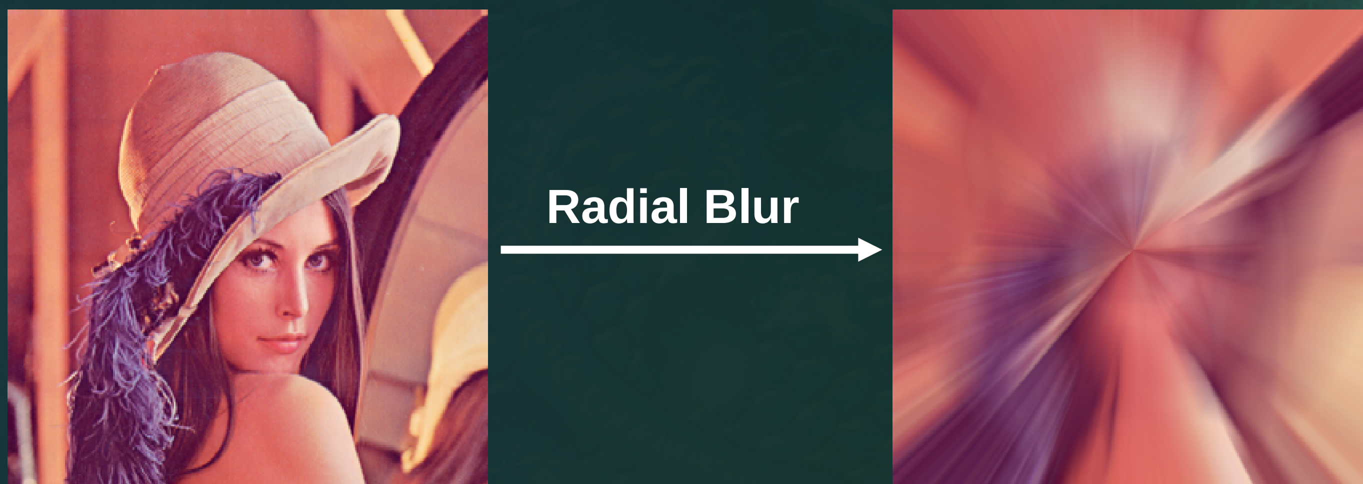
Hair WITHOUT Shadows

Hair WITH Shadows

Shafts of Light : "God Rays"

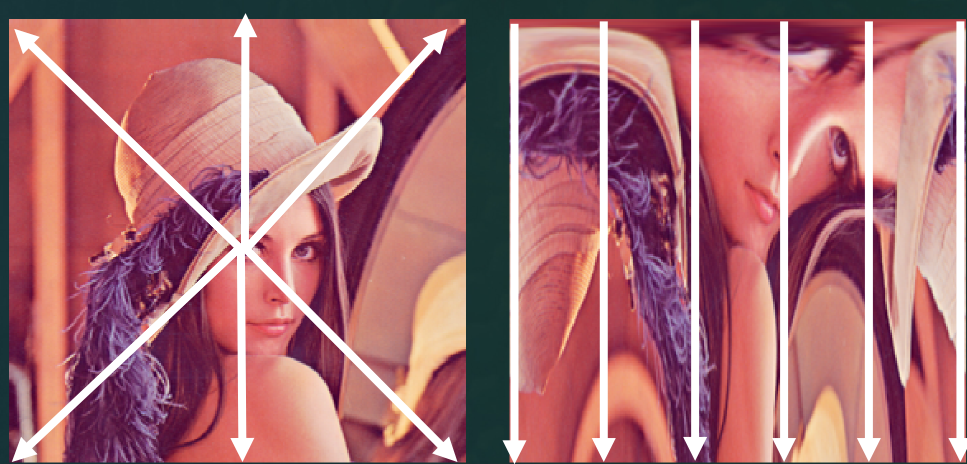


Shafts Are Based on a Radial Blur Effect



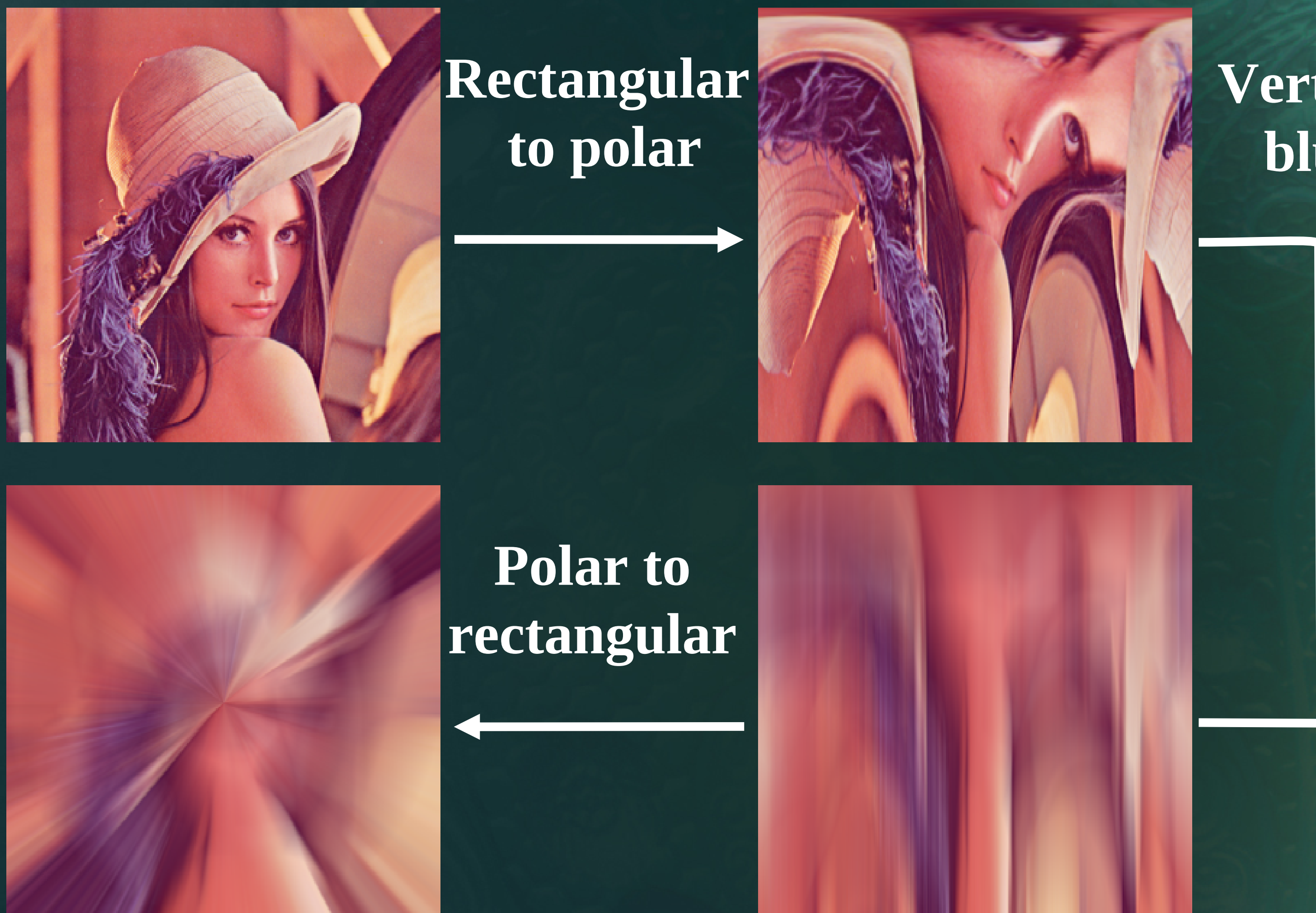
Radial Blur Effect

- Transform into polar co-ordinates (x,y)->(r,theta)
  - Use a grid where position = (r,theta) and texcoord = (x,y)
- Blur in the radial direction
- Transform back into Cartesian co-ordinates
  - Use the same geometric warping, but with positions and texture co-ordinates reversed



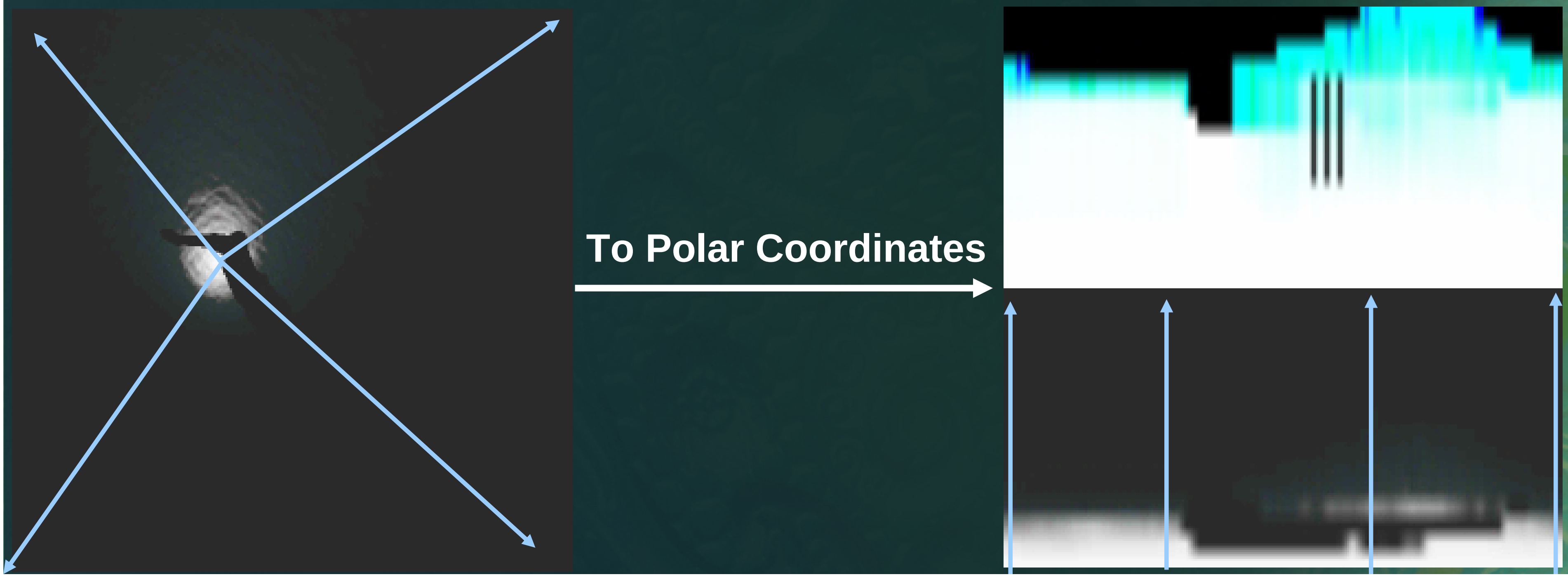
From Cartesian to Polar coordinates

Radial Blur Effect: Visuals



Radial Blur Effect: In the Demo

- Render bright regions
- Render shadow caster in alpha
- Blur in the radial direction, subtract occlusion factor (from alpha channel)



“God Rays” in the Demo

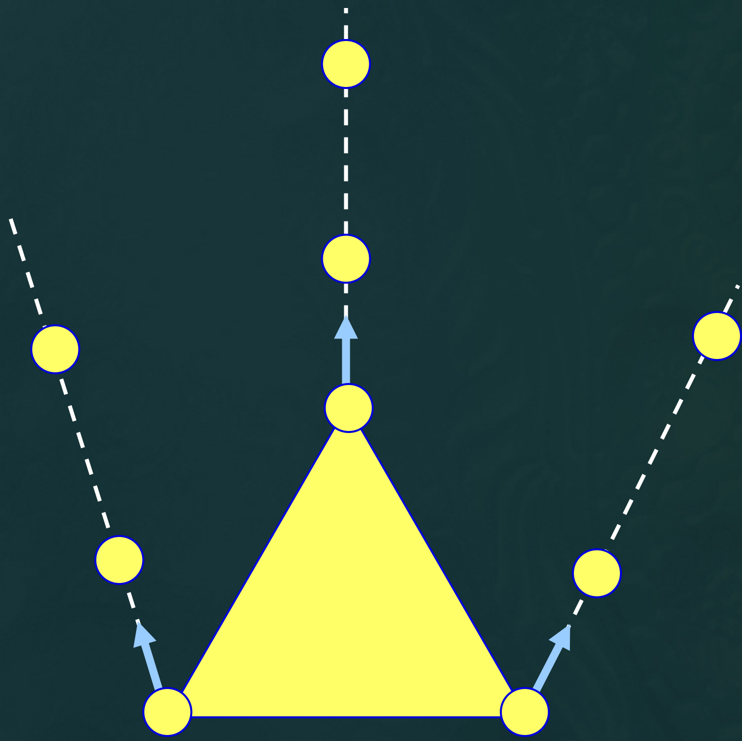
Hair Geometry & Dynamics

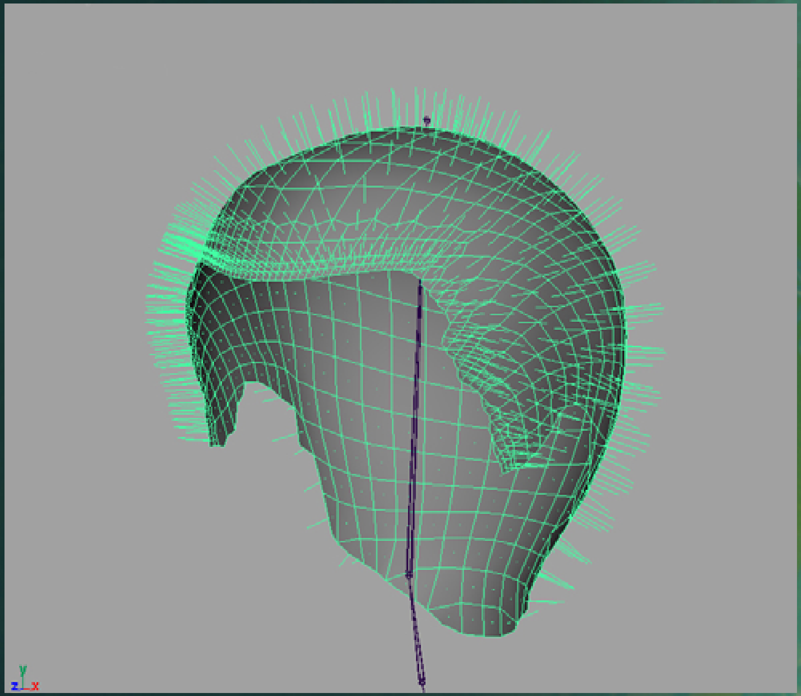
Hair Geometry: Overview

- 4095 individual hairs driven by 762 “control hairs”
- “Control hairs “
  - Set of hairs that is really driven by dynamics/collisions
  - Based on a particle system, where particles are connected by distance constraints.
  - Grown from a reference geometry
- “Fine hair” geometry is created by smoothing & interpolating the “control hair”.

Hair Geometry: Layout & Growth

- "Control hair" grows from a dedicated geometry





Hair Geometry: Control Hairs (left image)

- Physics/dynamics/collisions are performed on the control hairs

Control Hair

Fine Hair

Hair Geometry : Lifecycle (per frame)

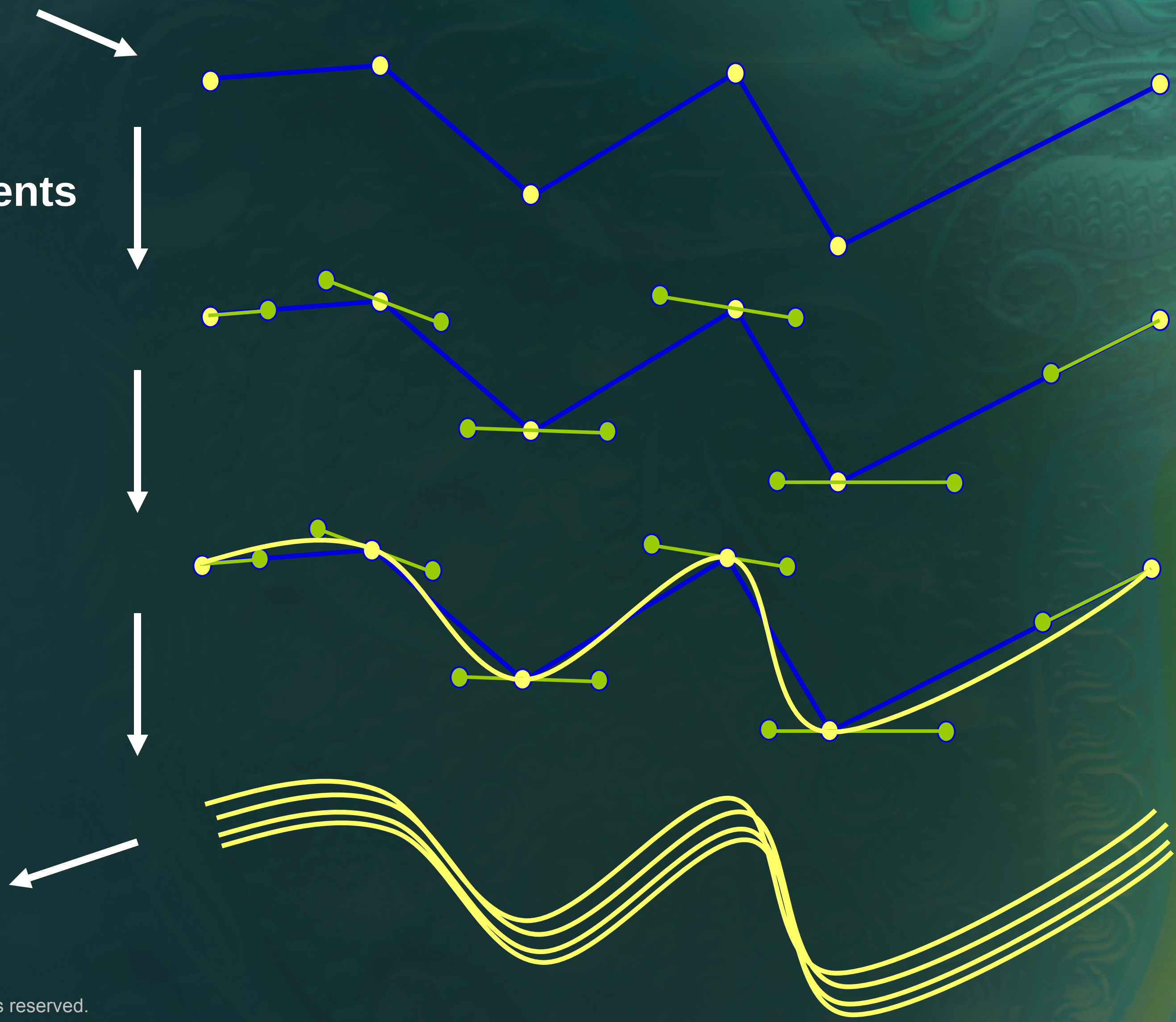
Dynamics

Compute tangents

Tessellate

Interpolate

3D API



Hair Dynamics

- Based on a particle system
- Uses the “Verlet” integration
  - previous frame position to compute velocity

$$\mathbf{x}' = 2\mathbf{x} - \mathbf{x}^* + \mathbf{a} \cdot \Delta t^2$$

$$\mathbf{x}^* = \mathbf{x}.$$

Reference: “**Advanced Character Physics**”
Thomas Jakobsen, IO Interactive, Denmark.

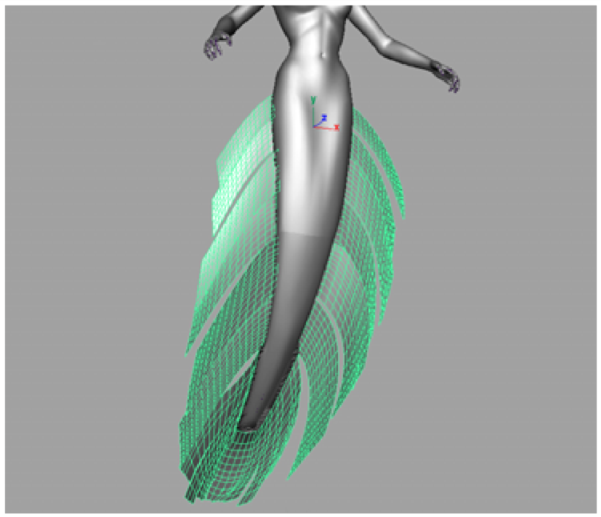
Hair Dynamics: Constraints

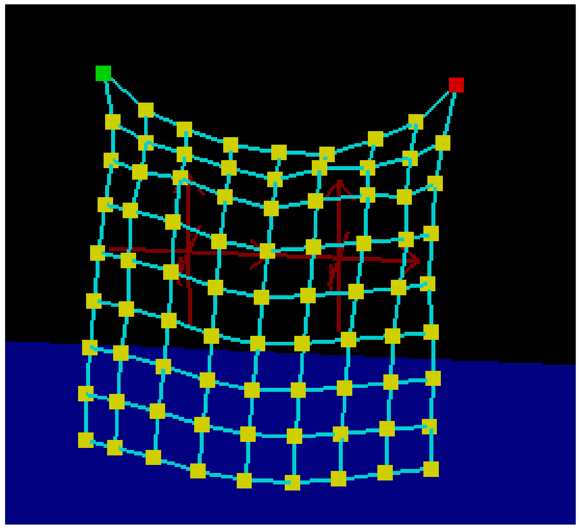
- Two constraint types:
  - Infinite mass: applied to the "hair root" particles. Allows the head to "pull" the hair.
  - Distance constraints: forces "control hair" segments length to stay constant

- If we apply those constraints iteratively, the particles will globally converge to the desired solution

Fins

- Fins are a cloth simulation.
  - Any mesh can be turned into a cloth by using triangle edges as constraints





Soft Shadows

- Based on
  "Texture Space Diffusion" see "GPU Gems":
  - Do the regular shadow mapping computations
  - But render in Texture Space
    - Using the UV coordinates as Vertex Shader Output Position
  - Blur the Texture Space B&W shadow result
  - Use the blurred shadow result in place of shadow compare when rendering the character

Soft Shadows: Visualizations

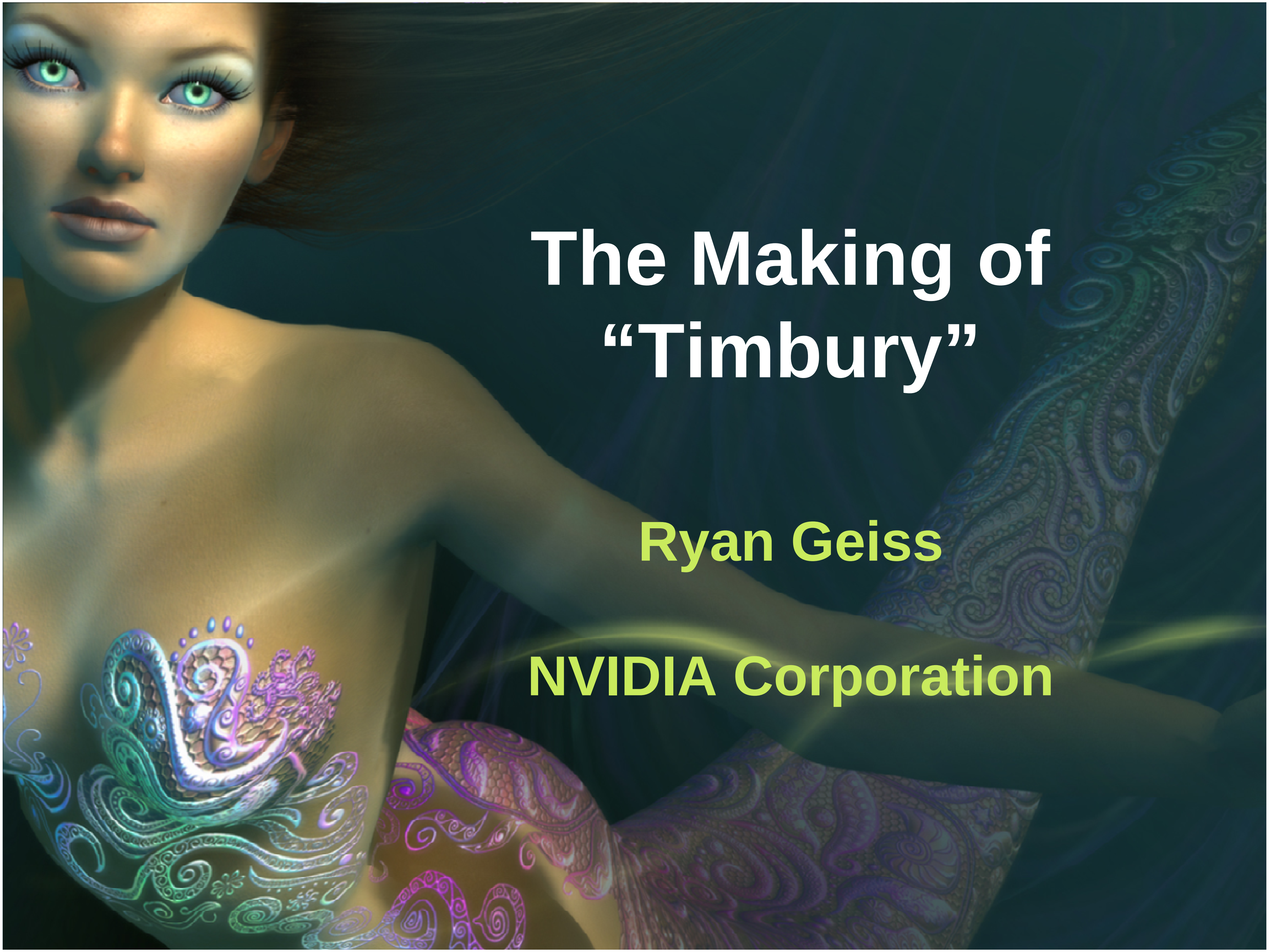
UV Space rendering

Soft Shadows: Challenges

- Unfold character in UV Space
- Visible Seams were UV are not continuous

Future Work

- Move more work to the GPU
  - Physics
  - Collisions (screenshot - Simon's cloth demo)
  - Curves Tessellation
  - Normal / Tangent computation
  - Hair Interpolation
  - Anything, really :)


The Making of
“Timbury”
Ryan Geiss
NVIDIA Corporation

Full Name:	Timbury Entonin Mudgett
Born:	Cleveland, England, 1896
Profession:	Entomologist
His deal:	He’s off to do “science.”

Technologies & Effects Used

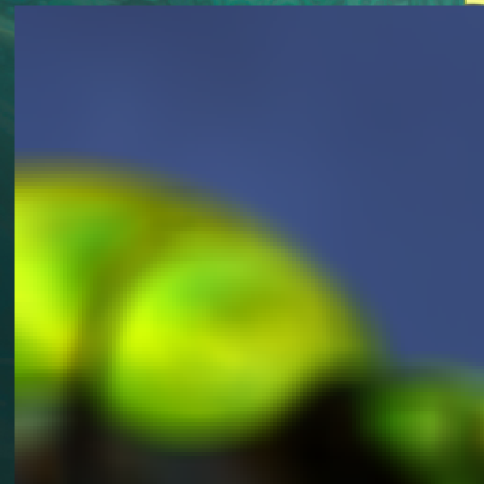
- High Dynamic Range (HDR) lighting
  - allows very high-precision lighting computations
  - Automatic Gain Control: camera responds to bright light
    - look at bright light -> normal objects become silhouettes
    - “regular” white tones (like a shirt) actually darken, but saturated whites (like the sun) *stay bright*
- Environmental lighting with fp16 cubemaps
- Post-processing “softening” of the image
- Animation: Skinning & Blendshapes
- Adaptive Subdivision Surfaces
- “Soft” (multi-tap) shadows
- Refractive Eyeglasses

The Rendering of a Frame

- Render scene to a texture
- Analyze how bright it was
- Make a copy and blur it profusely
- Mix the crisp and blurred together (to soften the image);
- Darken it (based on light analysis);
- Draw final image to the screen.

Rendering (More Detail)

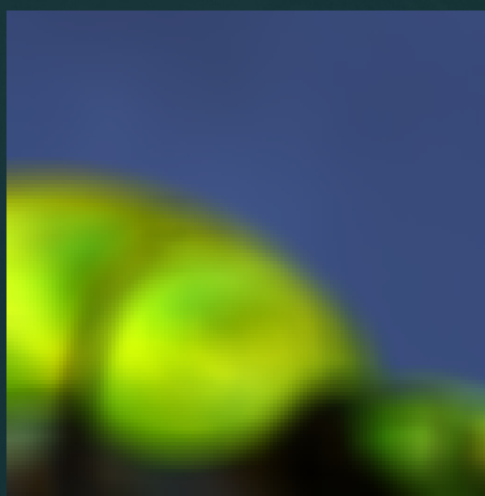
- First, render entire scene to a big fp16 texture.
- Render that to a 256x128 texture, sampling 3x3 source texels per (destination) pixel.
- Four more passes, all on 256x128 fp16 textures:
  - blur on x
  - blur on y
  - blur on x
  - blur on y
- (creates a nice near-gaussian blurred image) (but fast!)
- For final render pass, average this blurred image with the original crisp image for a nice, soft, cinematic look.

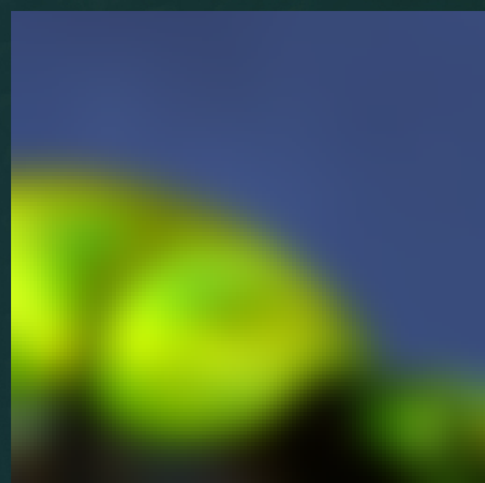


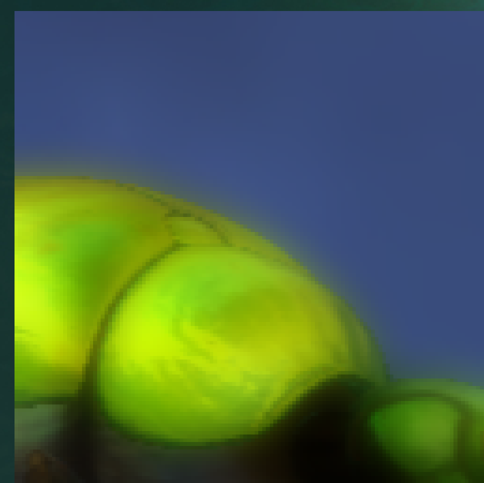
- original

blur/x

blur/y







- blur/x (2)

blur/y (2)

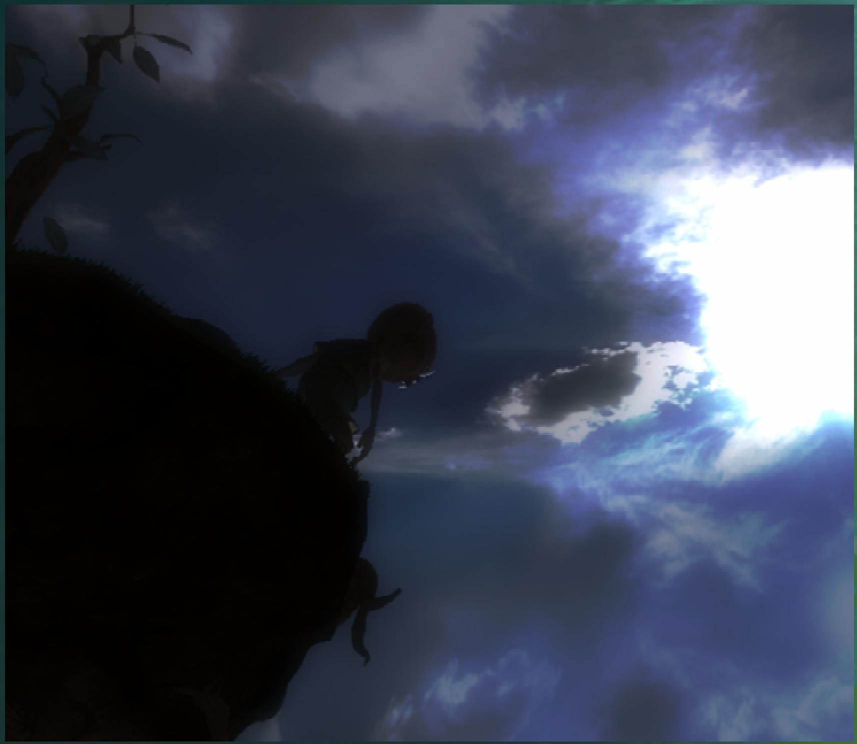
mixed

Automatic Gain Control (AGC)

- Mimics what a camcorder or human eye does in response to too much light: shrinks the aperture.
- The really bright part of our scene is the sun
  - about 40 times brighter than most colors in scene
- Looking around the scene, you can see the aperture open/close in response to how much light is coming in. [Demo]
- Look at the sun → aperture shrinks, turning most objects into silhouettes.

AGC Example

regular lighting



looking into the sun

Use of fp16 HDR Textures

- fp16 source textures include:
  - sky cubemap itself
  - all cubemaps used for env. lighting (1 diffuse, 8 specular).
- fp16 render targets include:
  - primary render target
  - all post-processing render targets.
- fp16 source textures were stored on disk using Industrial Light & Magic's **OpenEXR** file format.
  - Free source!: http://www.openexr.org
- **Advantages:**
  - high-quality lighting computations & postprocessing effects
  - enables AGC effect

Render Flow: Adding AGC

- To determine amount of light coming into the camera:
- Downsample the blurred image [from softening process] to an 8x8 image.
- Downsample again, to a 1x1 image.
  - complex fragment shader runs on just 1 pixel
  - takes 16 samples; with bilinear filtering, hits all 64 source texels.
  - give slightly more weight to samples near center.
  - for each sample, take the luminance: lum = dot(float3(0.3,0.48,0.22));
  - average light level for this frame: L = sum(lum values) / sum(weights)
  - write float3(1/L, 1/L, 1/L) as the output of the shader.

Render Flow: Adding AGC (cont'd)

- Shader for the final compositing pass:
  - mix the crisp & blurred images (~50/50)
  - scale that result by a sample from anywhere on the 1x1 texture (holding 1/L).

Linear vs. Nonlinear Tone Mapping

- Our AGC implementation finds the average luminance for all pixels, then scales the scene's brightness by its inverse – pretty simple.
- A more advanced tone mapping method* involves summing the $\log_2$ of the luminance values, then scaling in a special way:
  - AvgLumLog = $\sum \log_2$ luminance
  - AvgLum = $2^{AvgLumLog}$
  - in final pass: color' = grey * color / (1+color)
    - where grey is your "middle grey" value (~0.5).
- [Demo: hit 't' to activate tone mapping..]

* *from **Photographic Tone Reproduction for Digital Images** - Erik Reinhard, Mike Stark, Peter Shirley and Jim Ferwerda.*

log-sum & tone mapping:

linear sum & simple scaling:

(+) good distrib. of luminances

(-) poor distrib. of luminances

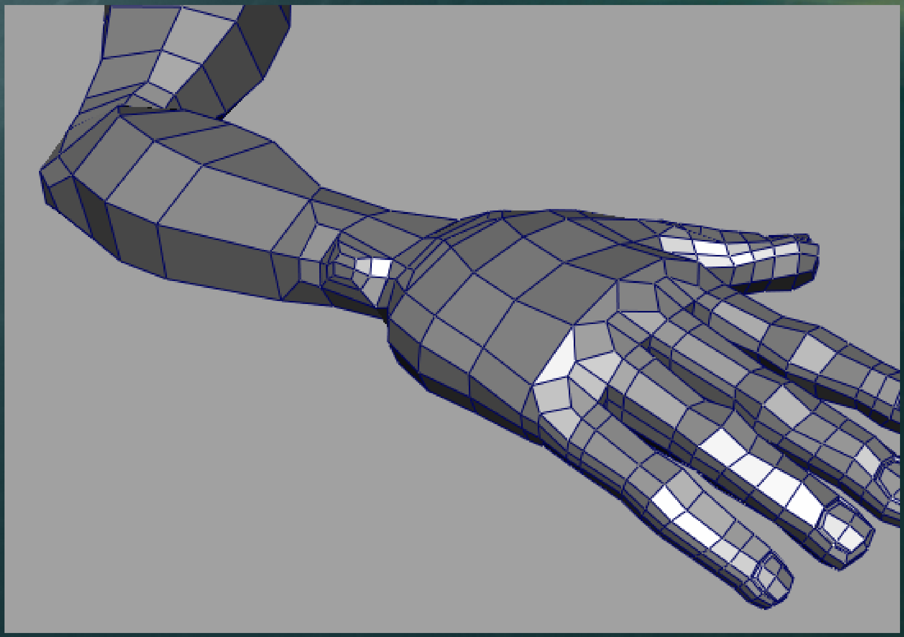
(-) poor contrast

(+) good contrast

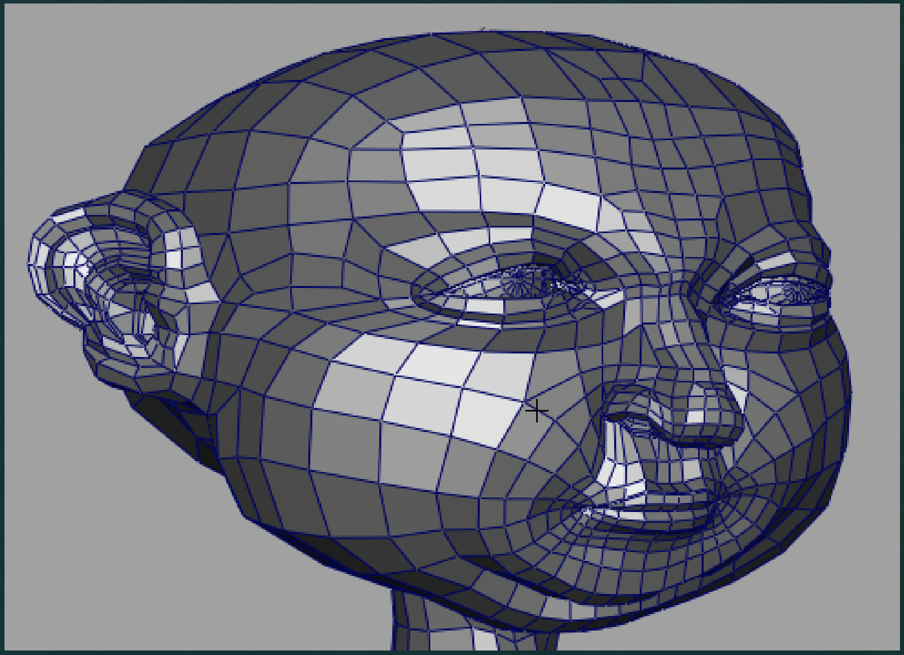
Adaptive Subdivision Surfaces

- Work by Michael Bunnell of Nvidia
- Goal: adaptively subdivide polygons to keep meshes looking good & drawing efficiently.
  - add polygons as you zoom in, as an elbow bends (increasing curvature), etc.

    [...Demo]
- Original mesh: prefer quads, but triangles work too
  - should be low-resolution, but still just high enough to describe essential features of the model *(...see next slide).*
- At startup, convert it entirely to Catmull-Clark patches (quads).
- **Goal:** each frame, adaptively subdivide polygons until the screen-space “error” is <= 1.0 pixels.



original control meshes for arm, hand



original control mesh for head

Adaptive Subdivision Surfaces (2)

- The “error” is how far the edges are from the ideal, smoothed surface.
  - Specifically, the error is the distance from the center of each edge, to the center of the ideal curved edge, in **world space**.
  - Project that distance value into screen space – this is the error, in pixels.
- If two opposite sides of a quad have high error, tessellate along those edges. likewise for the other two edges.
  - For things like cylinders or arms, produces tessellation in the direction where it’s needed most.

Links

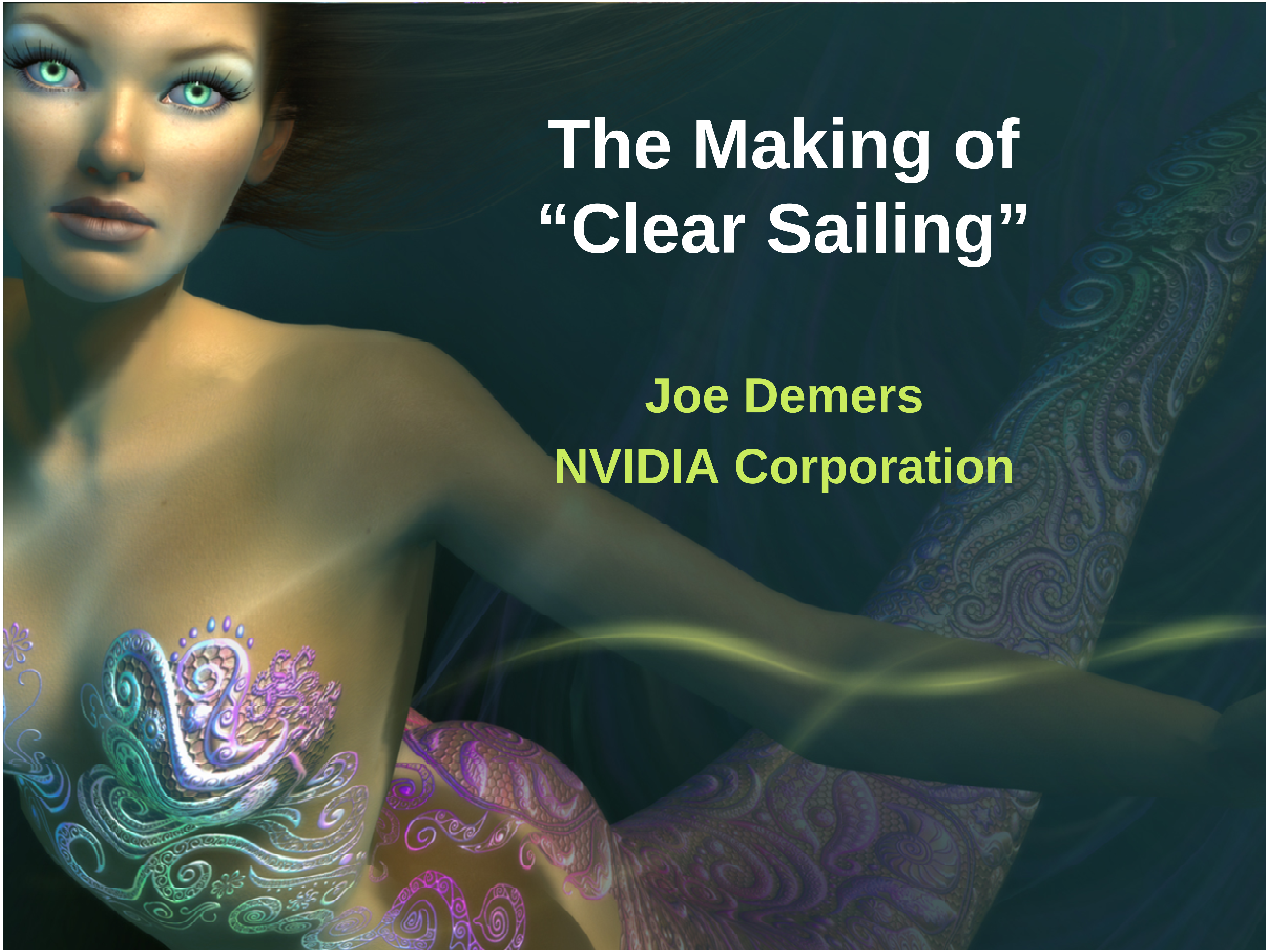
http://www.nzone.com/object/nzone_timbury_home.html

http://developer.nvidia.com/

http://www.openexr.org

Outtakes

Outtakes



The Making of “Clear Sailing”

Joe Demers
NVIDIA Corporation

Ocean Simulation and Rendering

- Our goal was to create a realistic looking ocean with choppy waves and a ship sailing on it
- This required us to:
  - simulate choppy ocean waves
  - tessellate the ocean surface
  - render ocean water (and foam)
  - calculate physics for the ship from the waves
  - render spray where the ship meets the waves

Ocean Simulation and Rendering

Ocean Simulation

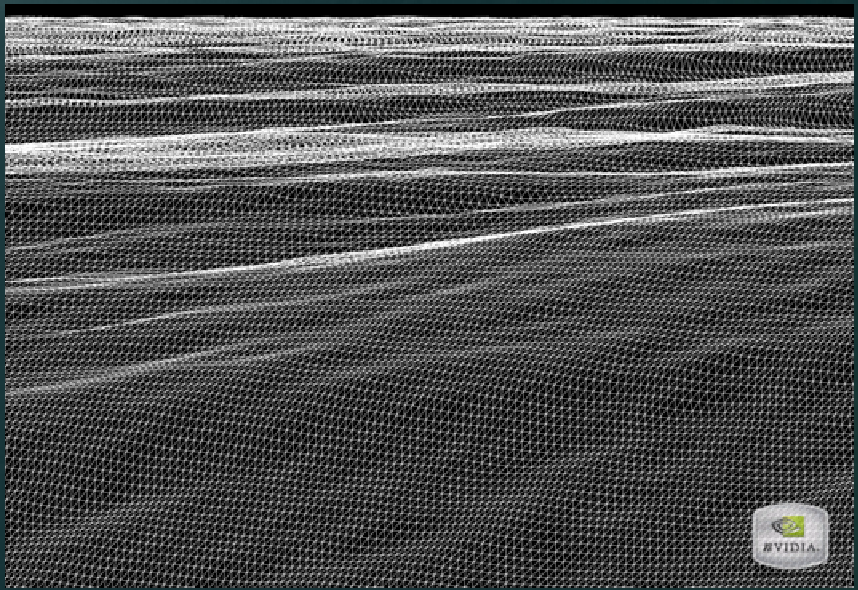
- The two common models in high-end ocean simulation are the Gerstner wave model and FFT-based statistical models
- We chose the Gerstner wave model for its simplicity and non-periodicity
- The Gerstner wave model moves points on the surface of the ocean in circles parallel to the direction of travel of the wave, allowing for ‘cusping’ as the wave height increases
- We found 45 Gerstner waves on a 150x150 grid gave us the best quality/performance tradeoff for the Clear Sailing demo

Ocean Tessellation

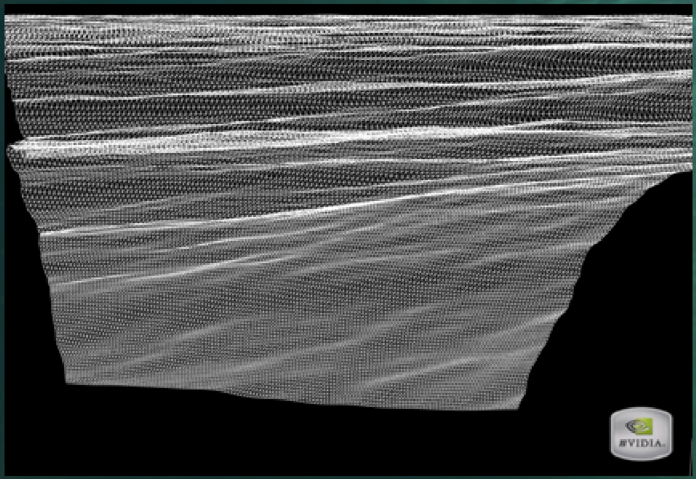
- Most previous techniques create regular grids of vertices in world space, and either tile the grid, or apply dense fog after the grid, or both
- We tessellated in eye space, mapping a regular grid to the intersection of the ocean plane and the camera viewport
- This allows us to only simulate and render geometry that is seen, and tessellate more finely in the foreground than the background

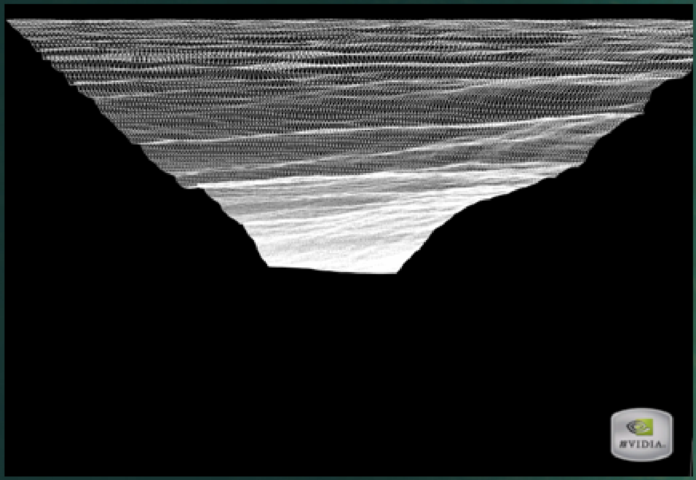
Ocean Tessellation

Screen-filling tessellation



Zoom out to see tessellation





Freezing the geometry and pulling the camera back allows us to see the actual geometry being drawn

Ocean Rendering

- Rendering deep water involves multiple sources of lighting
  - sunlight (directional light) reflected off the surface
  - sky light (cubemap texture light) reflected off the surface
  - scattered light (constant term) from below the water surface
- We can blend between the reflected and scattered terms using a simple fresnel function
- Unfortunately, fresnel exponents that look good along the surface tend to wash out the ocean when looking straight down, so a fairly low exponent works best when all viewing angles are possible

Reflecting the Ship

- The ship occludes reflected light in two ways
  - sunlight is occluded by using z-buffer shadows
  - skylight is occluded by ray casting into a 2d geometry imposter for the ship
- A little per-pixel noise breaks up the reflection and gives the illusion of higher frequency waves

Reflecting the Ship

Rendering Foam

- Foam is effectively a semi-transparent layer above the water surface
- Foam is generated (i.e. the foam layer is made opaque) where waves cusp, and also along the wake lines behind the ship
- We used render to texture with some additive blending to allow the foam to fade off over time

Rendering Foam

Ship Physics

- The ship is thrown about by the waves, but doesn't affect the water
- The ship moves up and down, and pitches and rolls, but doesn't slide along the ocean surface or turn left or right
- Above the water, the ship is affected by gravity and wind
- Below the water, the ship is affected by friction and a buoyancy proportional to the amount of water displaced
- The physics requires a lot of tuning/tweaking to get it to look right, but having tuning sliders also means you can have lots of fun
  - the ship moves like a toy boat if you increase the wave speed and decrease the scale that physics is computed at (effectively scaling the universe)

Ship Lighting

- The ship is lit from 4 light sources
  - direct sunlight, shadowed via z-buffer shadows
  - sky lighting, via diffuse and specular cubemaps
  - ambient light, baked global illumination from the sky
  - reflected light, a directional bluish light from below
- A diffuse color map gives the ship color
- Bump maps and specular maps give the ship more detail and give the appearance of more geometric complexity than there really is

Ropes & Sails

- The sails are two-sided, and softly glow when the sun is behind them
- It's very important for the facing-the-sun and away-from-sun shaders match when the normal is perpendicular to the sun, otherwise you'll see seams
- Ropes are drawn using lines, which change thickness depending upon distance from the viewer
- You can get a nice antialiasing effect by fading their transparency when the line thickness falls below 1 pixel
- Using alpha-to-coverage means you don't even need to sort!

Ropes & Sails

Splashes and Spray

- Splashes
  - emit particles when cannonballs hit water
  - render large particles with a texture of many small droplets
- Spray
  - when the ribs (or keel) of the ship move down through the water surface, emit particles between those ribs
  - faster motion generates more spray thrown further (higher)

Splashes and Spray

Smoke and Splinters

- Smoke
  - when the ship’s cannons fire, emit fairly large particles along the path of fire with animated textures (using a 3d texture)
  - smoke particles start moving very quickly, but then dampen their motion almost immediately, and slowly grow and fade
  - smoke particles "lit" by darkening lower half - cheesy, but looks good
- Splinters
  - when cannonballs hit the ship, generate lots of little triangles
  - move and tumble them with simple (but fast) verlet dynamics

Smoke and Splinters

Post Processing

- HDR-style glow
  - using a simple 8-bit single channel hdr effect
  - render overbrightness into the alpha channel
  - blur the alpha and add it back into the scene
- A little fog helps integrate the water, sky and boat together into the scene

Post Processing

Questions?

References

- Jerry Tessendorf. “Simulating Ocean Water”. SIGGRAPH 2001 Course notes
  - course notes no longer online, here are the slides:
  - http://probe.ocn.fit.edu/slides2001.pdf
- Hinsinger, D., Neyret, F., and Cani, M.P. “Interactive Animation of Ocean Waves”, Symposium on Computer Animation, 2002
  - http://www-imagis.imag.fr/Publications/2002/HNC02/index.gb.html