Real-Time Hair Rendering on the GPU

Sarah Tariq

NVIDIA

SIGGRAPH2008

NVIDIA

Motivation

- Academia and the movie industry have been simulating and rendering impressive and realistic hair for a long time
- We have demonstrated realistic real time results [Nalu, 2003]
- GPU is powerful and programmable enough to do all simulation/rendering

Results

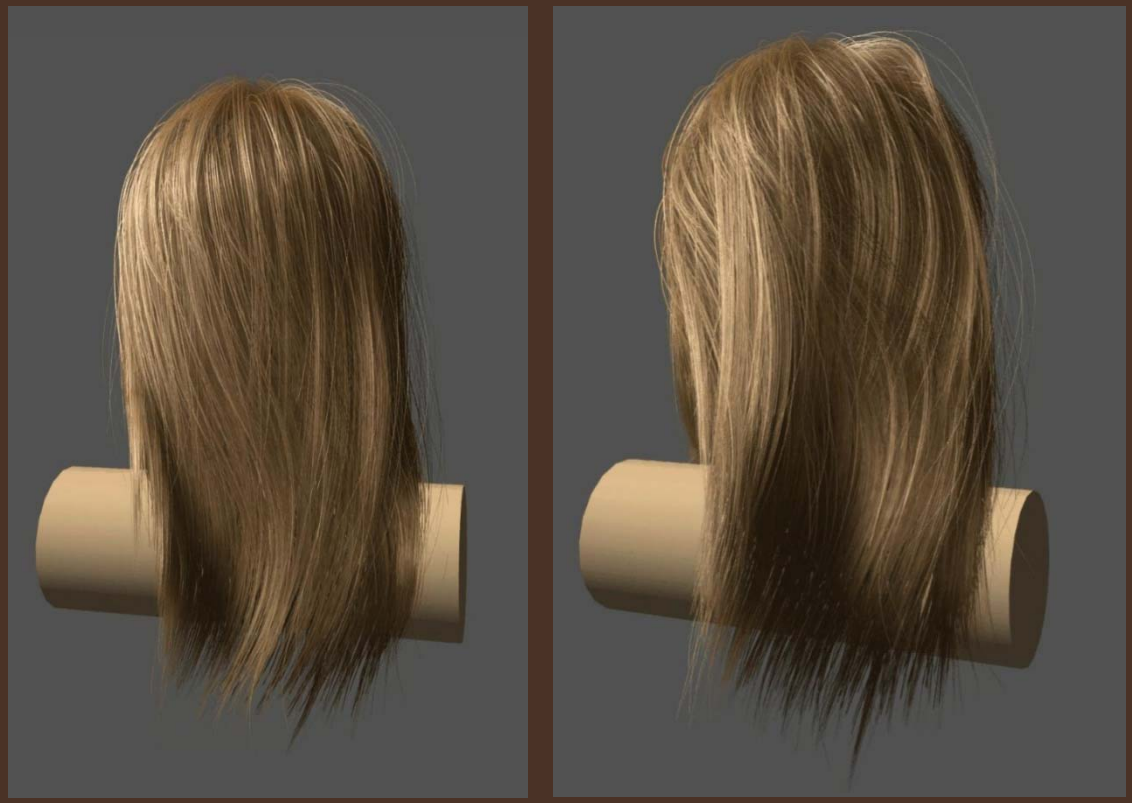
- 166 simulated strands
- 0.99 Million triangles

- Stationary: 64 fps
- Moving: 41 fps

- 8800GTX, 1920x1200,
- 8XMSAA

Results

- 166 simulated strands
- 2.1Million triangles
- 24fps
- 8800GTX, 1280x1024
- 8xMSAA
- 2xSSAA with 5 taps



- In this talk I will cover only hair rendering

- Real Time Hair Simulation and Rendering on the GPU
  Session: Lets get physical

  Thursday Room 502B. 1:45-3:30

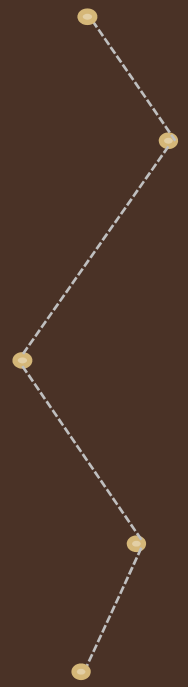

NVIDIA.
Import Guide Hair
Simulate Guide Hair
Tessellate and Interpolate Guide Hair
Render Final Hair
SIGGRAPH2008

Tessellation and Interpolation

SIGGRAPH2008

NVIDIA

Tessellation



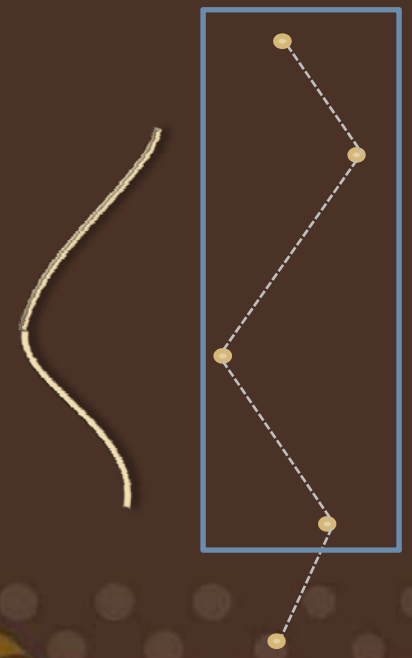
Simulated Vertices

Smoothly Tessellated Hair

Tessellation

- We use B-Splines
  - Uniform cubic b-splines
    - Pre-compute and store partial results
  - Automatically handle continuity
  - Do not interpolate endpoints
    - So we repeat end points

$$x(t) = \frac{1}{6}\begin{bmatrix} P_0 & P_1 & P_2 & P_3 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$



Interpolation

Multi Strand Interpolation

Clump Based Interpolation

Interpolation

- Clump Based Interpolation
  - Each interpolated strand is defined by
    - 2D offset that is added to the guide strand in the direction of its coordinate frame. Pre-computed and stored in constants
    - Clump radius which changes along the length of the guide strand
- Multi Strand Interpolation
  - Each strand is defined by 3D weights which we use to combine the 3 guide strands

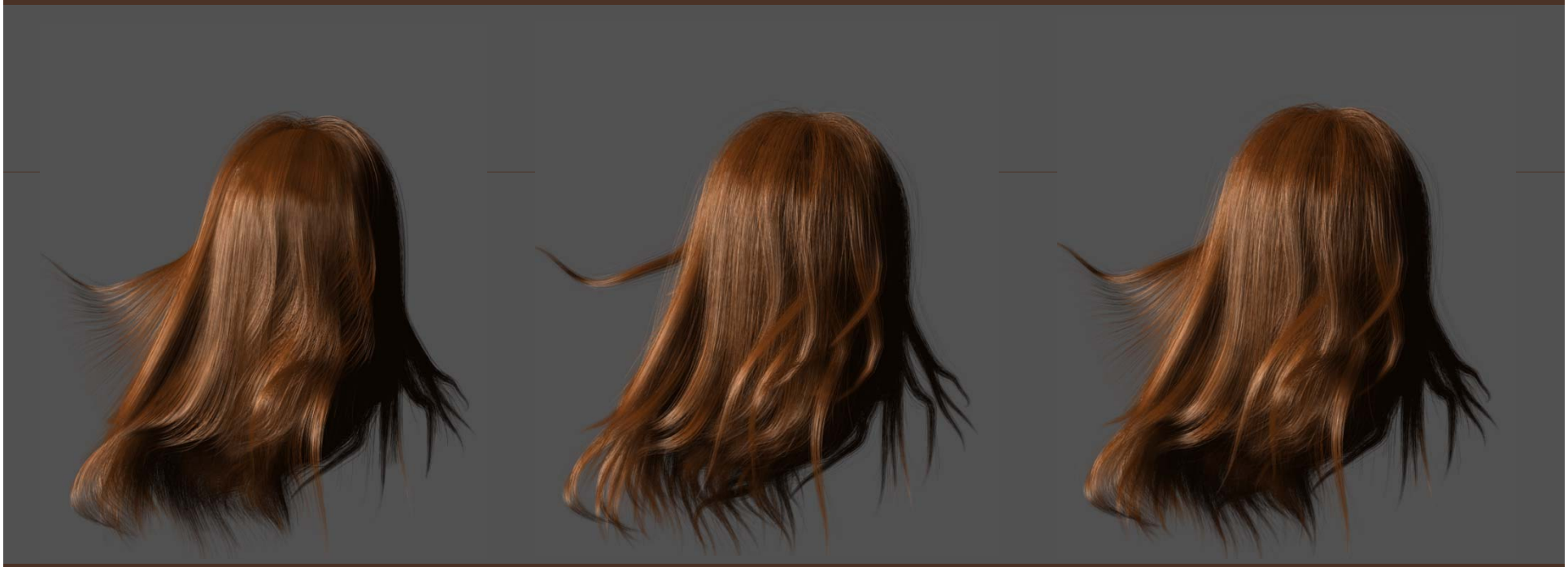
Interpolation

Multi strand Interpolation

Clump Based Interpolation

Combination

Interpolation



Multi strand Interpolation

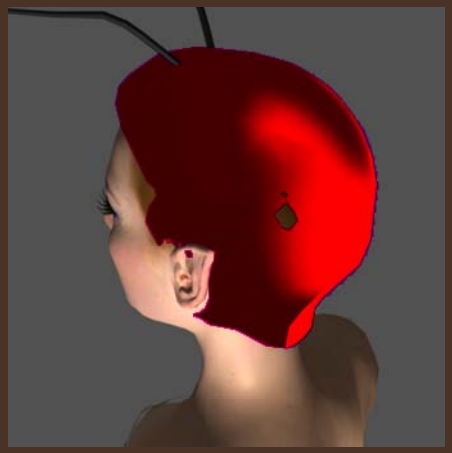
Clump Based Interpolation

Combination

Modulate density across scalp

- Red: Local density of hair

- For example in this demo
  - Multi strand based hair has higher density near the center of the head



Multi Strand Interpolation

Clump Based

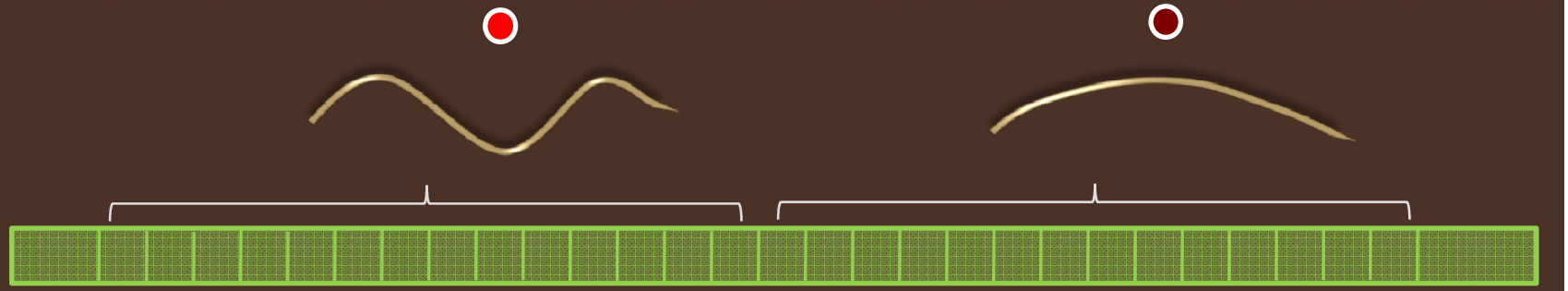
Process

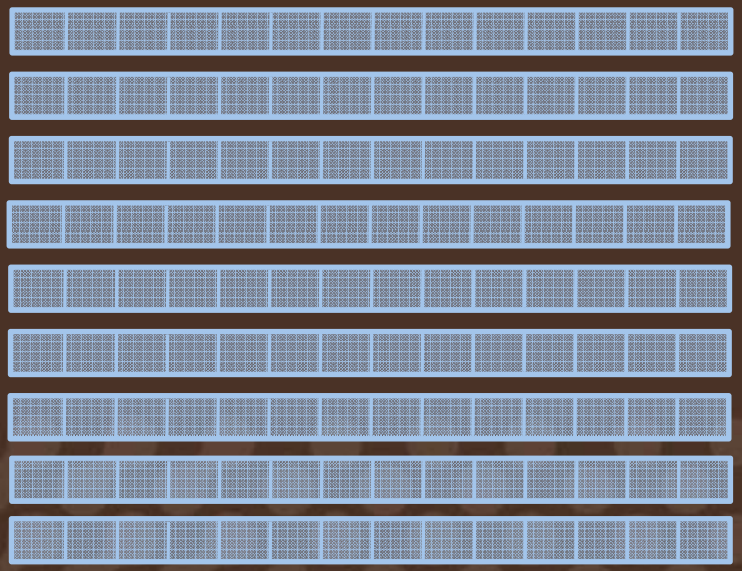
- Create a tessellated dummy hair and render it N times, where N is the number of final hairs
- In the VS, load from Buffers storing simulated strand attributes
  - Constant attributes: strand texcoords, length, width etc
  - Variable attributes: vertex positions, coordinate frames

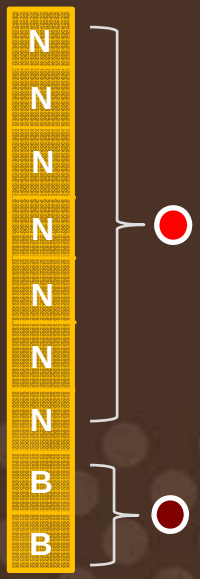
Process

- Stream out the data after each stage to minimize re-computation
  - Tessellate the simulated strands and Stream out
  - Interpolate the tessellated strands and Stream out
  - Render final hair to shadow map
  - Render final hair for rendering
- Each stage uses data computed and streamed out from previous stage

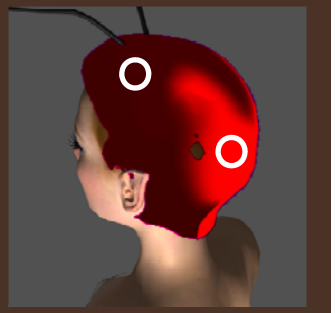
Indexing







Indirection
Buffer



SIGGRAPH2008

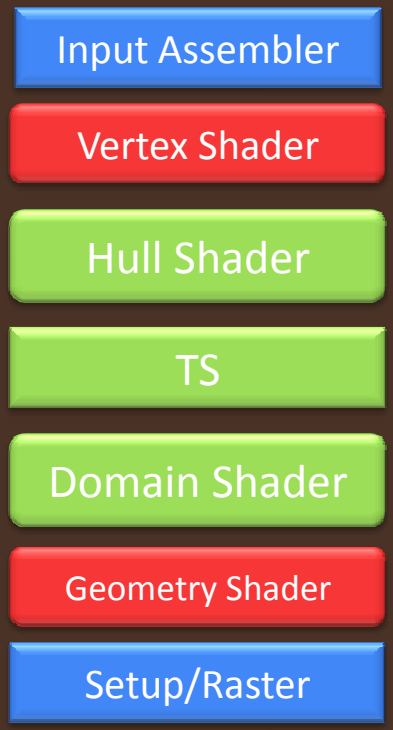
Using Dx11 Tessellation Engine

SIGGRAPH2008

NVIDIA

Tessellation Pipeline

- Direct3D11 extends Direct3D10 with support for programmable tessellation
- Two new shader stages:
  - Hull Shader (HS)
  - Domain Shader (DS)
- One fixed function stage:
  - Tessellator (TS)



ISO Lines

- Output from the tessellation engine will be a set lines of equal number of segments

- We can either render these directly
- Or we can expand these to triangles in the GS

ISO Lines

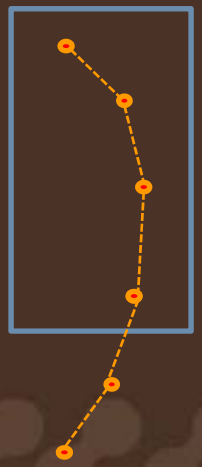
- Input an arbitrary patch
- For each patch output a number of lines with many segments per line
  - The number of lines output per patch and the number of segments per line are user controlled and can be different per patch
  - The positions of the vertices of the line segments are shader evaluated

Interpolating and Tessellating hair

- With Tessellation engine we can create tessellated and interpolated hair on the fly
- Benefits:
  - Easy and intuitive
  - More programmable
    - Can create geometry only where needed
    - Reduce detail where not needed
  - Continuous LOD

Pipeline

Input



Patch of Simulated Guide Hair

HS — Calculate LODs

TS — Generate topology

DS — Calculate vertex attributes

GS — Expand lines to quads

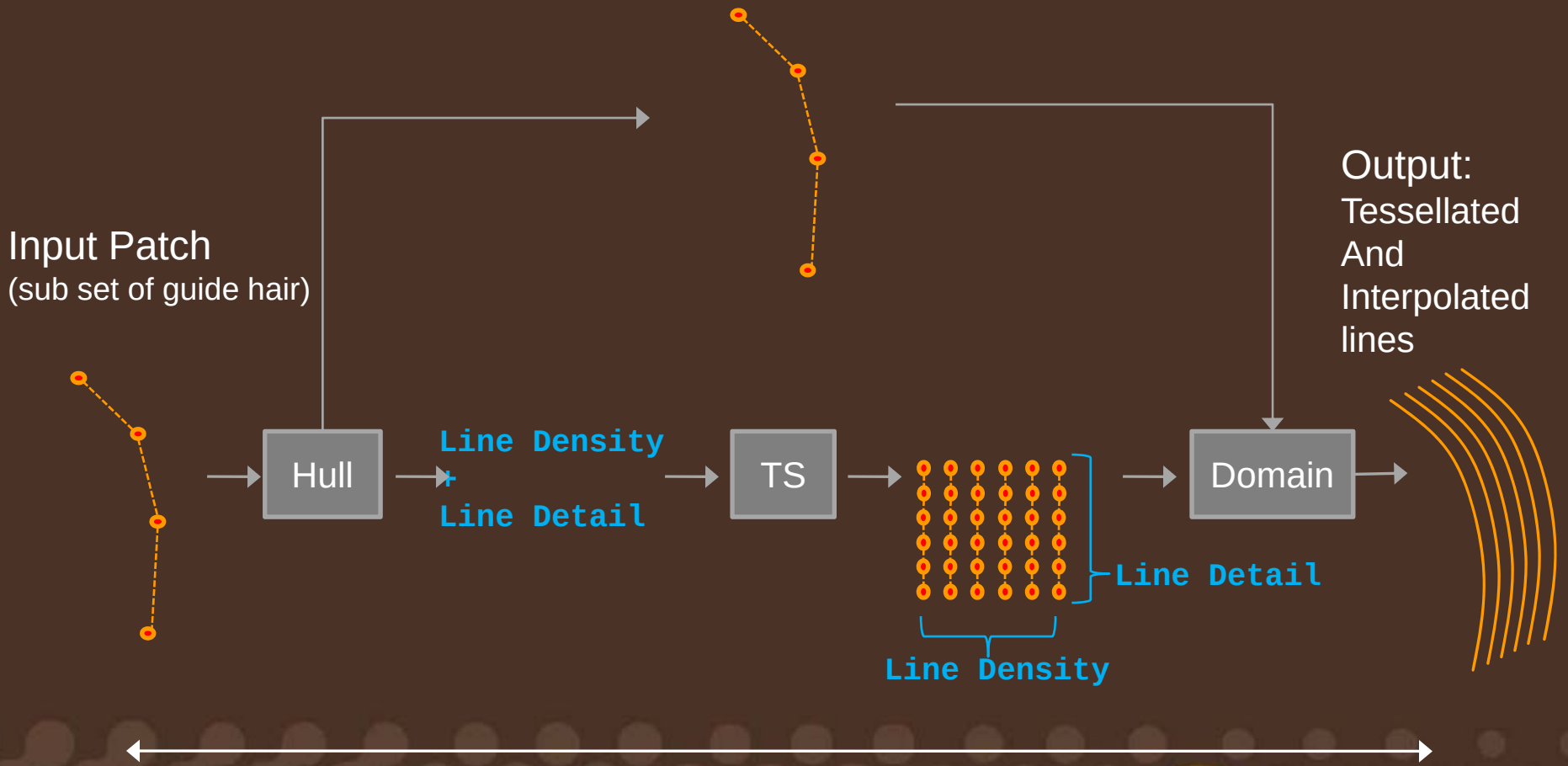
PS — Shade

Output

Tessellated, Interpolated, Rendered Hair

SIGGRAPH2008

Clump Based Hair Tessellation and Interpolation



Single Pass on GPU

Alternative Pipeline

Simulated Guide Hair

HS

Calculate LODs

TS

Generate topology

DS

Calculate vertex attributes

SO

Tessellated Guide Hair

HS

Calculate LODs

TS

Generate topology

DS

Calculate vertex attributes

GS

Expand lines to quads

PS

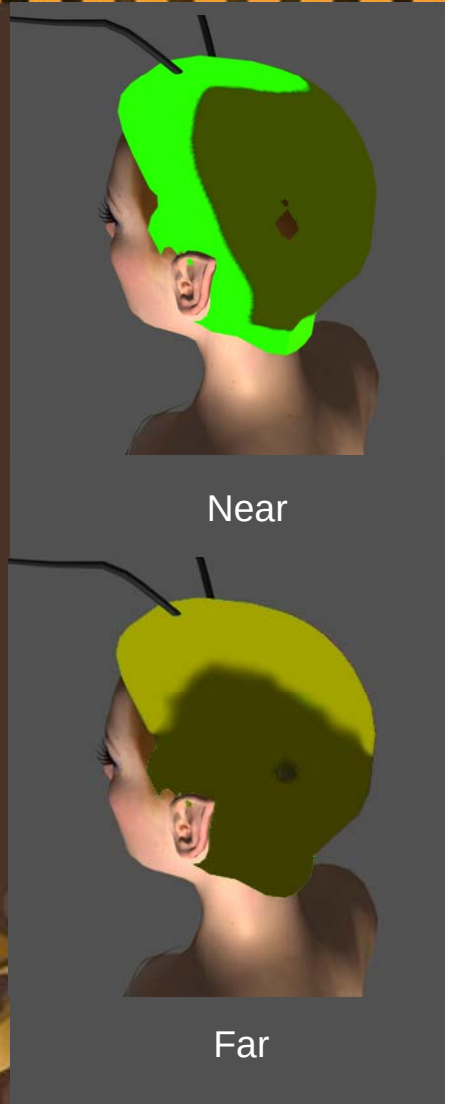
Shade

LOD

- Can use the distance of patch from camera to decide on the LOD
  - Low LOD levels would use
    - less number of lines
    - thicker lines
    - less segments per line
    - less complex shading

LOD

- LOD can be procedural
  - For LOD 0.5 render only 50% of lines in a given patch
- LOD can also be artist defined
  - Artists can create density/width maps for different LOD of the hairstyle
  - The Hull Shader can lerp between appropriate LOD textures to decide on the line density, and line thickness



Near

Far

Rendering

SIGGRAPH2008

NVIDIA

Rendering

- Lines have issues:
  - No floating point line width
  - No textures across line
    - These are useful for simulating the look of many hair
    - Rendering hair with complex color variations
- Render camera facing triangle strips
  - Can either expand lines to strips in the GS
  - Or can render instanced triangle strips

Shading: Kajiya and Kay

- Kajiya and Kay *[Rendering fur with three dimensional textures* (SIGGRAPH '89)]

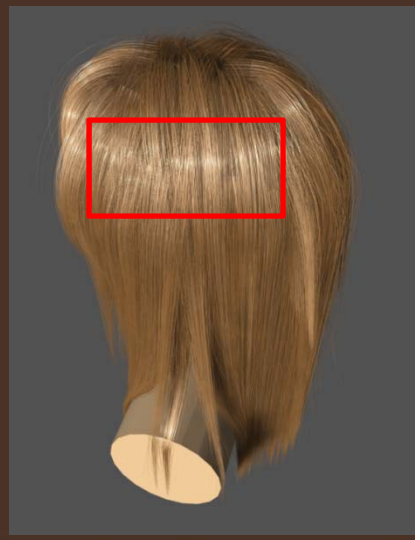
- Diffuse = $\sin(\mathbf{T},\mathbf{L}) = \sqrt{1 - (\mathbf{T} \cdot \mathbf{L})^2}$
  Specular = $[\mathbf{T} \cdot \mathbf{L} * \mathbf{T} \cdot \mathbf{E} + \sin(\mathbf{T},\mathbf{L}) \sin(\mathbf{T},\mathbf{E})]^p$
  $= [\mathbf{T} \cdot \mathbf{L} * \mathbf{T} \cdot \mathbf{E} + \sqrt{1 - (\mathbf{T} \cdot \mathbf{L})^2} \sqrt{1 - (\mathbf{T} \cdot \mathbf{E})^2}]^p$

- Ivan 2006
- fake dual specular highlights
  - primary highlight shifted towards tip
  - secondary highlight shifted towards root

Tangents

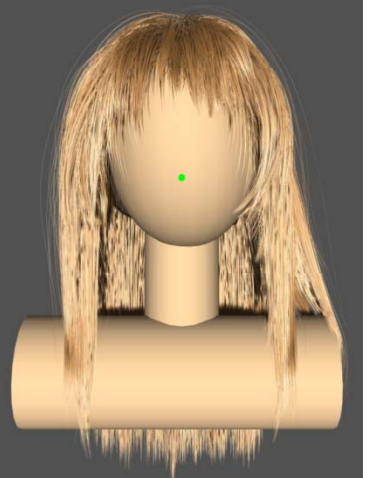
- Need to have smooth tangents
  - Calculate tessellated and interpolated tangents
- Add jitter to tangents in order to break strong highlights
  - Randomly a per strand constant bias to tangents towards or away from root
  - Add per pixel noise to tangents

$$x(t) = [T_0 \quad T_1 \quad T_2] \begin{bmatrix} 0.5 & -1 & 0.5 \\ -1 & 1 & 0.5 \\ 0.5 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}$$

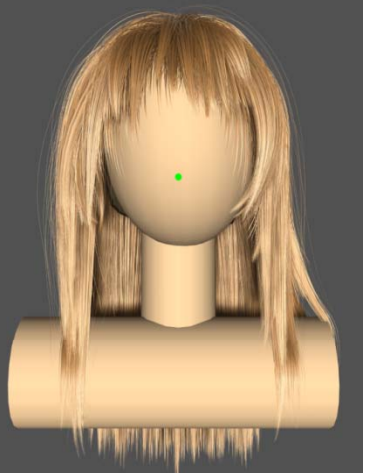


Shadows

- Material Model: Opaque hair
- Essential Requirements
  - No flickering, smooth shadows
  - Soft Shadows
- Do PCF with multiple taps
  - tShadowMap.SampleCmpLevelZero(ShadowSampler, texcoord, z, int2(dx, dy));
  - Helps reduce temporal/spatial aliasing
  - Calculate shadows in VS and interpolate across hair length to further reduce aliasing



1 tap

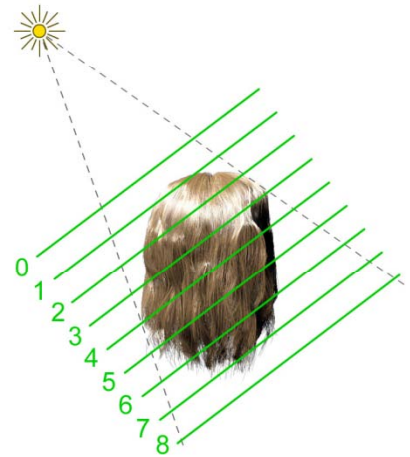


36 taps

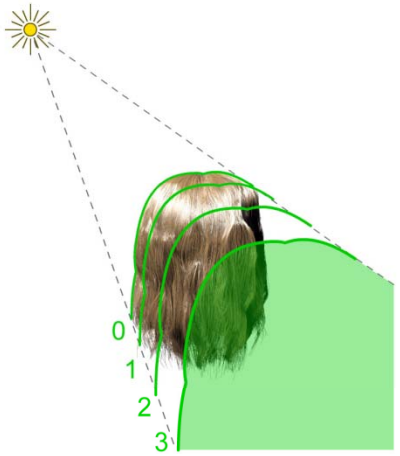
Shadows

- Material Model: Translucent Hair

- If hair is semi-transparent then we need volumetric shadows
  - [Yuksel and Keyser 08], [Kim and Neuman 01], [Lokovic and Veach 01]
  - discritize the space into layers



Opacity Shadow maps

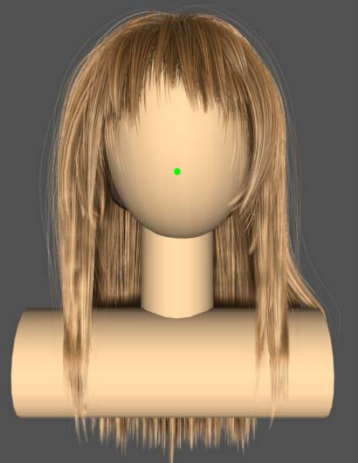


Deep Shadow Maps

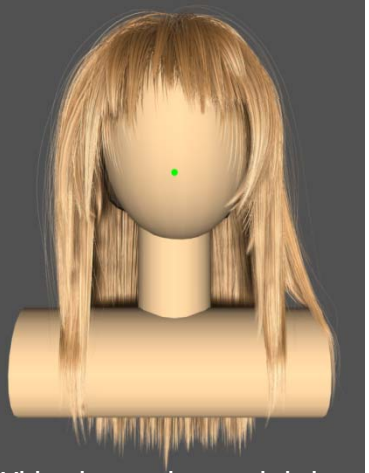
[Images courtesy of Yuksel 08]

Shadows

- We do absorption weighted PCF
  - Similar to [Halen 06]
  - Weigh the PCF sample by
    1 - exp(g_SigmaA * d)
    - d is the difference between the depth of the current shaded point and the closest point to the light



No absorption weighting



With absorption weighting

Antialiasing

- Human hair is very thin
- Typically alpha blending is used to hide aliasing
  - Requires sorting geometry which is time consuming
  - Can use depth peeling [Everitt 01], [Bavoil and Myers 07]
    - Scalable: can decide to render only the first 4 depth layers for example
  - Or [Sintorn and Assarsson 08]

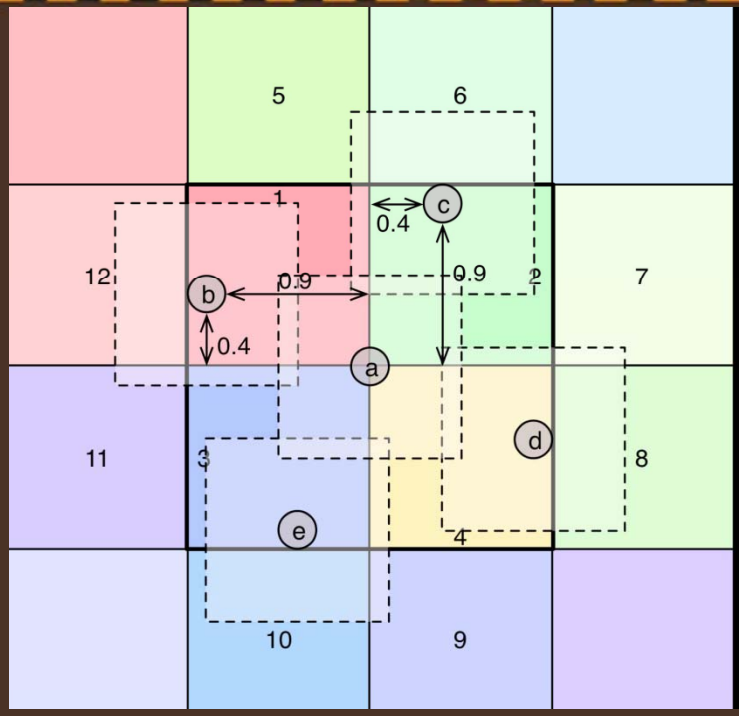
No Alpha Blending

With Alpha Blending

[Sintorn and Assarsson 08]

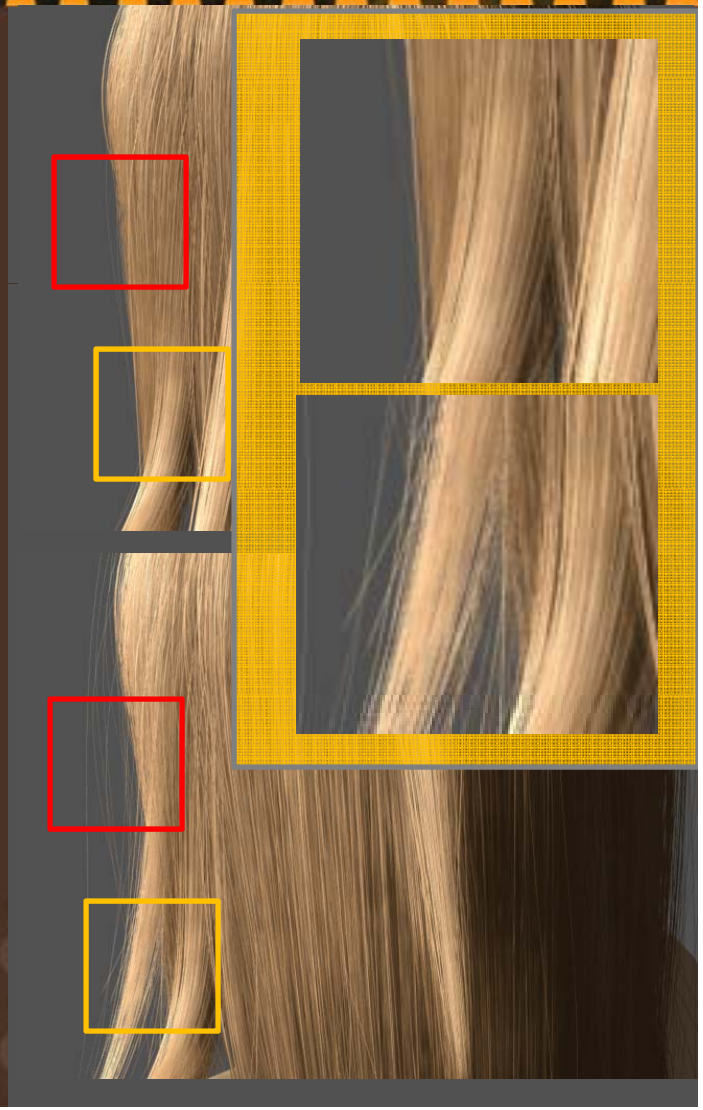
Antialiasing

- Can also use Alpha To Coverage
  - Does not require sorting
  - Does require MSAA
  - Need depth pre-pass to get earlyZ
- We use a combination of MSAA and SSAA
  - 8xMSAA
  - 2xSSAA with 5 taps



Add random deviations to hair

- Pre bake and store deviations which are added to interpolation offsets along the length of the hair
  - Most hair deviate towards the tips
  - Some very deviant and thin hair
- Other
  - Taper hair width towards the hair tip
  - Randomize width per-hair strand



Thank you!

SIGGRAPH2008

NVIDIA