Hair Rendering and Shading

Thorsten Scheuermann
3D Application Research Group
ATI Research, Inc.

Overview

- Hair rendering technique using polygonal a hair model
- Shader: Mix of
  - Kajiya-Kay hair shading model
  - Marschner's model presented at SIGGRAPH 2003
- Simple approximate depth-sorting scheme
- Demo

Hair Rendering

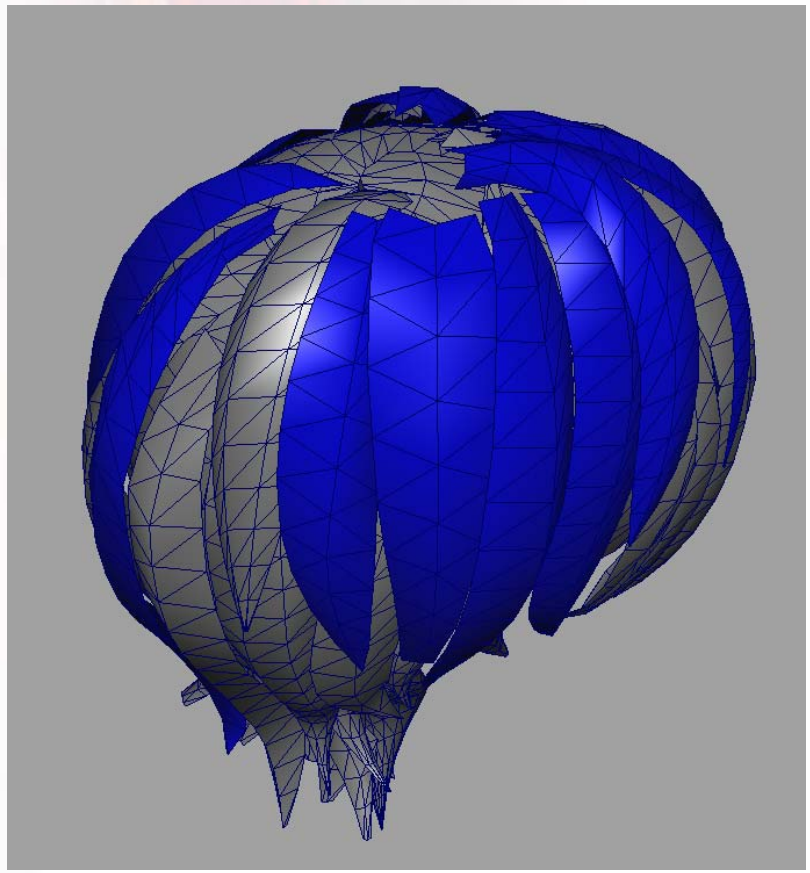
- Hair is important visually
  - Most humans have hair on their heads
- Hair is hard:
  - There is a lot of it
    - 100K-150K hair strands on a human head
  - Many different hair styles
  - ~25% of the total render time of “Final Fantasy - The Spirits Within” was spent on the main character’s hair

Why We Chose a Polygonal Hair Model

- Lower geometric complexity than line rendering
  - Makes depth sorting faster
- Integrates well into our art pipeline

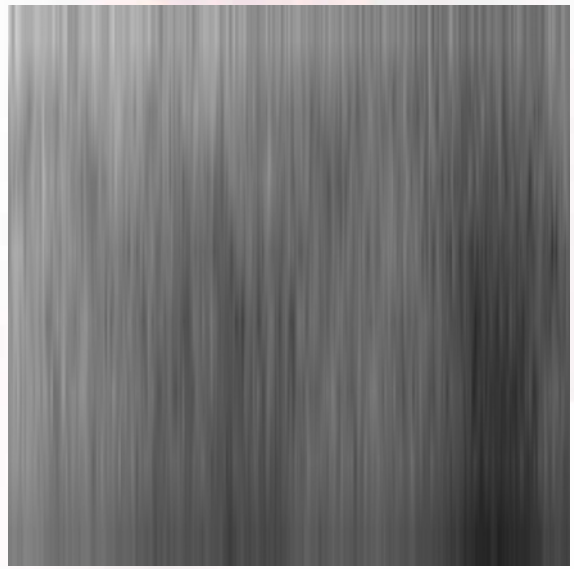
Hair Model Authoring

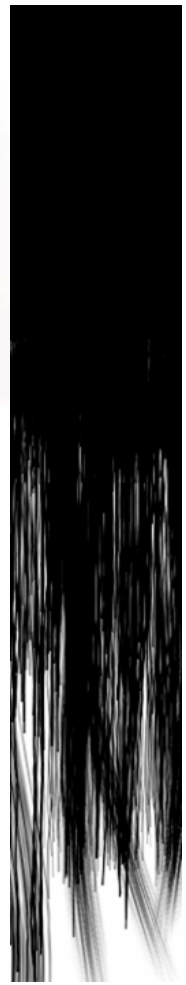
- Several layers of patches to approximate volumetric qualities of hair
- Ambient occlusion to approximate self-shadowing
  - Per vertex



Hair Model - Textures

- Base texture
  - Stretched noise
- Alpha texture
  - should have fully opaque regions
- Specular shift texture
- Specular noise texture

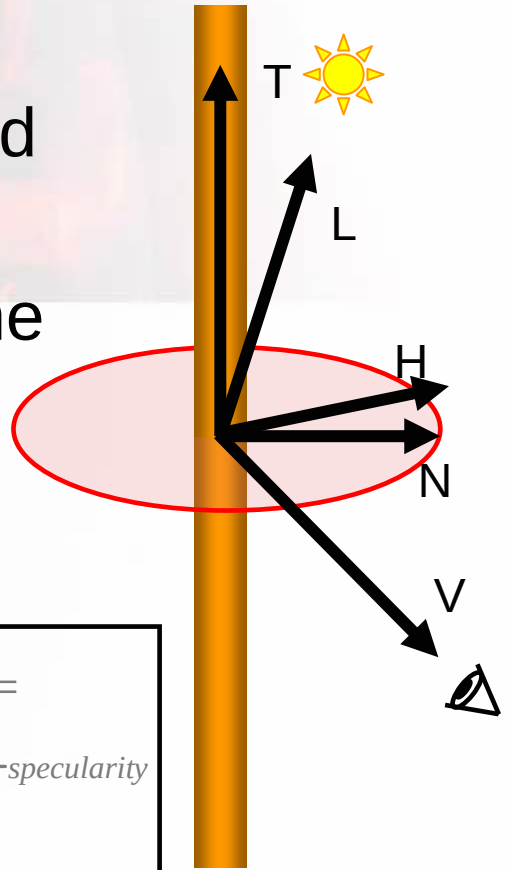




More on these later…

Hair Lighting: Kajiya-Kay Model

- Anisotropic strand lighting model
- Use hair strand tangent (T) instead of normal (N) in lighting equations
- Assumes hair normal to lie in plane spanned by T and view vector (V)
- Example: Specular N.H term

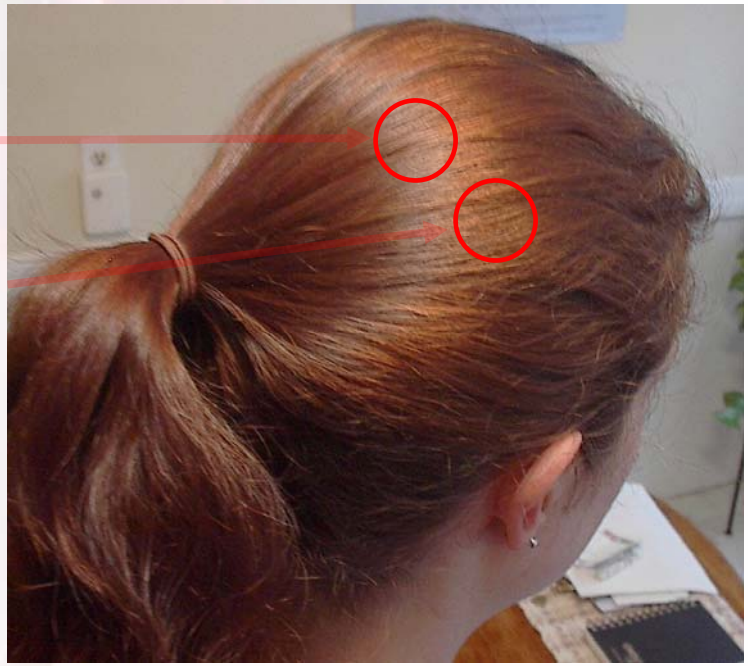


$$\cancel{\text{dot}(N,H)^{specularity}}$$

$$\sin(T,H)^{specularity} = \sqrt{1-\text{dot}(T,H)^2}^{specularity}$$

Hair Lighting: Marschner Model

- Based on measurements of hair scattering properties
- Observations
  - Primary specular highlight shifted towards hair tip
  - Secondary specular highlight
    - colored
    - shifted towards hair root
  - Sparkling appearance of secondary highlight
- Math is complex, we're just trying to match these observations pheno-menologically



Shader Breakdown

Vertex Shader

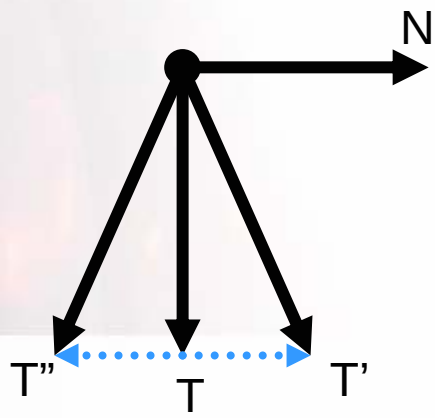
- Just passes down tangent, normal, view vector, light vector, ambient occlusion term

Pixel Shader

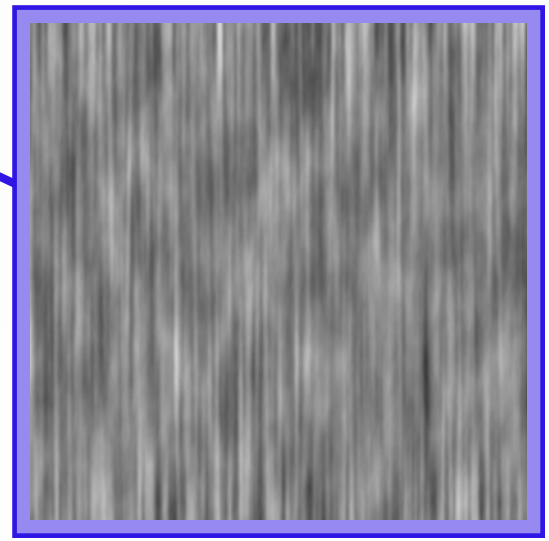
- Diffuse Lighting
  - Kajiya-Kay diffuse term $\sin(T, L)$ looks too bright without proper self-shadowing
  - We use a tweaked N.L term
- Two shifted specular highlights
- Combining terms

Shifting Specular Highlights

- To shift the specular highlight along the length of the hair, we nudge the tangent along the direction of the normal
- Assuming T is pointing from root to tip:
  - Positive nudge moves highlight towards root
  - Negative nudge moves highlight towards tip
- Look up shift value from texture to break up uniform look over hair patches

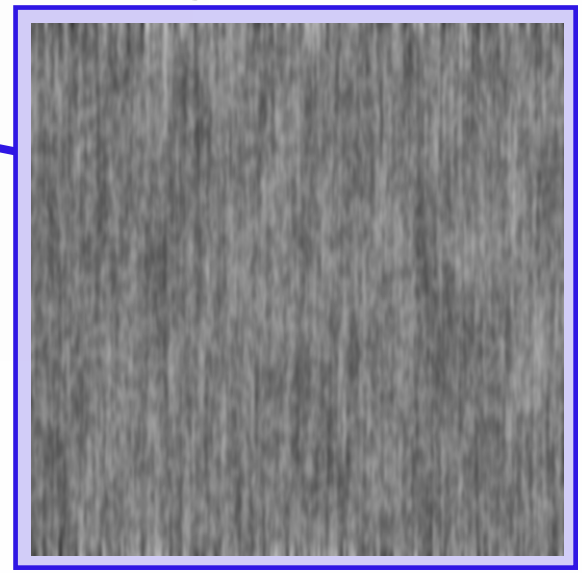



```
float3 ShiftTangent (float3 T, float3 N,
                     float shift)
{
   float3 shiftedT = T + shift * N;
   return normalize (shiftedT);
}
```



Specular Strand Lighting

- We do strand specular lighting using the half-angle vector
  - Using reflection vector and view vector would make the shader a little more complicated
- Two highlights with different colors, specular exponents and differently shifted tangents
- Modulate secondary highlight with noise texture



```
float StrandSpecular (float3 T, float3 V,
                      float3 L, float exponent)
{
   float3 H = normalize(L + V);
   float dotTH = dot(T, H);
   float sinTH = sqrt(1.0 - dotTH*dotTH);
   float dirAtten = smoothstep(-1.0, 0.0,
                               dot(T, H));
   return dirAtten * pow(sinTH, exponent);
}
```

Putting it All Together

(Note: external constants are light blue)

```
float4 HairLighting (float3 tangent, float3 normal, float3 lightVec,
                     float3 viewVec, float2 uv, float ambOcc)
{
  // shift tangents
  float shiftTex = tex2D (tSpecShift, uv) - 0.5;
  float3 t1 = ShiftTangent (tangent, normal, primaryShift + shiftTex);
  float3 t2 = ShiftTangent (tangent, normal, secondaryShift + shiftTex);

  // diffuse lighting: the lerp shifts the shadow boundary for a softer look
  float3 diffuse = saturate (lerp (0.25, 1.0, dot(normal, lightVec));
  diffuse *= diffuseColor;

  // specular lighting
  float3 specular = specularColor1 * StrandSpecular (t1, viewVec, lightVec,
                                                     specExp1);
  // add 2nd specular term, modulated with noise texture
  float specMask = tex2D (tSpecMask, uv); // approximate sparkles using texture
  specular += specularColor2 * specMask * StrandSpecular (t2, vieVec, lightVec,
                                                          specExp2);
  // final color assembly
  float4 o;
  o.rgb = (diffuse + specular) * tex2D (tBase, uv) * lightColor;
  o.rgb *= ambOcc;              // modulate color by ambient occlusion term
  o.a = tex2D (tAlpha, uv);     // read alpha texture
  return o;
}
```

Ambient Occlusion

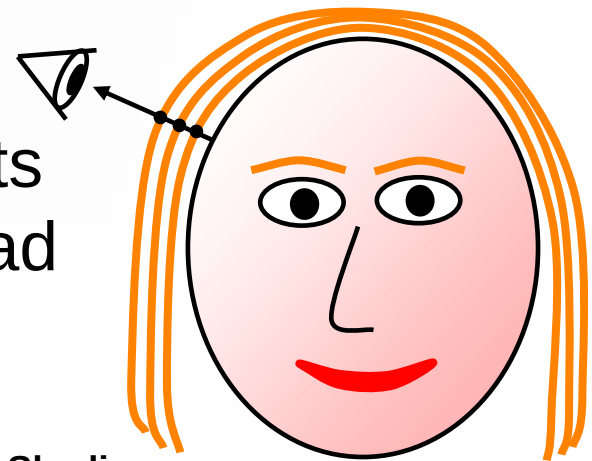
Diffuse Term

Specular Term

Combined

Approximate Depth Sorting

- Need to draw in back-to-front order for correct alpha-blending
- For a head with hair this is very similar to inside to outside
- Use static index buffer with inside to outside draw order, computed at preprocess time
  - Sort connected components (hair strand patches) instead of individual triangles



Sorted Hair Rendering Scheme

- Pass 1 – opaque parts
  - Enable alpha test to only pass opaque pixels
  - Disable backface culling
  - Enable Z writes, set Z test to `Less`
- Pass 2 – transparent back-facing parts
  - Enable alpha test to pass all non-opaque pixels
  - Cull front-facing polygons
  - Disable Z writes, set Z test to `Less`
- Pass 3 – transparent front-facing parts
  - Enable alpha test to pass all non-opaque pixels
  - Cull back-facing polygons
  - Enable Z writes, set Z test to `Less`

Performance Tuning

- Use early Z culling extensively to save us from running expensive pixel shader
- Usually half the hair is hidden behind the head
  - Draw head first
- Early Z culling can’t be used when alpha test is enabled!
  - Solution: Prime Z buffer with a very simple shader that uses alpha test
  - Use Z testing instead of alpha testing in subsequent passes for same effect
- Early Z culling saves considerable fill overhead!

Optimized Rendering Scheme

- Pass 1 – prime Z buffer
  - Enable alpha test to only pass opaque pixels
  - Disable backface culling
  - Enable Z writes, set Z test to `Less`
  - *Disable color buffer writes*
  - *Use simple pixel shader that only returns alpha*
- Pass 2 – opaque parts
  - Disable backface culling
  - Disable Z writes, set Z test to `Equal`
- Pass 3 – transparent back-facing parts
  - Cull front-facing polygons
  - Disable Z writes, set Z test to `Less`
- Pass 4 – transparent front-facing parts
  - Cull back-facing polygons
  - Enable Z writes, set Z test to `Less`

Demo

Pros and Cons

Pros:

- Low geometric complexity
  - Lessens load on vertex engine
  - Makes depth sorting faster
- Usable on lower-end hardware with simpler shaders or fixed-function pipeline

Cons:

- Sorting scheme assumes little animation in hair model
  - Things like dangling pony tails need to be handled separately
  - Sort geometry at run-time to overcome this
- Not suitable for all hair styles

Conclusion

- Polygonal hair model
- Hair lighting
- Simple approximate depth-sorting scheme
- Optimization Tips

References

- J. Kajiya and T. Kay. *Rendering fur with three dimensional textures*. In SIGGRAPH 89 Conference Proceedings, pp. 271-280, 1989.
- Stephen R. Marschner, Henrik Wann Jensen, Mike Cammarano, Steve Worley, and Pat Hanrahan, *Light Scattering from Human Hair Fibers.* In Proceedings of *SIGGRAPH 2003*.
- SIGGRAPH 2003 Hair Rendering Course Notes