Life on the Bleeding Edge: More Secrets of the NVIDIA Demo Team

Eugene d’Eon

NVISION 08
THE WORLD OF VISUAL COMPUTING

NVIDIA

Run the Demo

Concept Artwork

Concept Artwork

Concept Artwork

Concept Artwork

Concept Artwork

Skin Shading

Skin Shading

- Same shader as the Human Head demo
- Doug Jones’s assets transformed into the warrior
- New capture for the medusa face

Reusing Assets

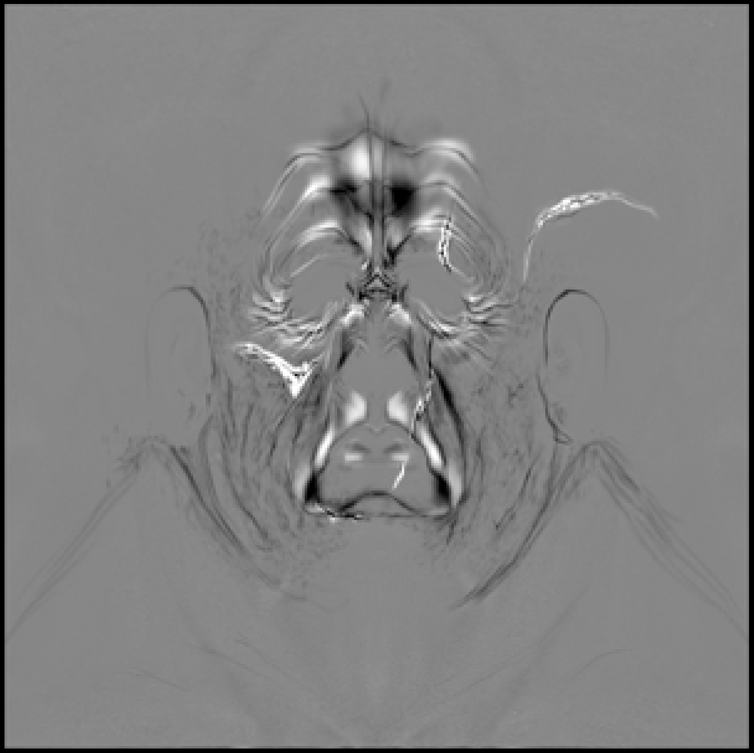
- Much of the realism of the Human Head demo comes from the high resolution face mesh
- We were successful at reusing the fine detail for a different face

Reusing Assets: Color

Human Head demo
color map

Warrior
color map

Reusing Assets: Normal



Scar displacement from Zbrush adds to original normals from the Human Head demo

Reusing Assets: Result

Reusing Assets: Result

Acquiring New Assets

- Female Medusa face:
  - New die cast and scan
  - New color map acquisition
  - Also captured lower resolution morph targets

Acquiring New Assets

Acquiring New Assets

Kenneth Wiatrak's studio

- Five high resolution synchronized cameras
- Also captures low resolution geometry

Kenneth Wiatrak’s studio

Kenneth Wiatrak’s studio

- Five high resolution photos
- Projected onto high resolution mesh by XYZRGB
- Combine to produce final high resolution color map

Medusa's Face

Medusa's Face

Medusa’s Face

Kenneth Wiatrak's studio

Kenneth Wiatrak's studio

- Became a reference for the creation of the 65 facial blend shapes
- Weren't able to use the color maps

Skin shader

Start →

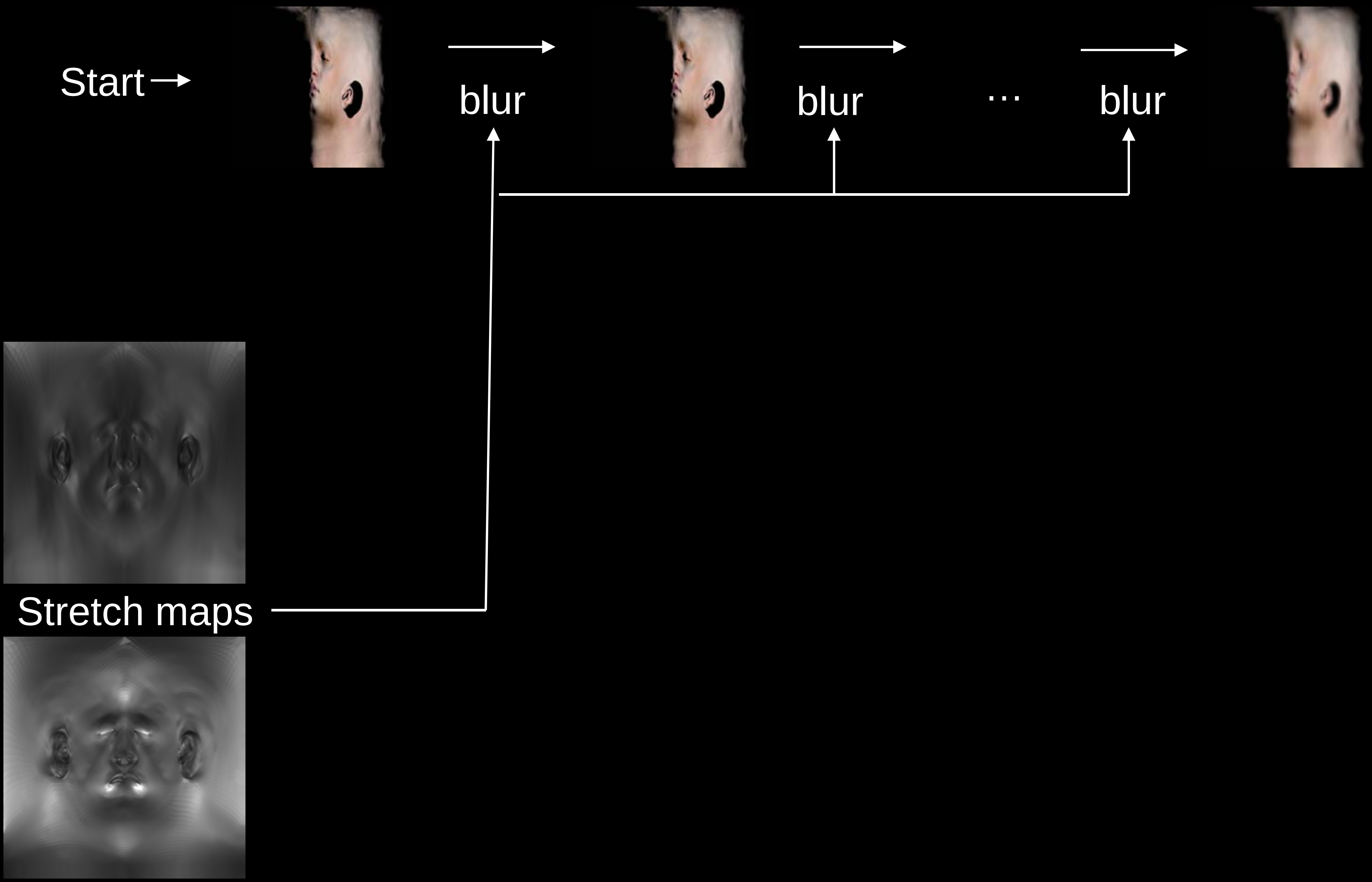
Skin shader

Start →

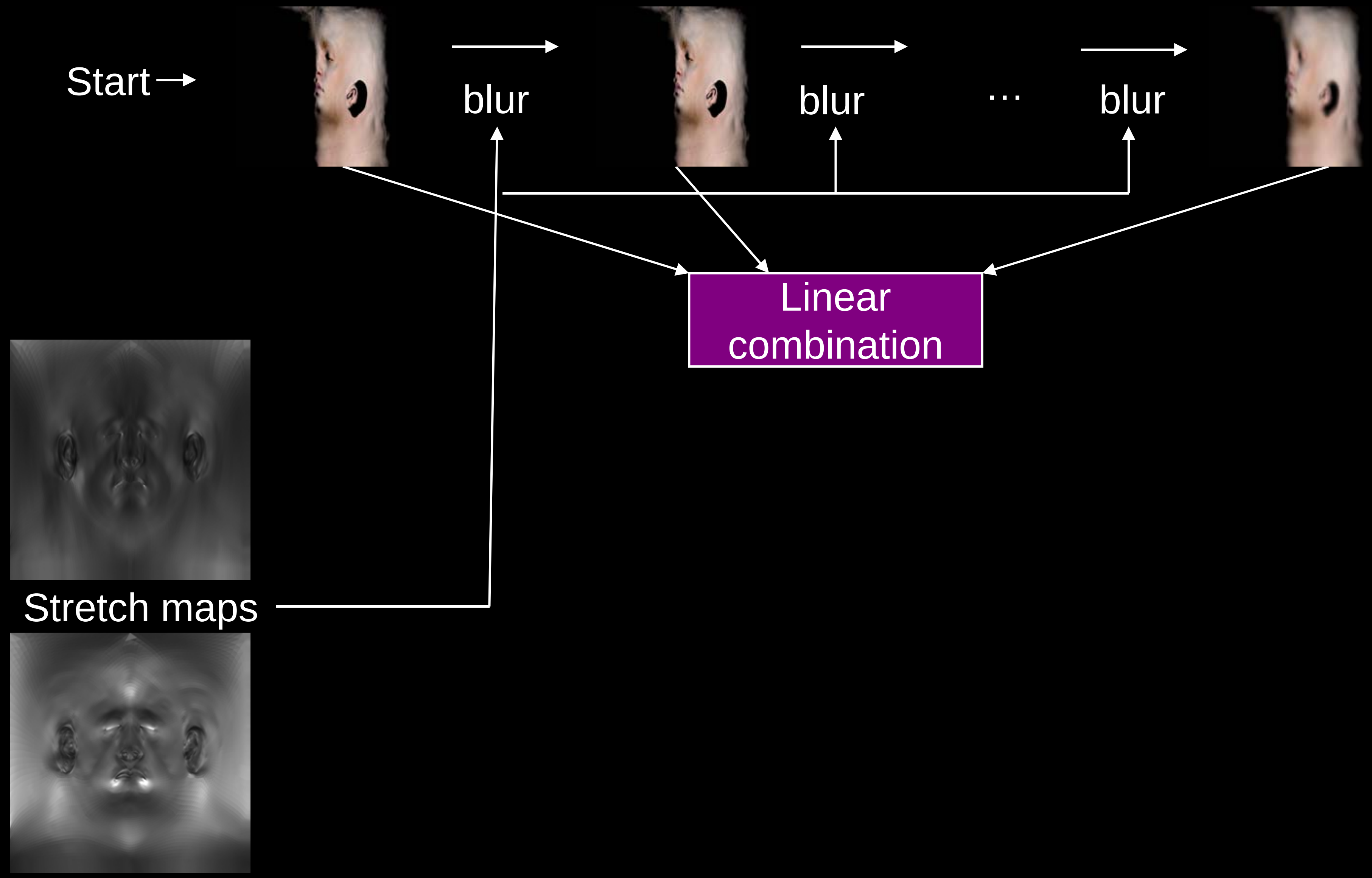
blur

Skin shader

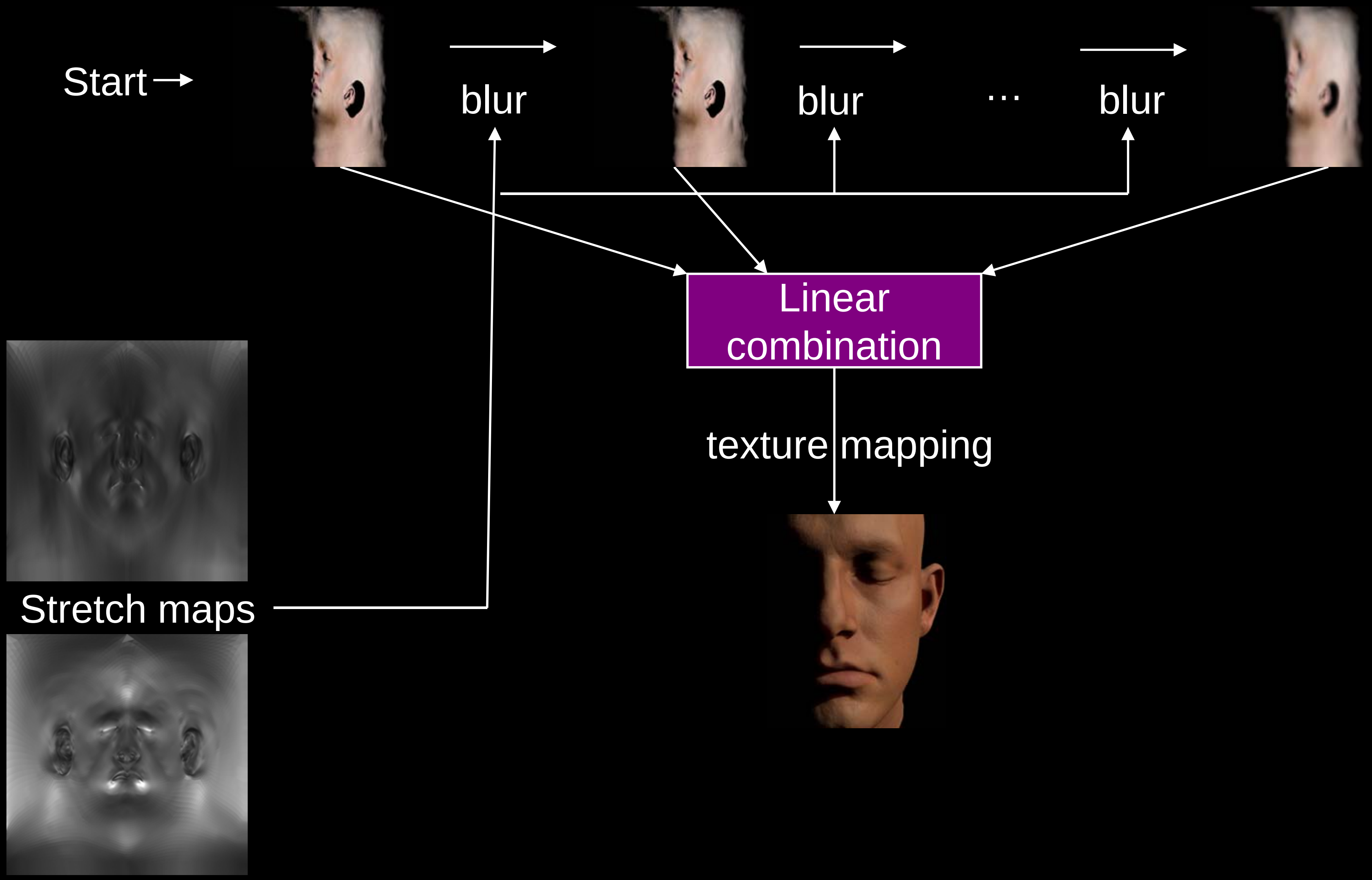
Skin shader



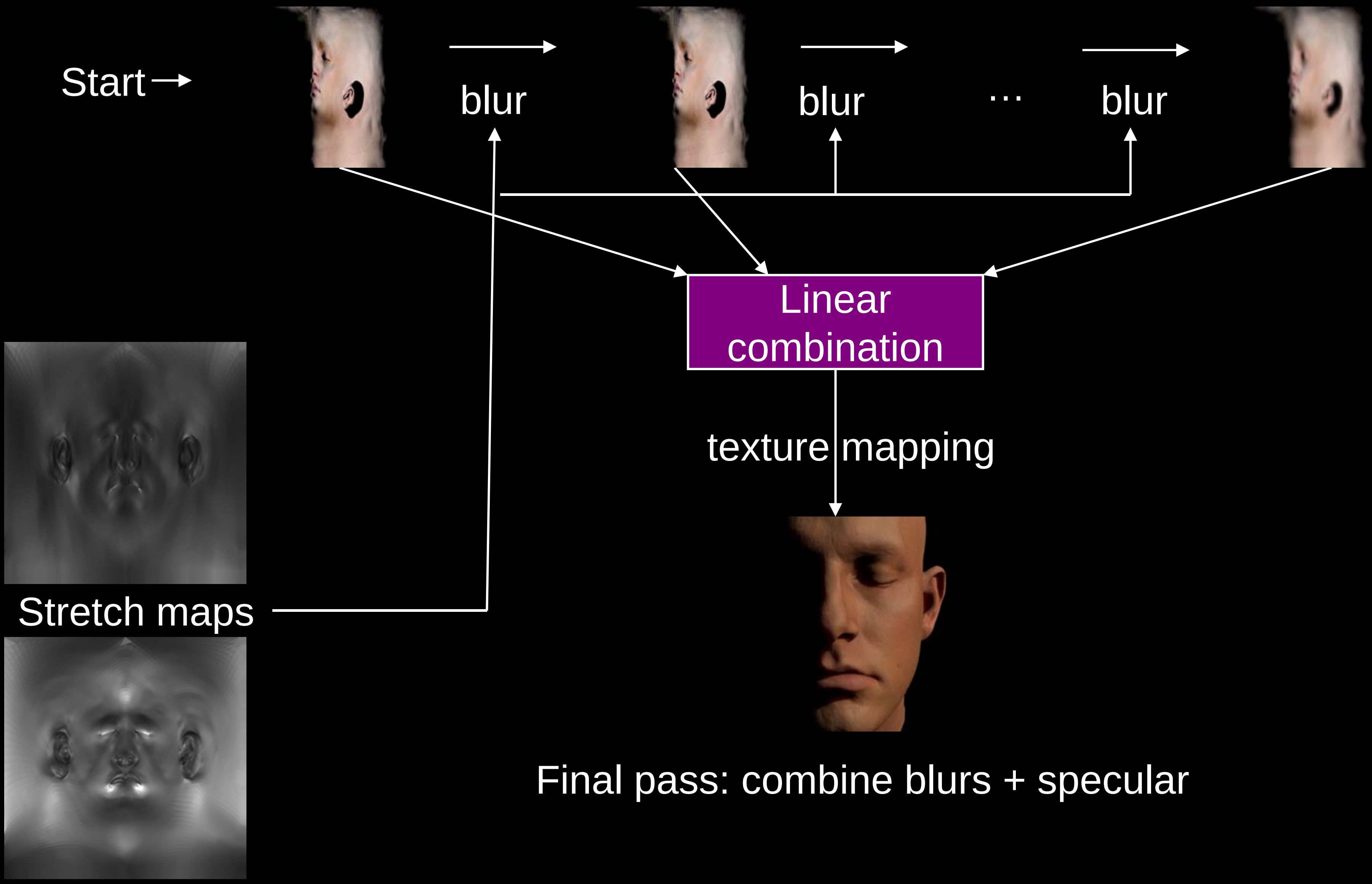
Skin shader



Skin shader


Start
blur
blur
...
blur
Linear combination
texture mapping
Stretch maps

Skin shader


Start
blur
blur
…
blur
Linear combination
texture mapping
Stretch maps
Final pass: combine blurs + specular

Skin Shader Optimizations

- Reuse blur targets for multiple skin surfaces

Skin Shader Optimizations

- Reuse blur targets for multiple skin surfaces
- skinDetailScale

Skin Shader Optimizations

- Reuse blur targets for multiple skin surfaces
- skinDetailScale
  - Texture space lighting at 1024x1024 for close up shots

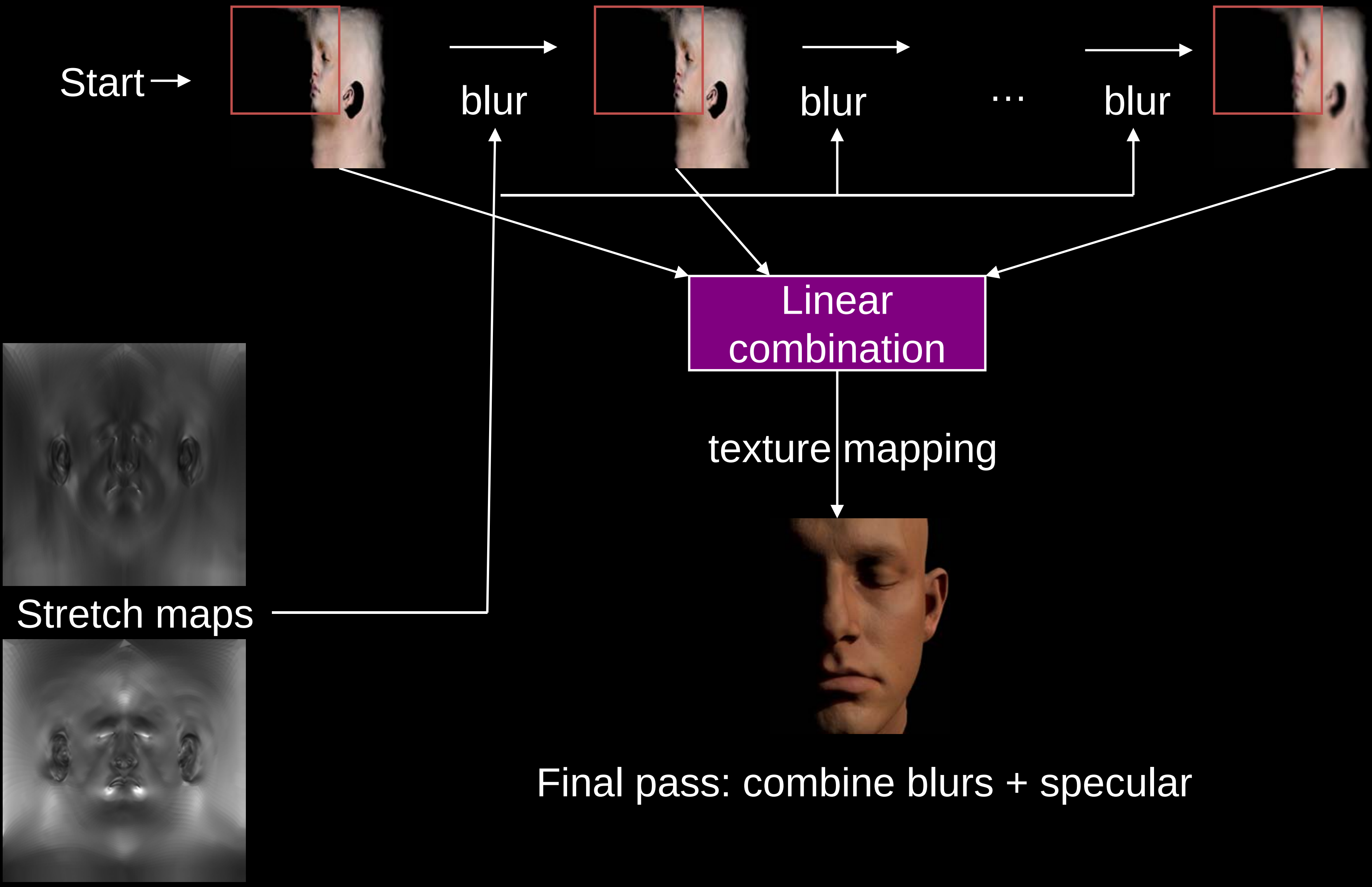
Skin Shader Optimizations

- Reuse blur targets for multiple skin surfaces
- skinDetailScale
  - Texture space lighting at 1024x1024 for close up shots
  - Reduce viewport in renderpasses to be % of total for

Skin Shader Optimizations

- Reuse blur targets for multiple skin surfaces
- skinDetailScale
  - Texture space lighting at 1024x1024 for close up shots
  - Reduce viewport in renderpasses to be % of total for
    - Texture space lighting pass
    - Each iterative blur pass
    - Facial shadow pass

Skin shader



Start
blur
blur
…
blur
Linear combination
texture mapping
Stretch maps
Final pass: combine blurs + specular

Skin Shader Optimizations

- skinDetailScale determined
  - Per shot (for Medusa)
  - Per frame based on bounded sphere projection on screen
- Significant performance gain

Screen-space bump mapping

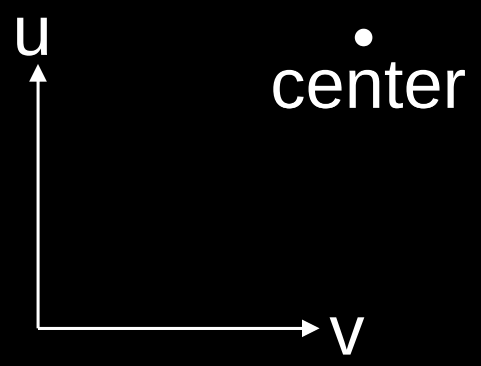
- Alternate form of bump mapping
- Store only single-channel displacement value over surface
- Each fragment being rendered accesses the displacement map 3 times
- Decide where to look in the displacement map based on the current view of the object

Screen-space bump mapping

- Use ddx and ddy of the texture coordinates to consider the displacement:
  - One pixel to the right
  - One pixel up
- Helps reduce aliasing
  - Wide spacing in displacement map:
    - Appropriate mip level of the displacement map

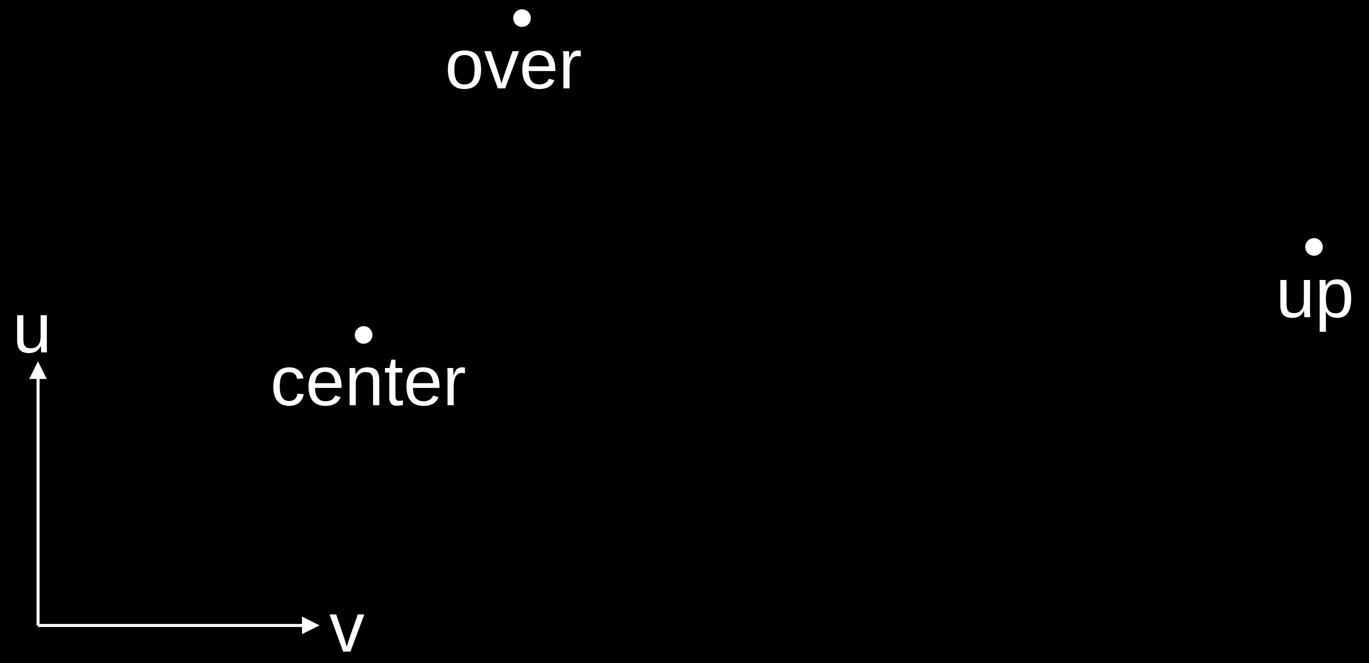
Screen-space bump mapping

- How do we compute the new normal?



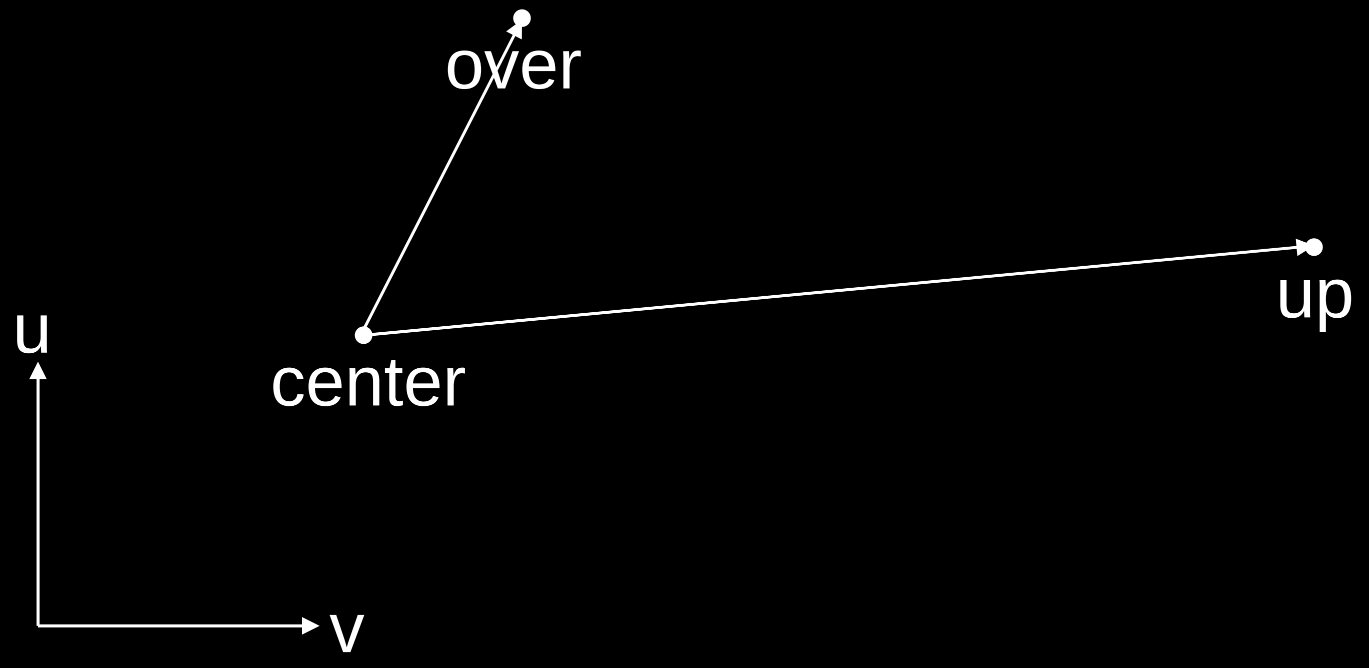
Screen-space bump mapping

- How do we compute the new normal?



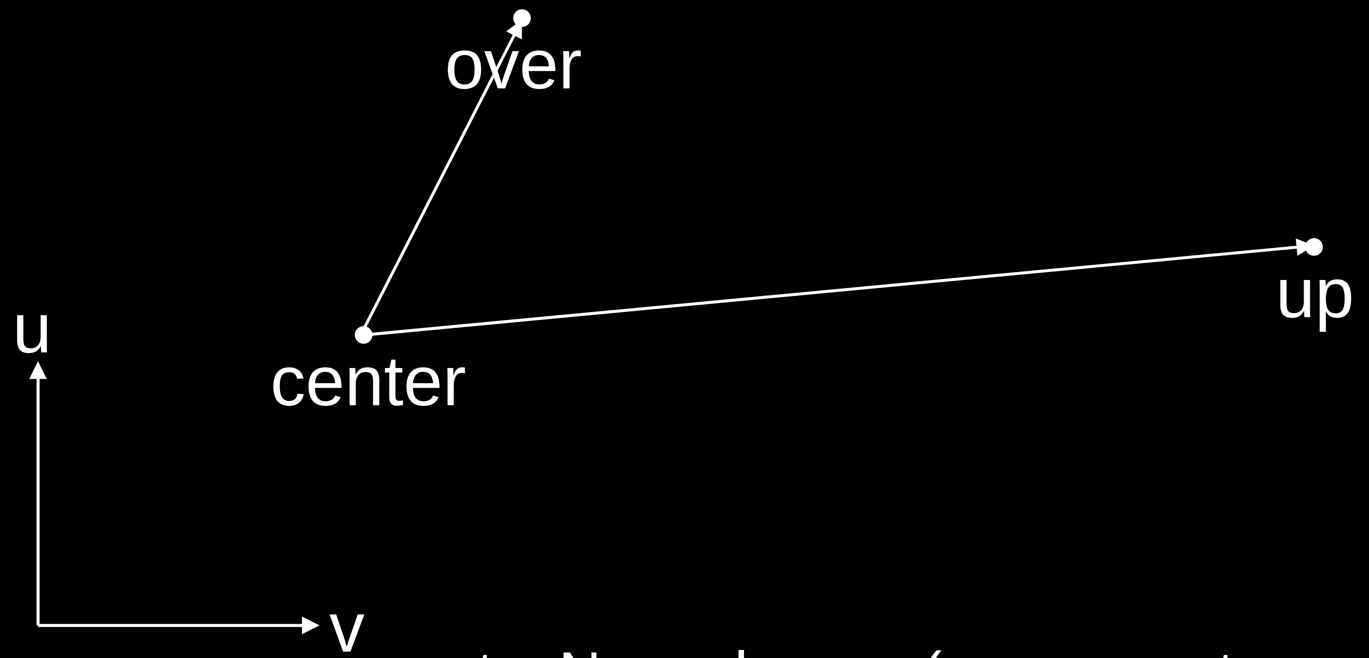
Screen-space bump mapping

- How do we compute the new normal?



Screen-space bump mapping

- How do we compute the new normal?



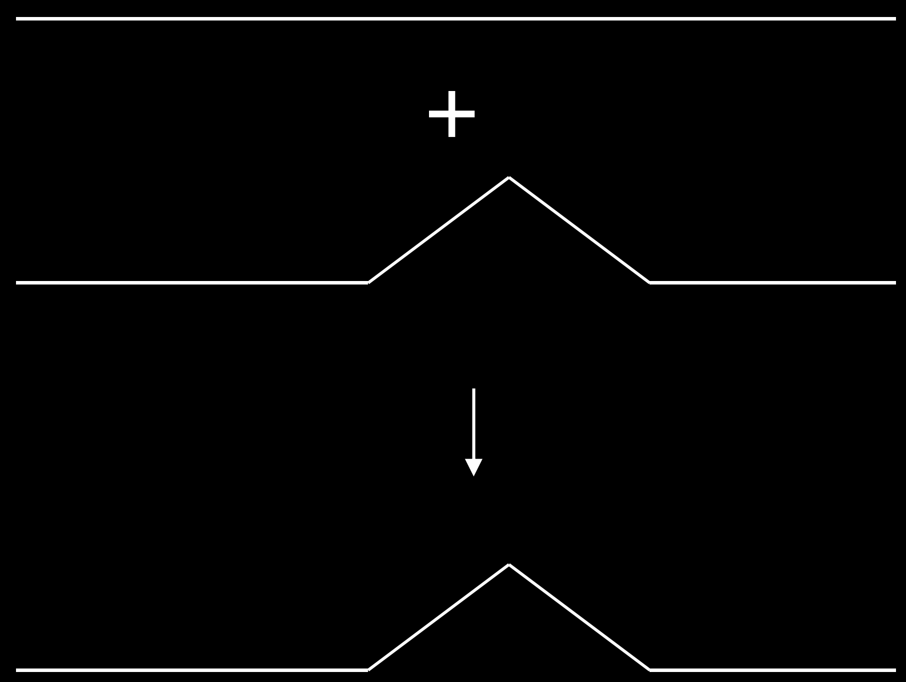
tanNormal=cross(over - center, up – center)

Screen-space bump mapping

- Advantages
  - 1/3 storage of a texture-space normal map
  - Normal map plugin can blur the detail of the displacement map
  - MIP mapping displacement makes sense
  - MIP mapping normals doesn't
  - Anisotropic filtering
  - Reduced aliasing

Screen-space bump mapping

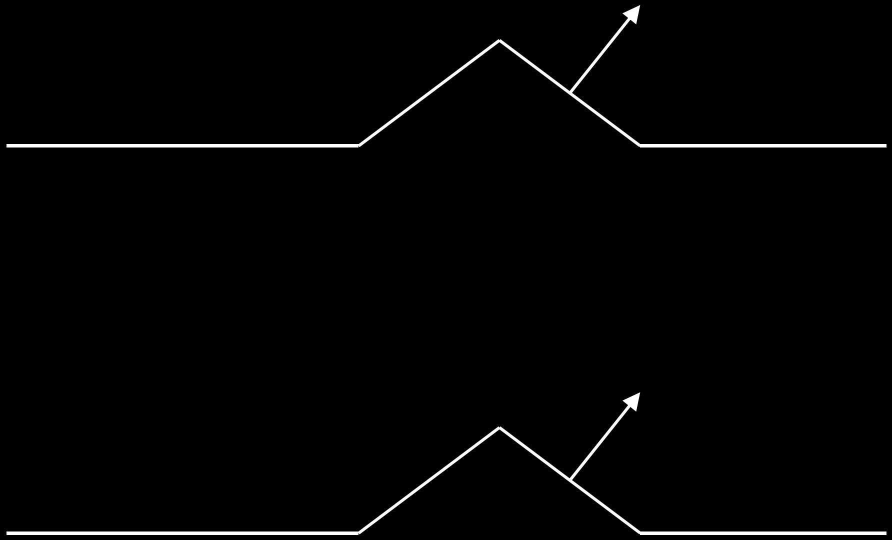
- Allows for multiple displacement maps to combine



Screen-space bump mapping

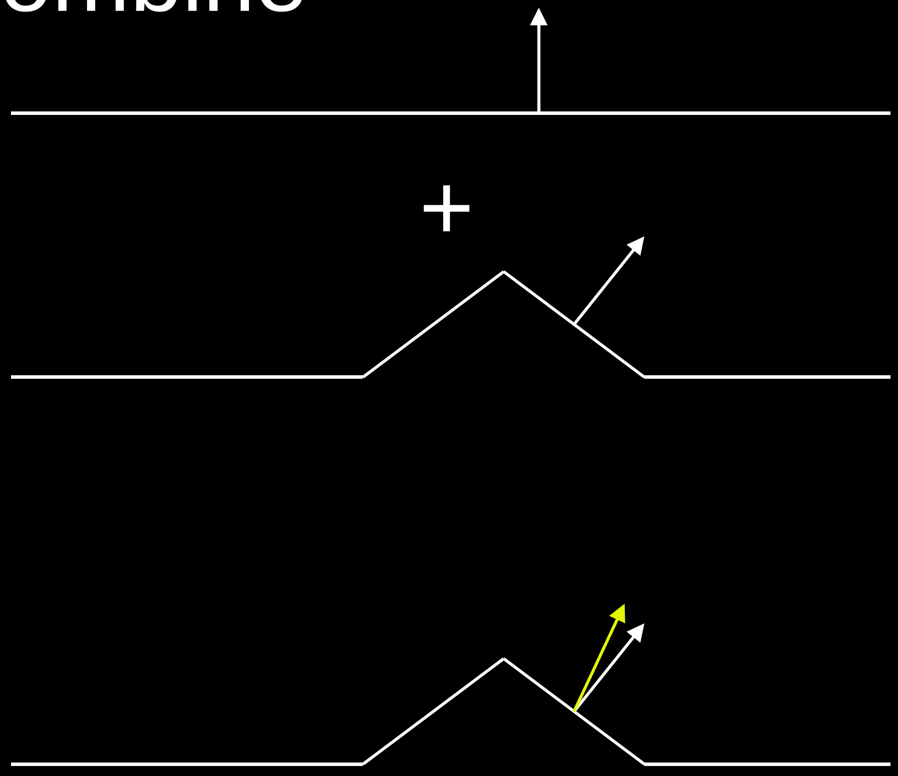
- Allows for multiple displacement maps to combine

+



Screen-space bump mapping

- Allows for multiple displacement maps to combine



Screen-space bump mapping

```
float  normalTap      = dispTex.Sample(TrilinearClamp, g2f.tex.xy);
float2 uv_offset_over = ddx( g2f.tex );
float2 uv_offset_up   = ddy( g2f.tex );
float  normalTapOver  = dispTex.Sample(TrilinearClamp, g2f.tex.xy +
   uv_offset_over);
float  normalTapUp    = dispTex.Sample(TrilinearClamp, g2f.tex.xy +
   uv_offset_up);

float displacementCenter = displacementHeight * normalTap;
float displacementOver   = displacementHeight * normalTapOver;
float displacementUp     = displacementHeight * normalTapUp;

float3 tanNormalBump = normalize( -cross( float3( uv_offset_over.x,
   uv_offset_over.y, ( displacementOver - displacementCenter ) ),
                                 float3( uv_offset_up.x, uv_offset_up.y, (
   displacementUp - displacementCenter ) ) ) );

float3 Nbump = normalize( T * tanNormalBump.x + B * tanNormalBump.y + N *
   tanNormalBump.z );
```

Screen-space bump mapping

Rainbow Boas

Artist requested this for medusa scales…

Cause?

- Two conflicting explanations on the web
  - Thin-film interference amongst many scale layers
  - Natural diffraction gratings grow on the scale surface

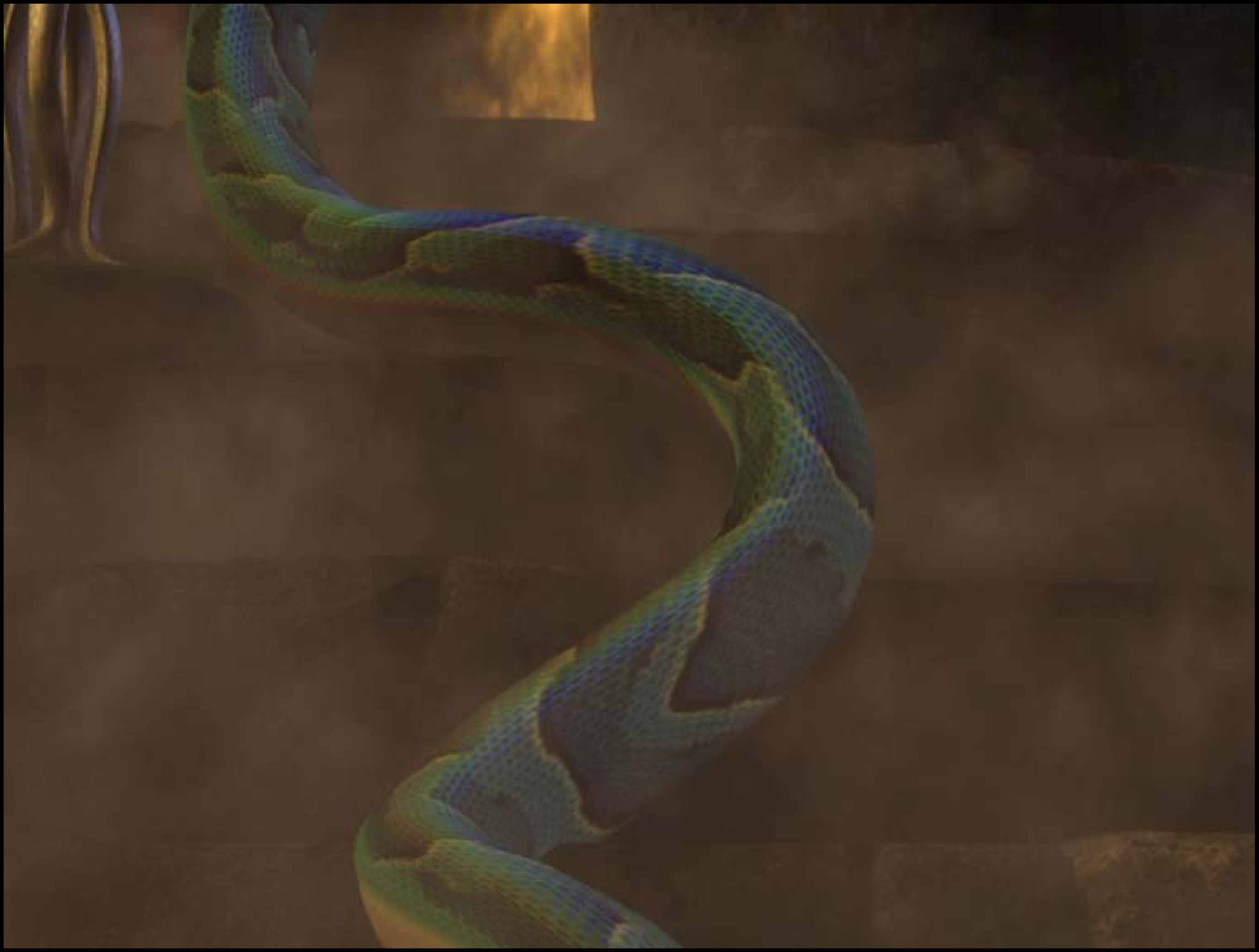
Took a Guess: Diffraction

- We tried diffraction and it looked pretty good
- Solution: GPU Gems 1: Jos Stam's diffraction chapter
- Simple fragment shader computes a colored diffraction term

Rainbow boa scale shader

Diffraction term

Rainbow boa scale shader



Diffraction term + diffuse term

Rainbow boa scale shader

Diffraction term + diffuse term + specular term

Crystals

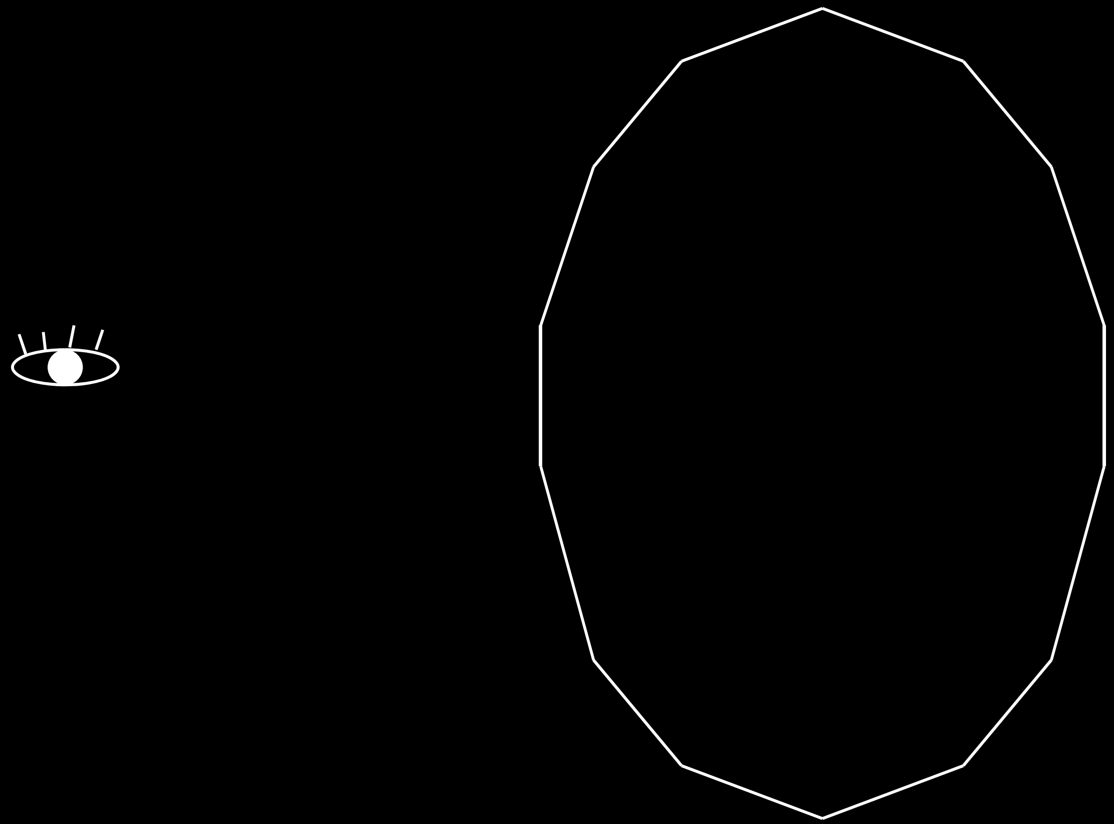
Crystal Appearance

- Combination of refracted and reflected light
- Reflected light is easy
- Refracted light: not so easy

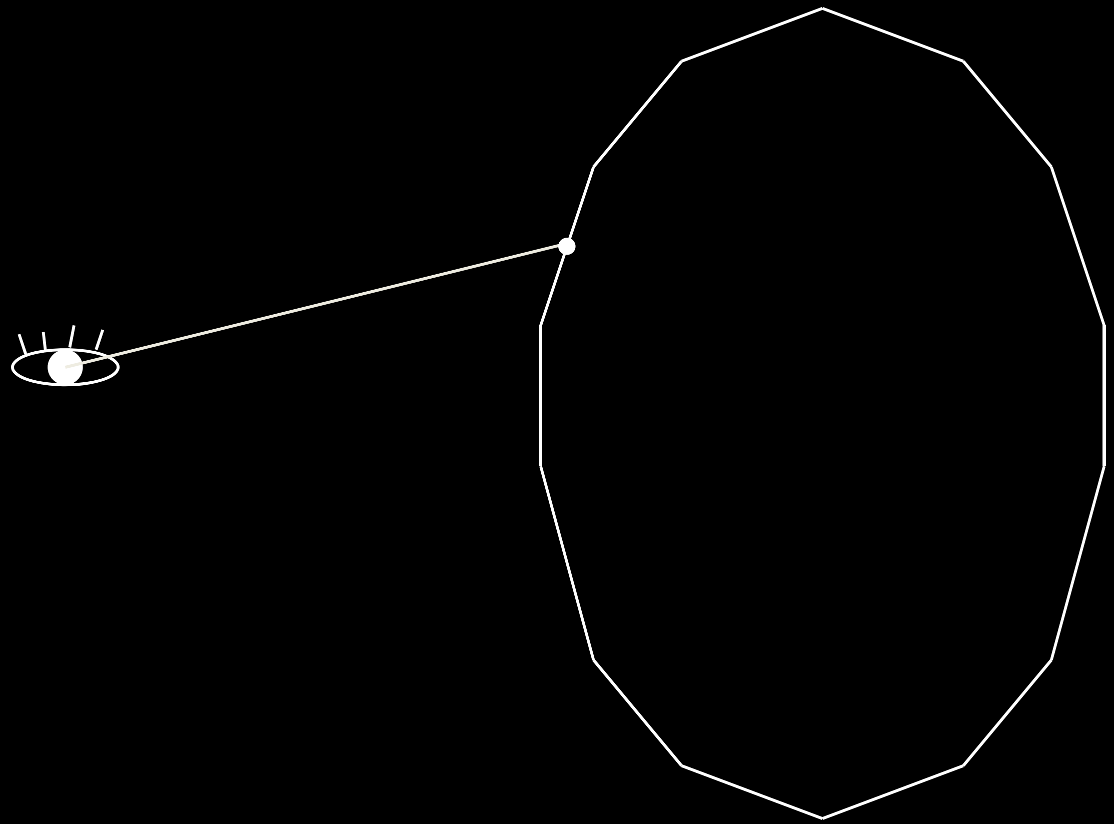
Previous work

- SIGGRAPH 2004
  - "Graphics Gems Revisited – Fast and Physically-Based Rendering of Gemstones"
    - Stephane Guy, Cyril Soler
- Very accurate images
- GPU implementation
- Fairly complex method
- If possible: find something cheaper

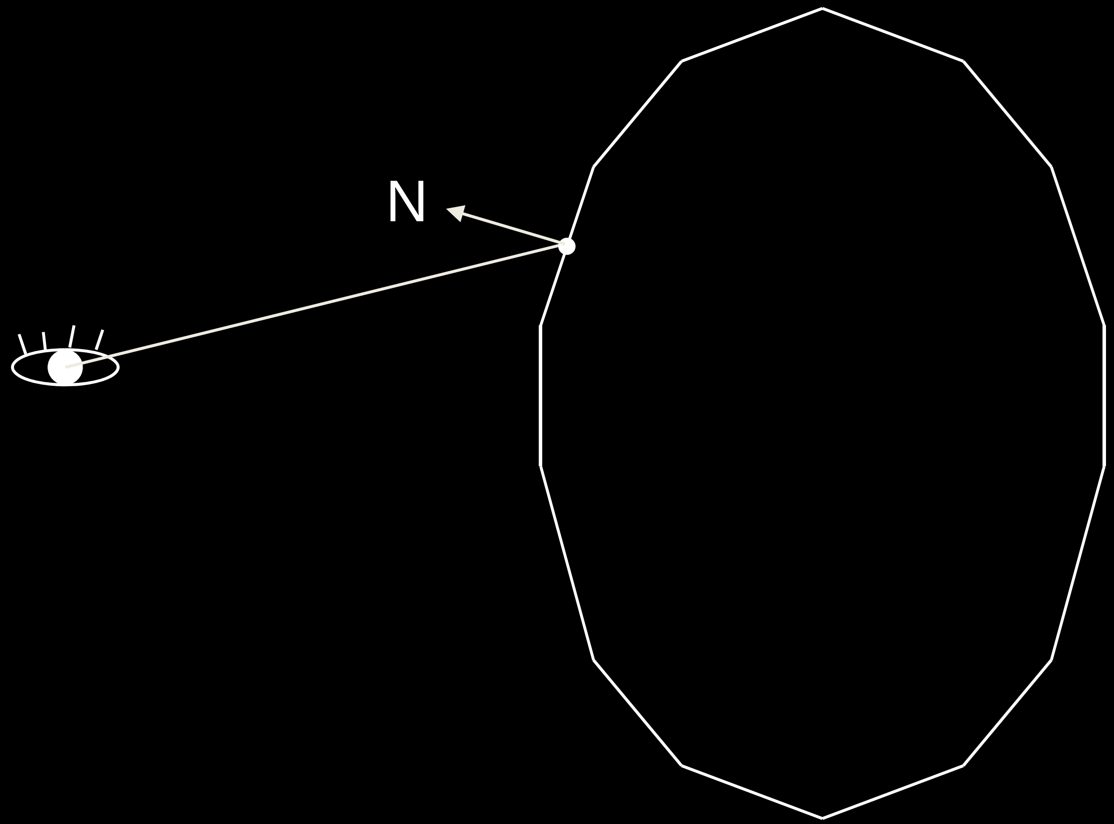
Crystal Reflectance



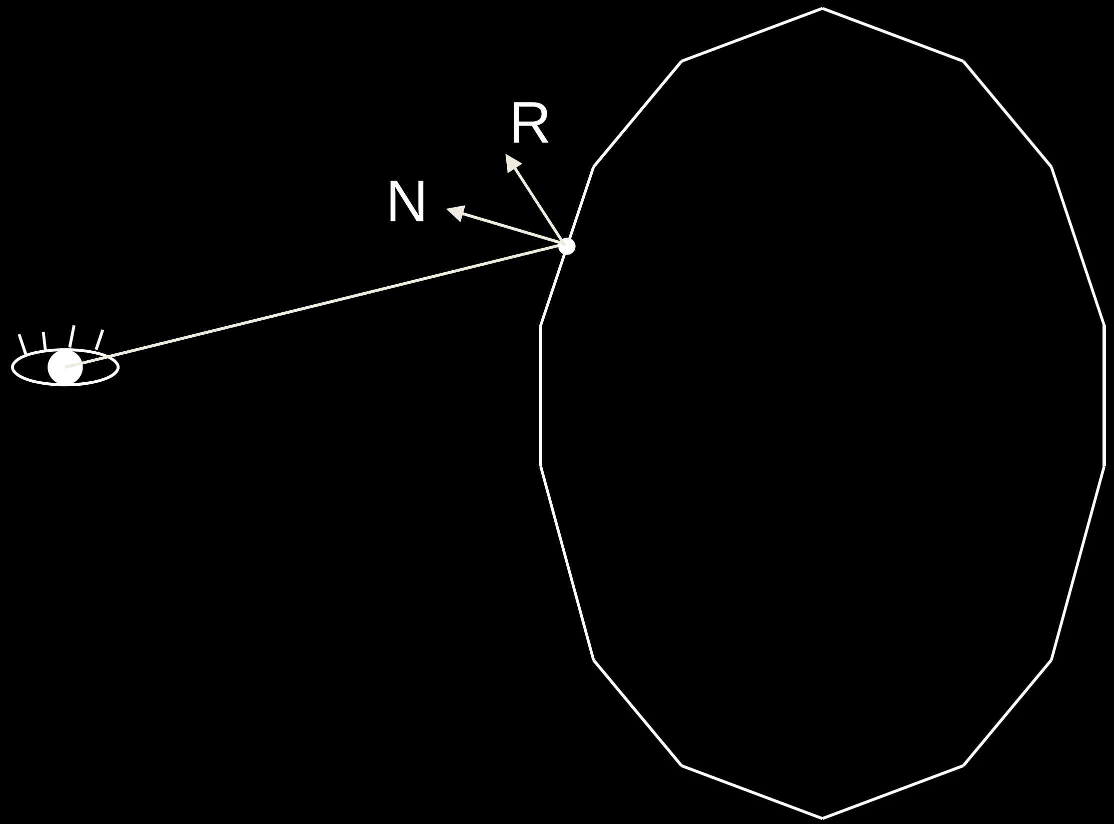
Crystal Reflectance



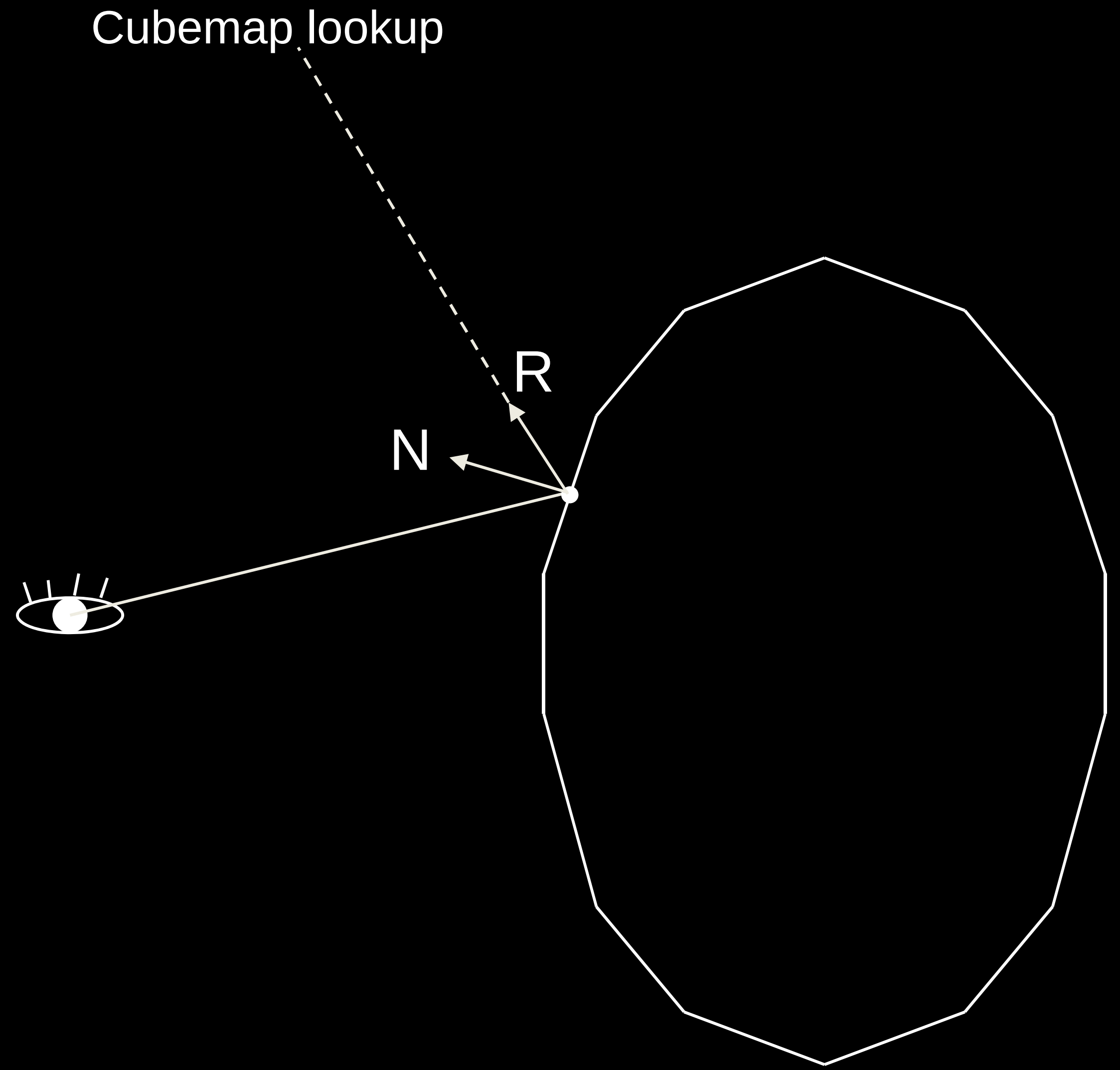
Crystal Reflectance



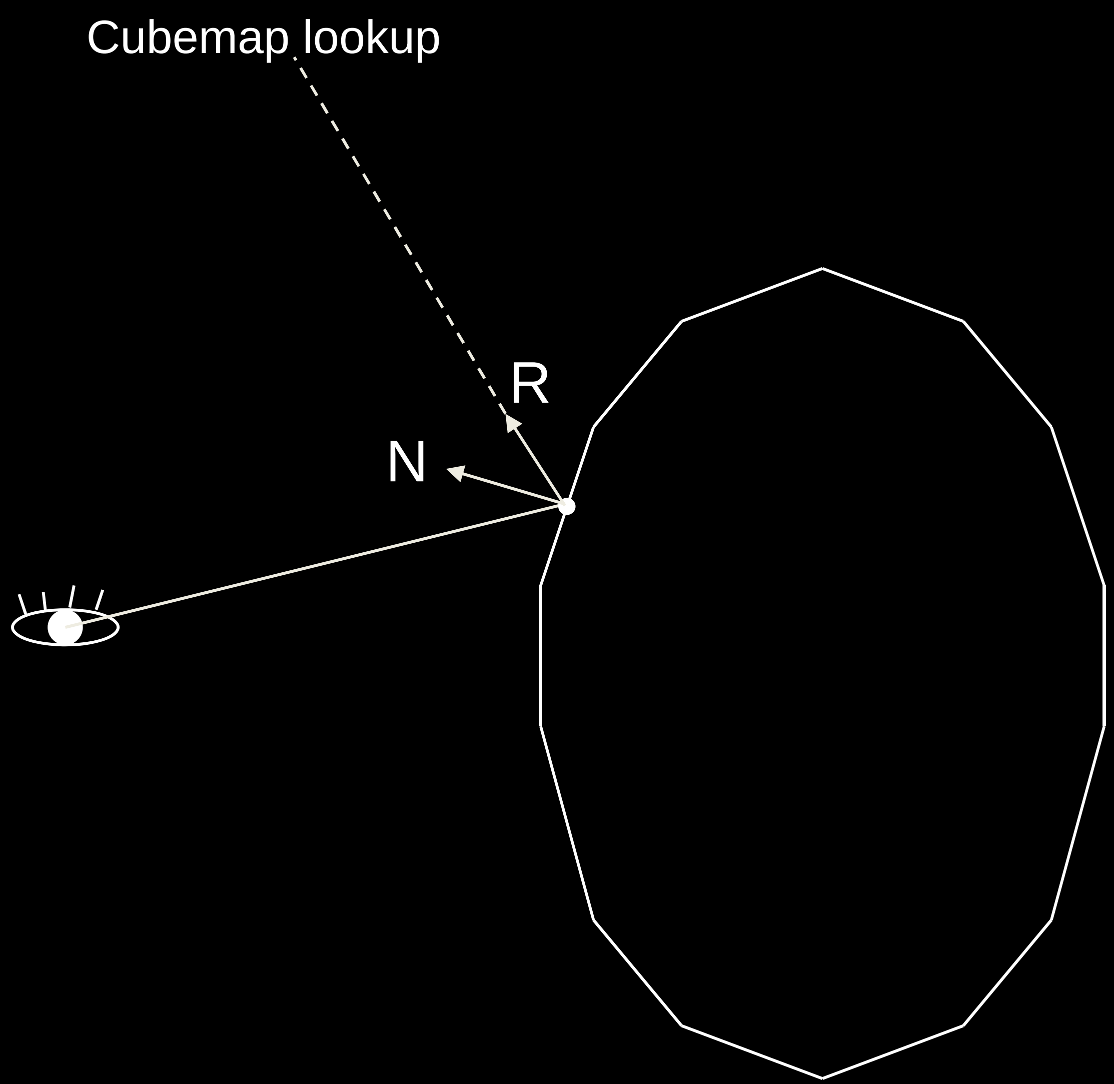
Crystal Reflectance



Crystal Reflectance

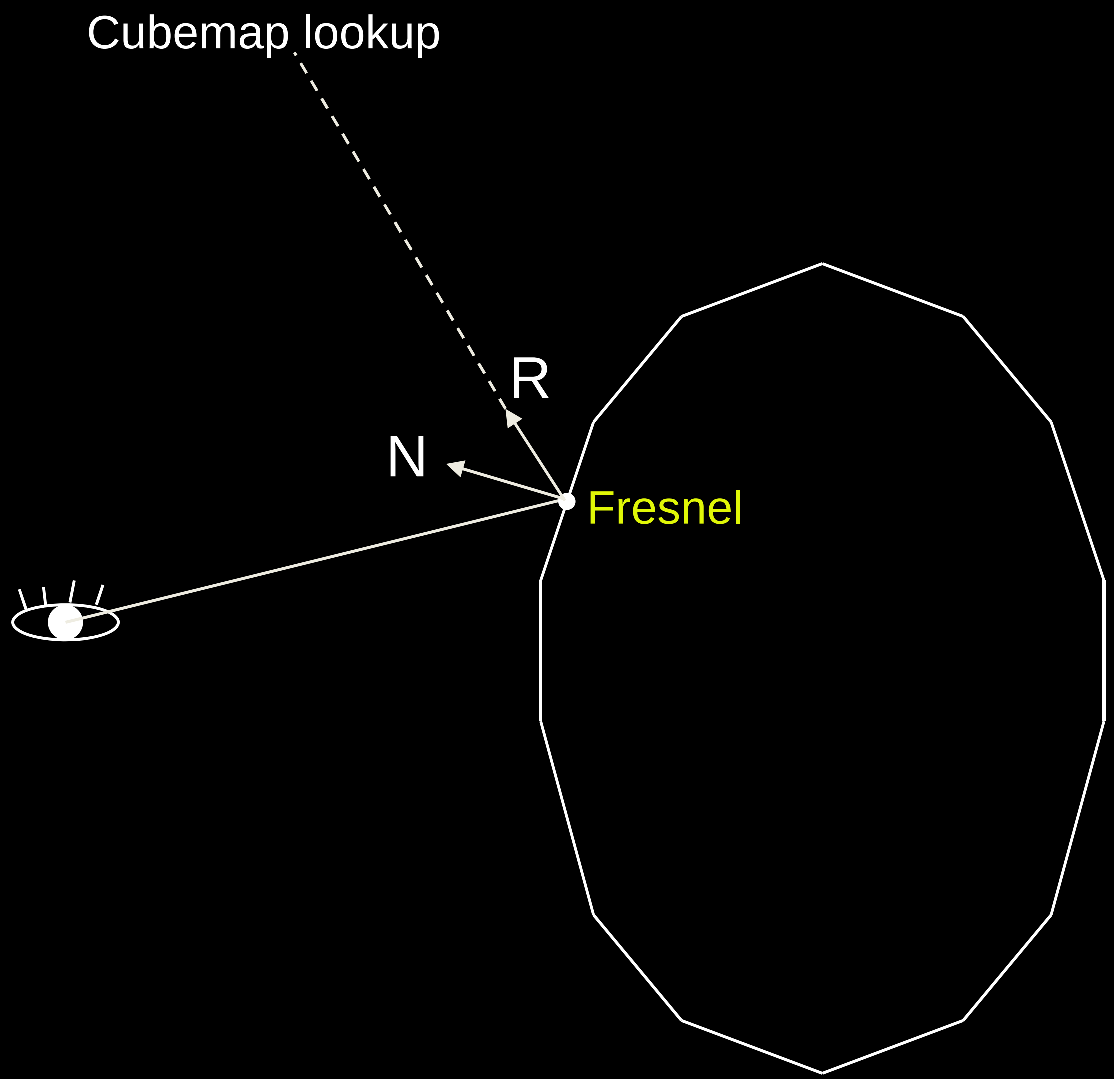


Crystal Reflectance

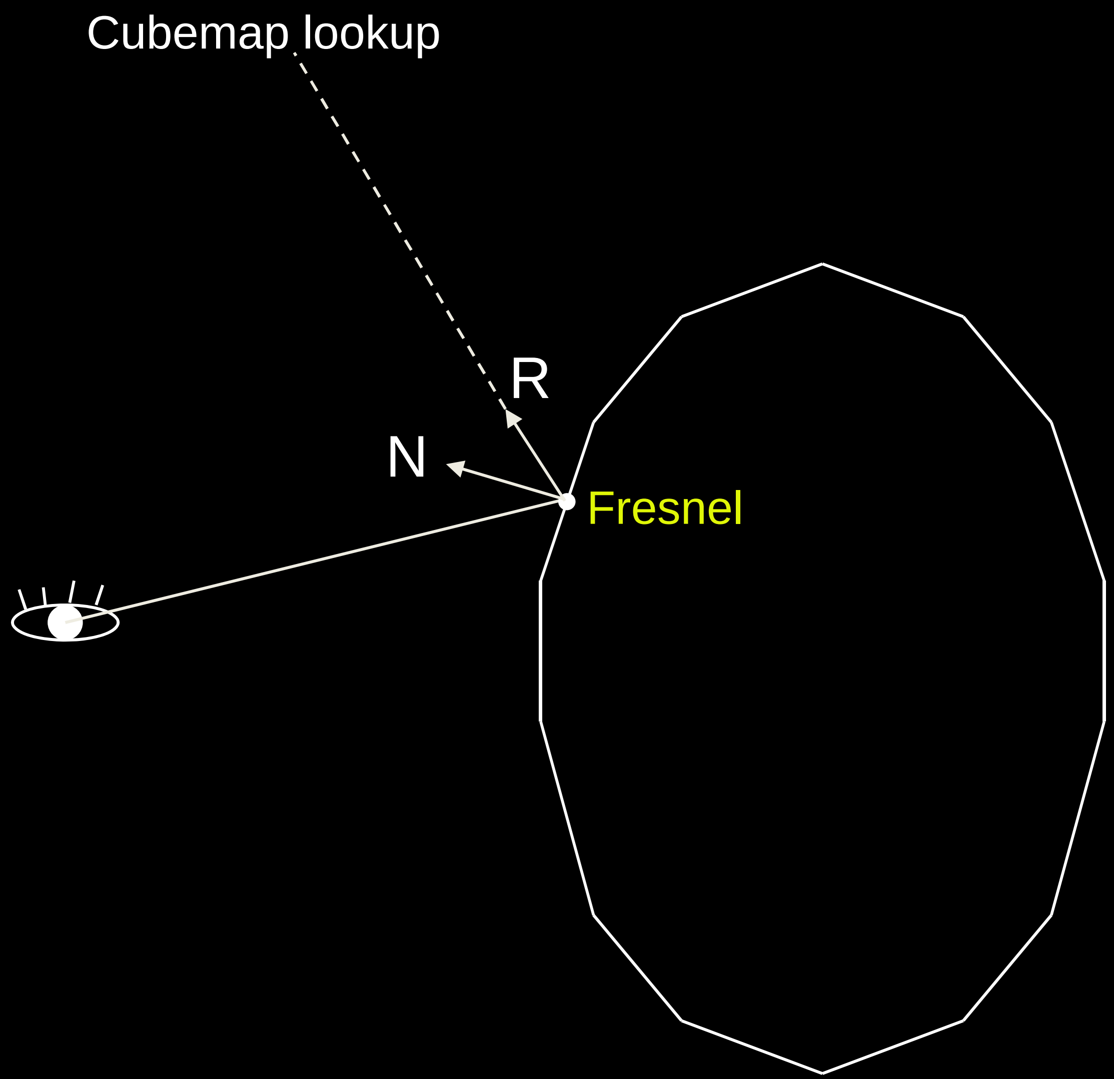


Reflectance only

Crystal Reflectance



Crystal Reflectance



Fresnel attenuated reflectance

Fresnel terms

```
// reflectance at normal incidence:
float F0 = pow( (1.0 - index)/(1.0 + index), 2.0 );
```

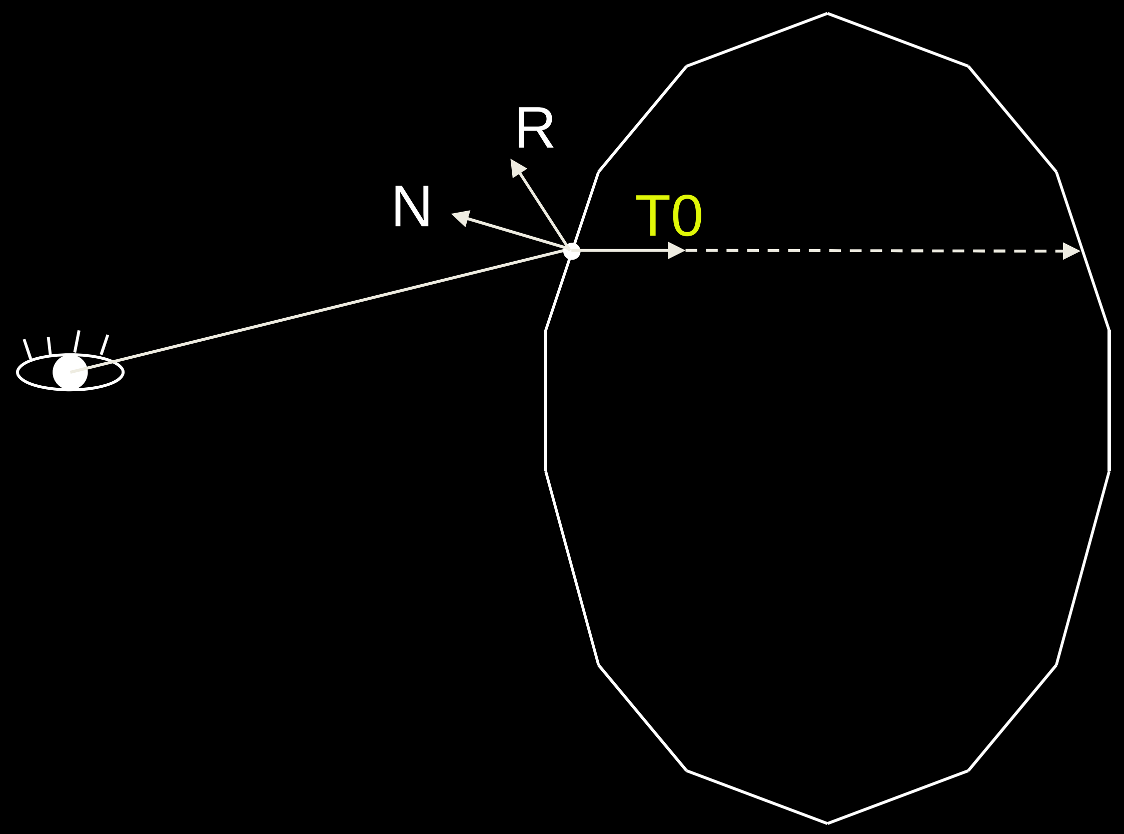
Fresnel terms

```
// reflectance at normal incidence:
float F0 = pow( (1.0 - index)/(1.0 + index), 2.0 );

// Schlick's approximation
float fresnelReflectance( float3 N, float3 V, float F0 )
{
  float base = 1.0 - dot( N, V );
  float exponential = pow( base, 5.0 );
  return exponential + F0 * ( 1.0 - exponential );
}

// note:
// fresnelTransmittance = 1.0 - fresnelReflectance
```

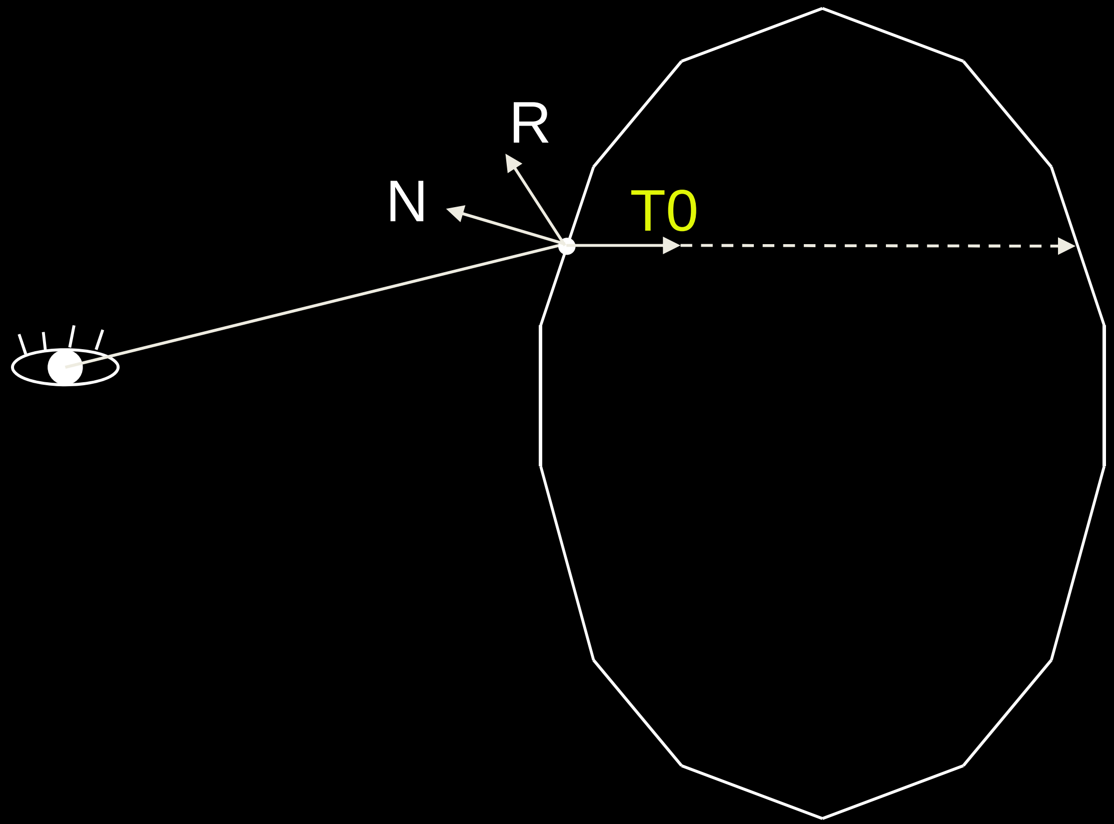
Fresnel terms

```
// NOTE: for crystal->air interactions use
float F0_out = pow( (index - 1.0)/(1.0 + index), 2.0 );
```

Crystal Transmittance

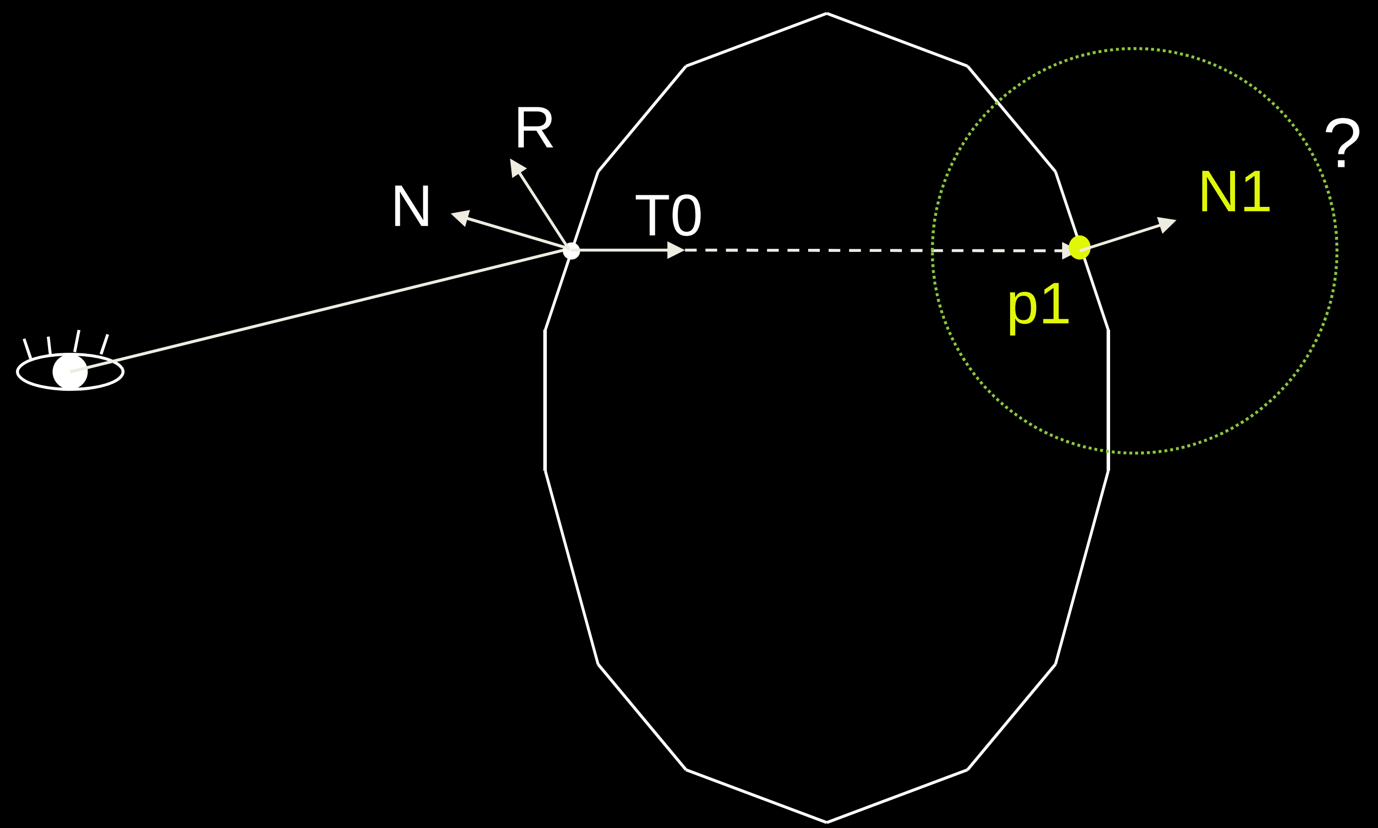


Crystal Transmittance



float3 T0 = refract(-V, N, 1.0 / index);

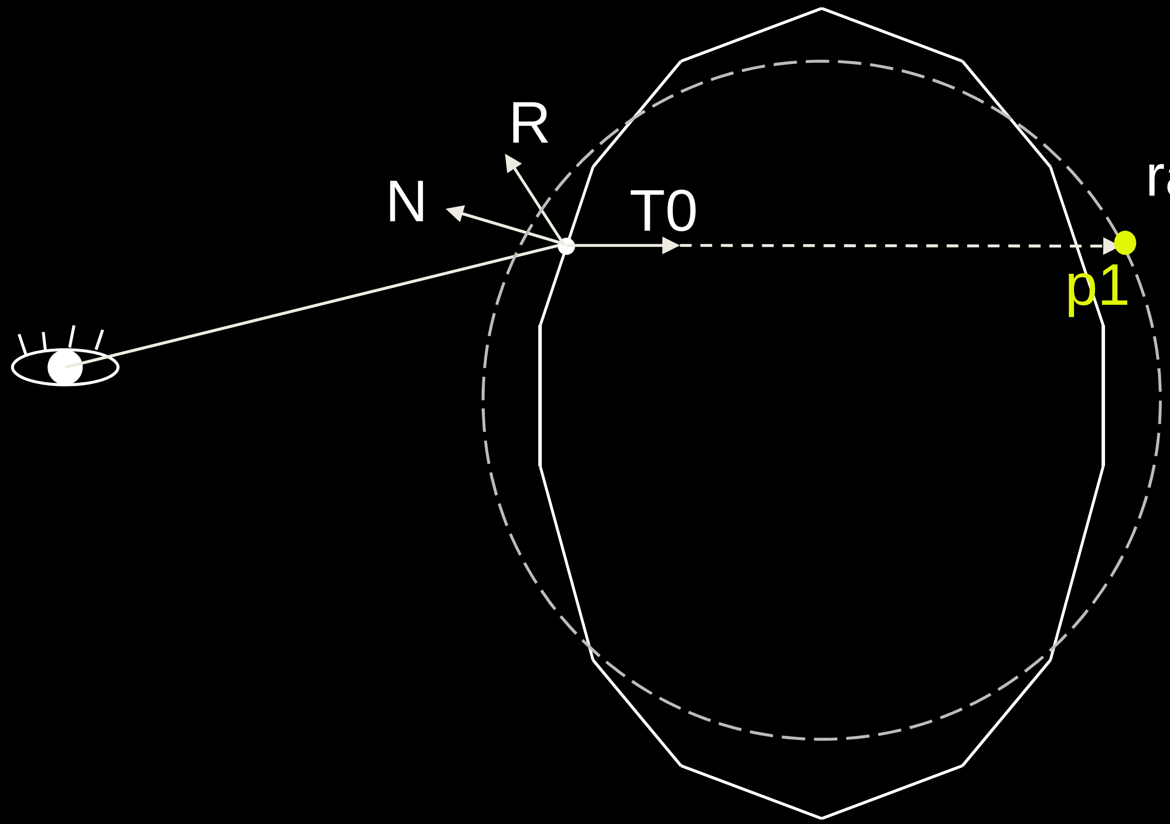
Computing ray-crystal intersections



Computing ray-crystal intersections

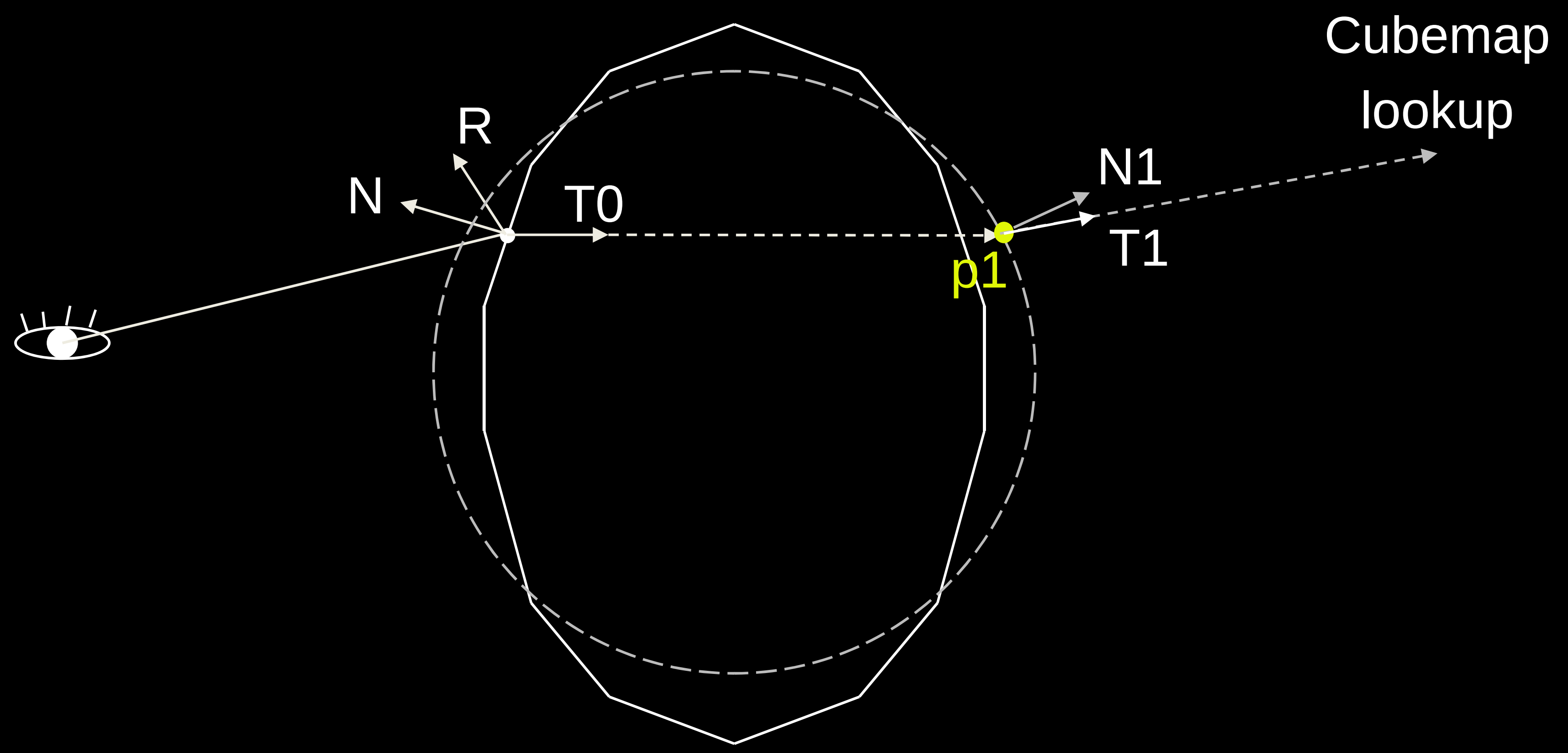
- Avoid ray-tracing the exact crystal triangle set
- The crystal is roughly a sphere
- Ray-sphere intersections are cheap

Computing ray-crystal intersections

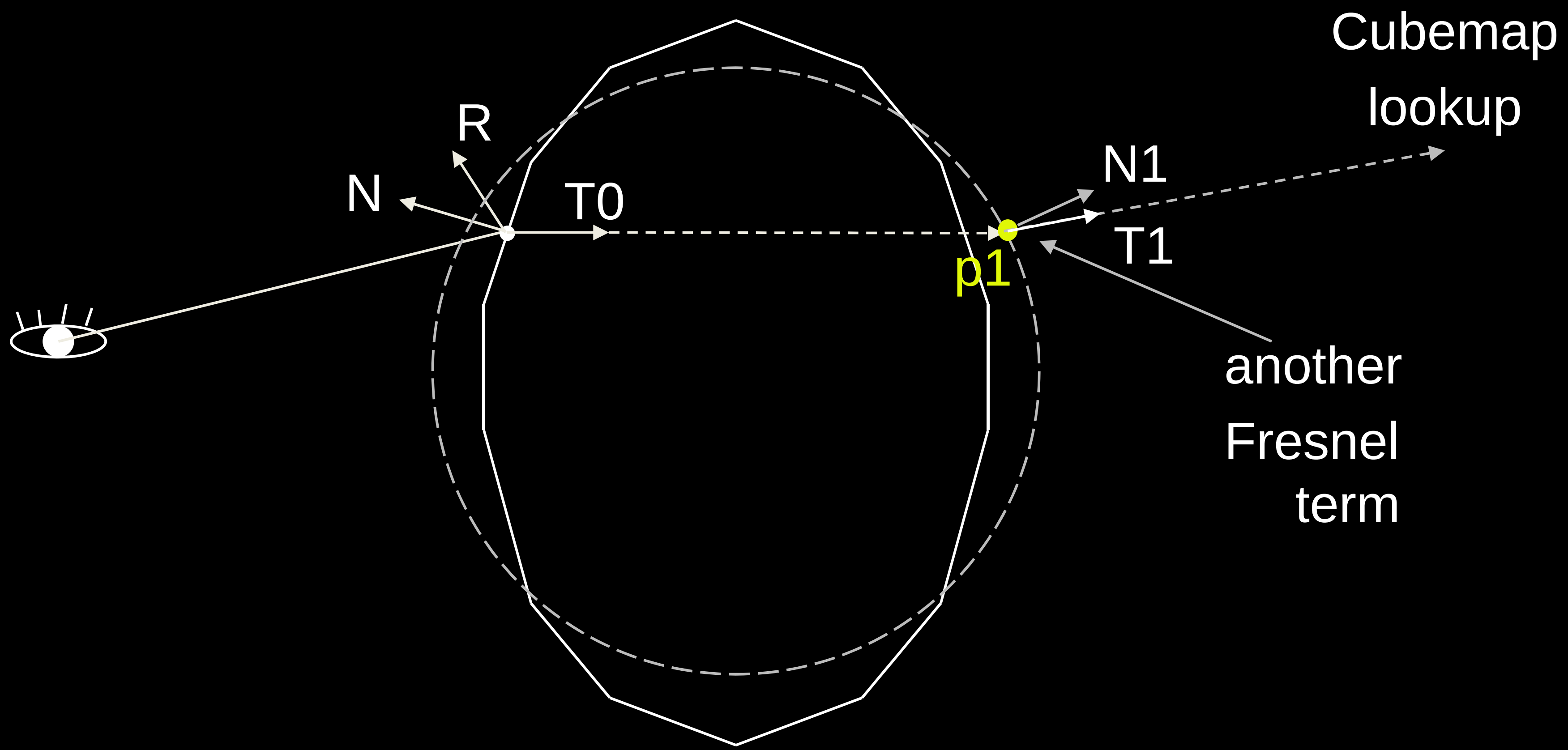


Use a simple
ray-sphere intersection
to find p1

Computing ray-crystal intersections



Computing ray-crystal intersections



float3 T1 = refract(T0, -N1, index);

fresnel1 = fresnelReflectance(N1, T0, F0_out);

Sphere Approximation

- Transmitted light is too uniform

Second Idea

- Change the normals on the sphere to be faceted

```
float3 facetNormal( float3 N, float facetSize )
{
  float3 scaledNormal = N * facetSize;
  float3 scaleandround = float3( round( scaledNormal.x ),
    round( scaledNormal.y ), round( scaledNormal.z ) );
  return normalize( scaleandround );
}
```

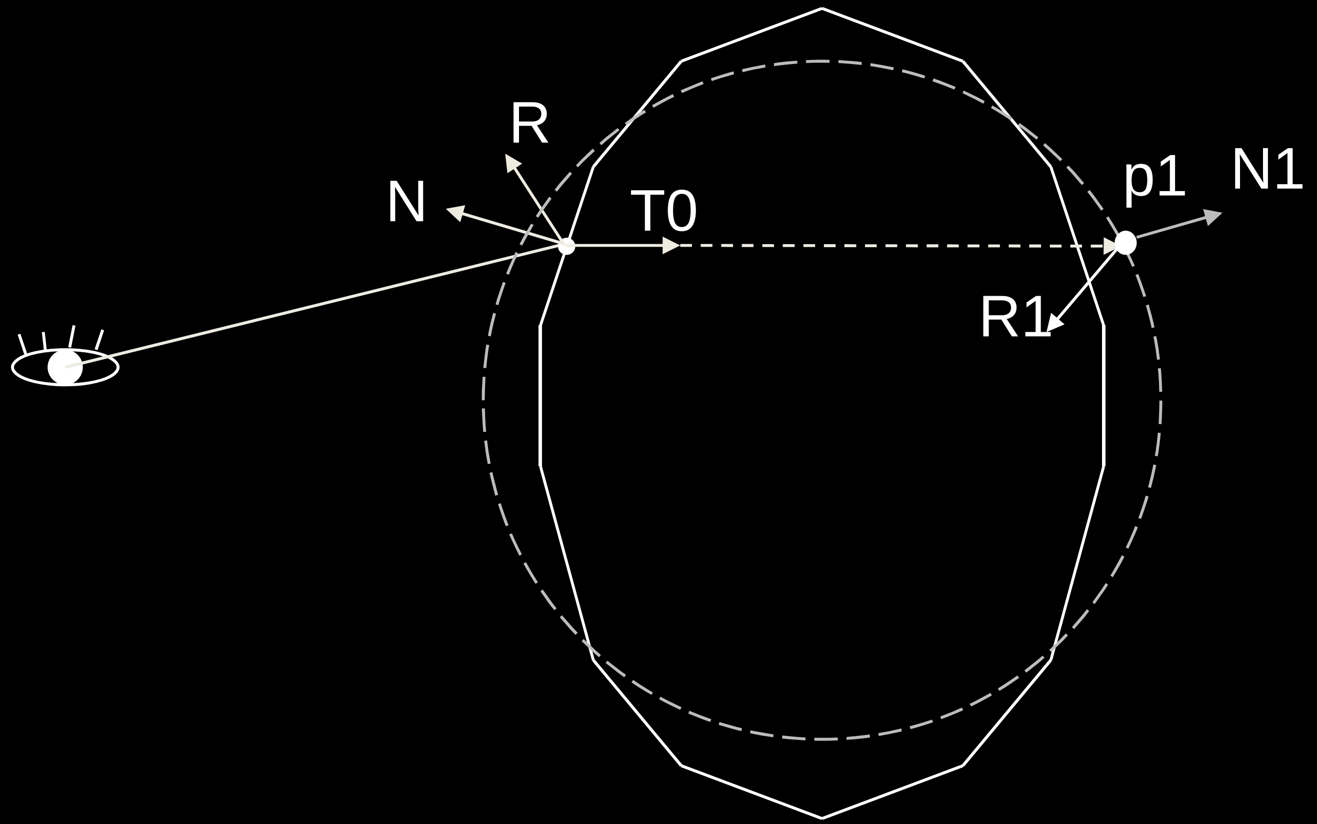
Faceted Normals

- Voila!

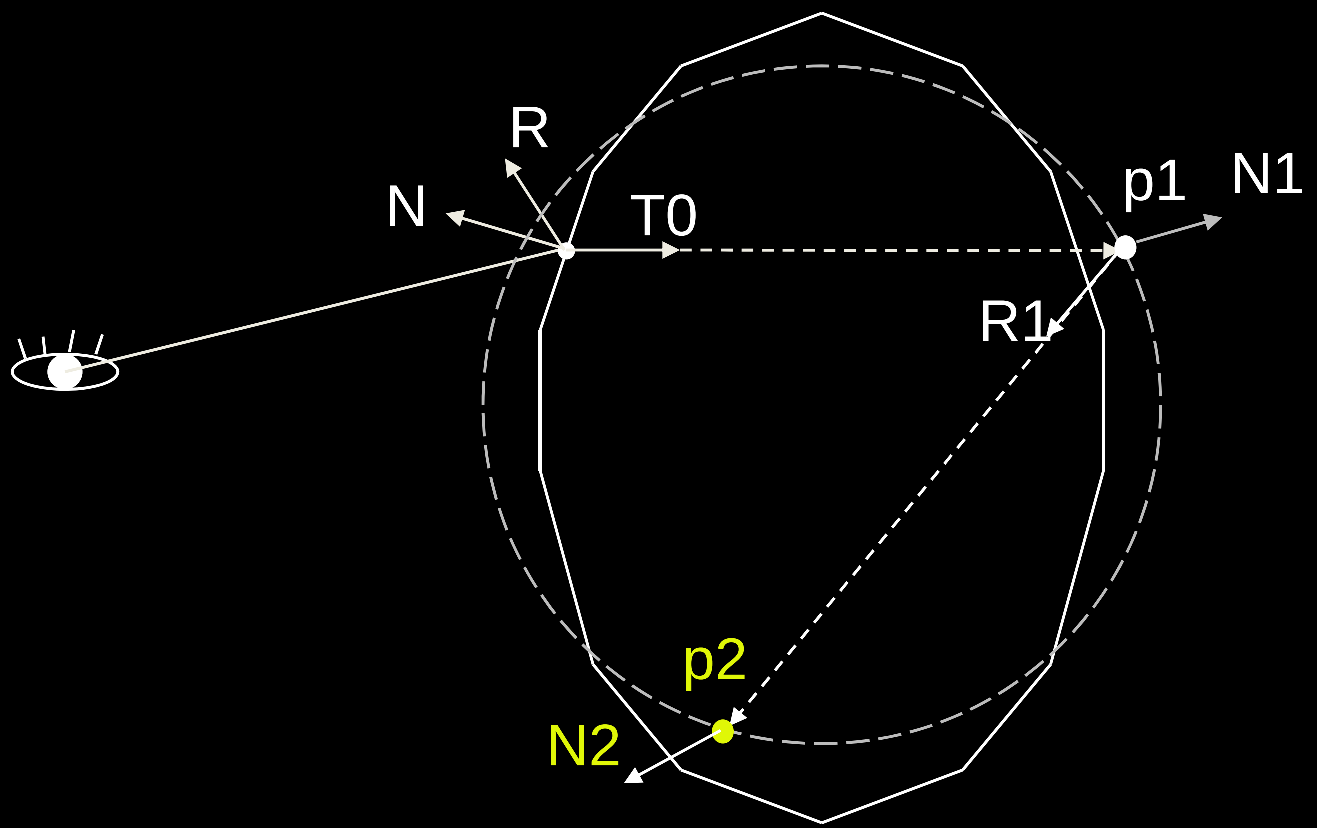
Faceted Normals

- Change the facet_size to your liking

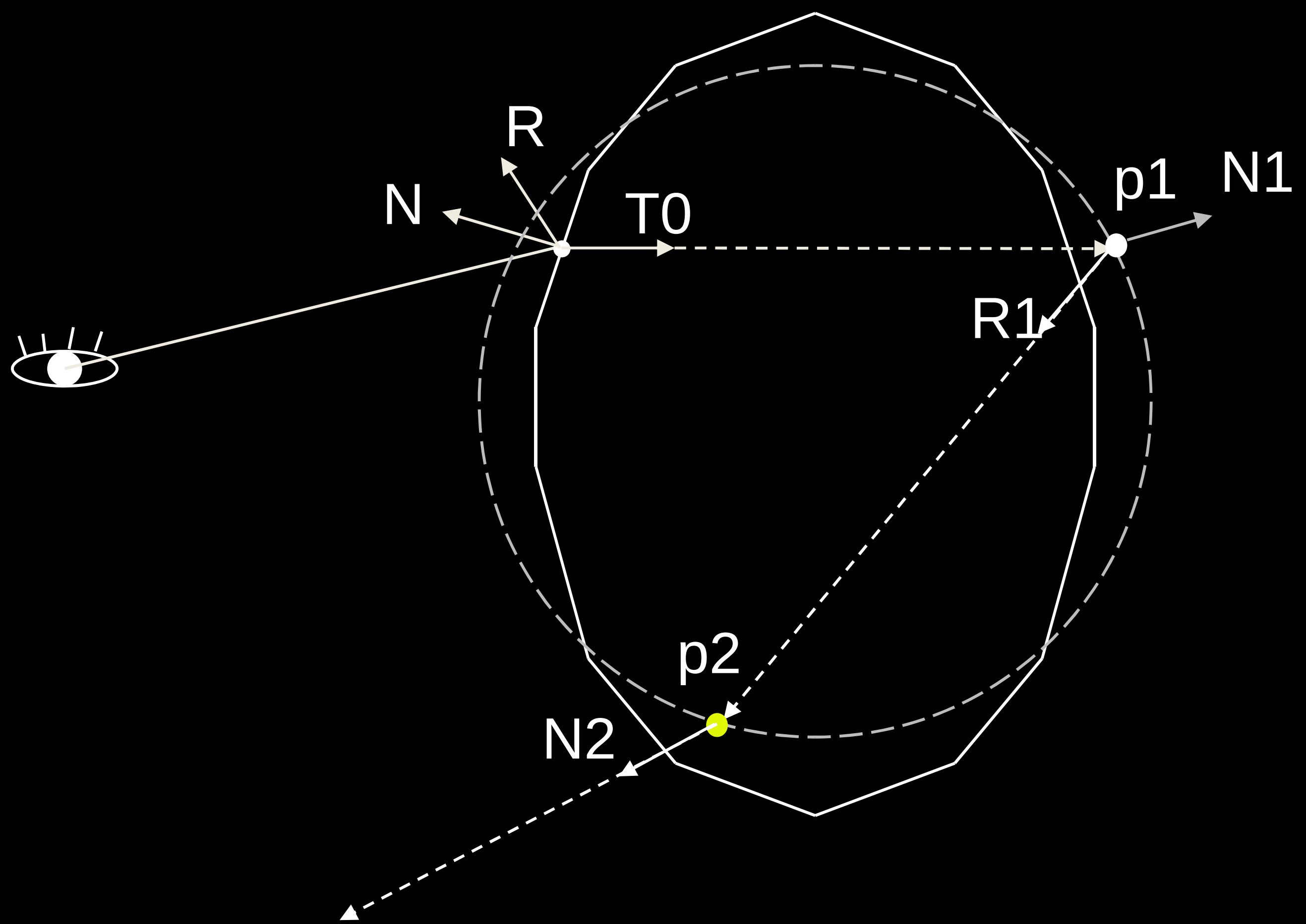
Further transmission



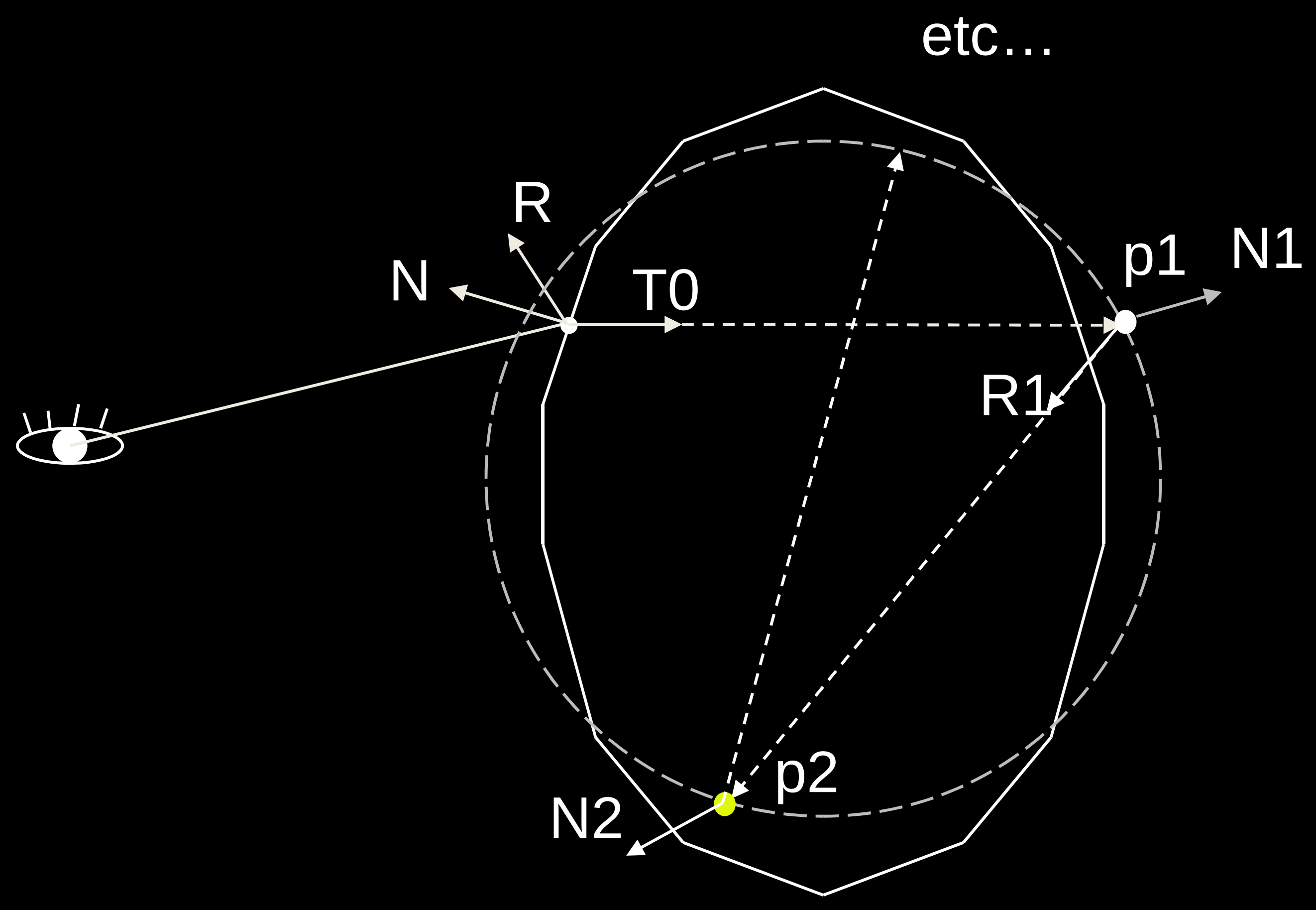
Further transmission



Further transmission

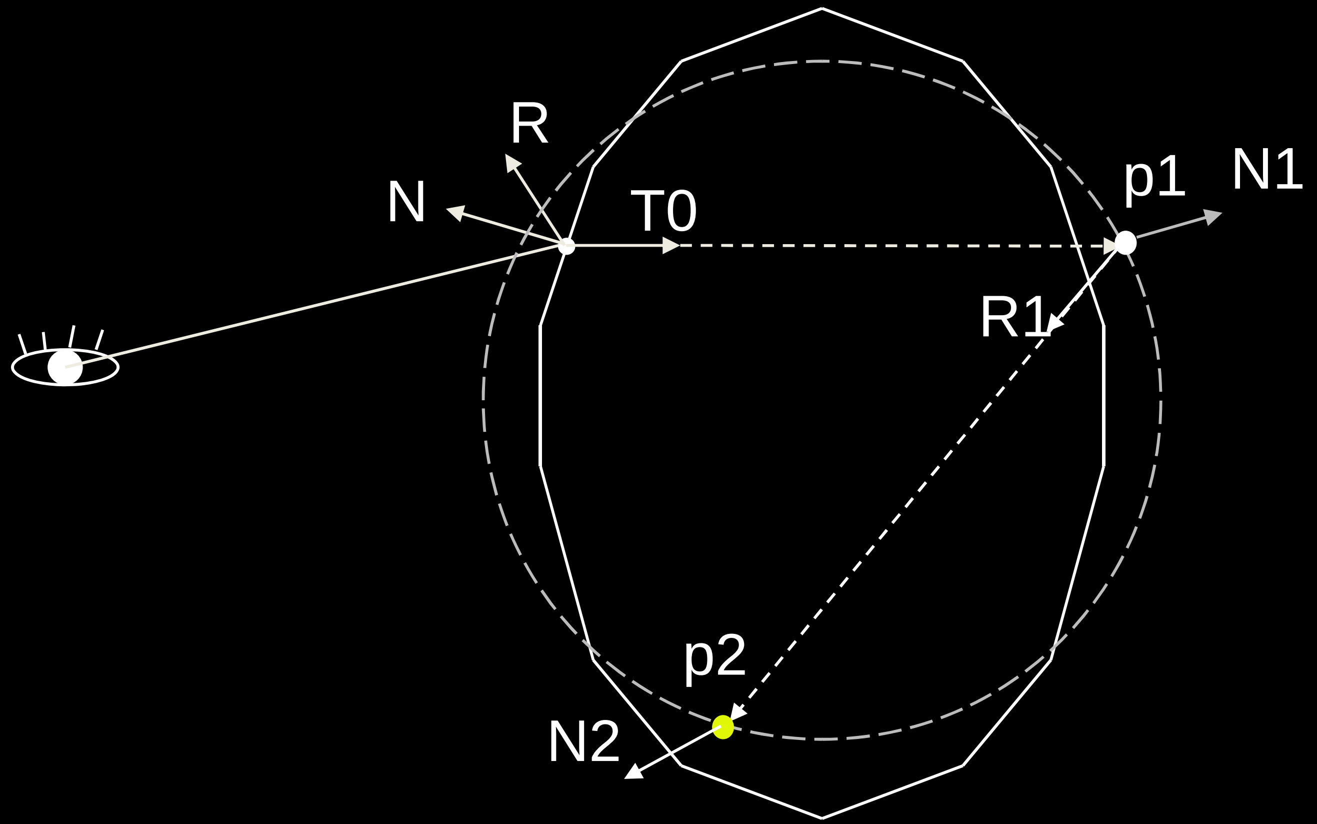


Further transmission



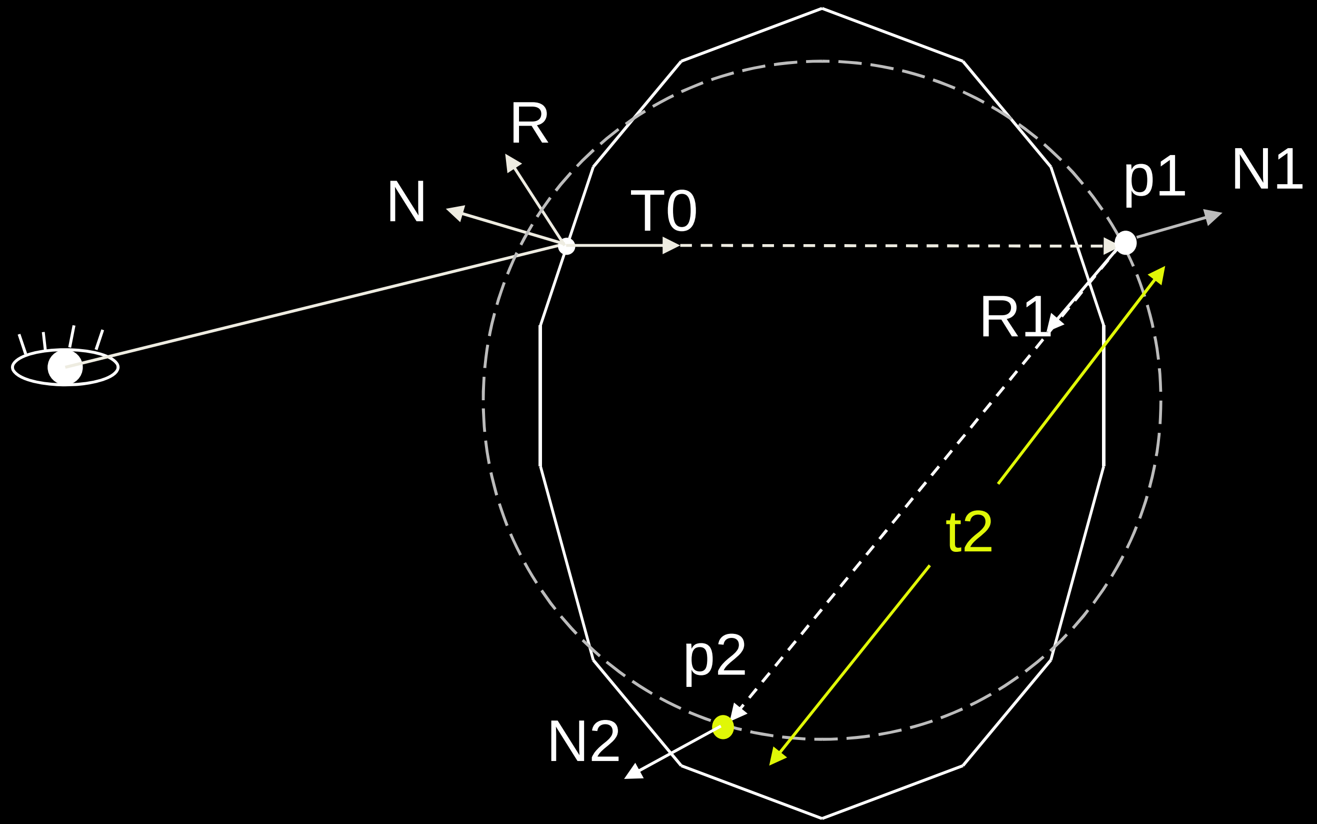
Colored Gemstones

- Light travelling through the crystal is absorbed based on distance



Colored Gemstones

- Light travelling through the crystal is absorbed based on distance



Attenuate by: exp(-t2 * absorp)

Final Crystal Reflectance

```
result.xyz = specularLight + fresnel0 * cube0 + ( 1.0 - fresnel0 ) *
  (
    exp( -t1 * absorp ) * ( ( 1.0 - fresnel1 ) * cube1 +
    fresnel1 * exp( -t2 * absorp ) * ( ( 1.0 - fresnel2 ) * cube2 +
    fresnel2 * exp( -t3 * absorp ) * ( ( 1.0 - fresnel3 ) * cube3 +
    fresnel3 * exp( -t4 * absorp ) * ( ( 1.0 - fresnel4 ) * cube4 +
    fresnel4 * exp( -t5 * absorp ) * ( ( 1.0 - fresnel5 ) * cube5 +
    fresnel5 * exp( -t6 * absorp ) * ( ( 1.0 - fresnel6 ) * cube6 ) ) )
    ) ) )
  );
```

Final Crystal Reflectance

+

reflectance

Final Crystal Reflectance

+

+

reflectance

1st transmittance

Final Crystal Reflectance

reflectance

+

1st transmittance

+

2nd transmittance

+

Final Crystal Reflectance

reflectance | 1st transmittance | 2nd transmittance | 4th transmittance

Final Crystal Reflectance

reflectance

+

1st transmittance

+

2nd transmittance

+

4th transmittance

+

6th transmittance

+

nvision 08 THE WORLD OF VISUAL COMPUTING

Final Crystal Reflectance

reflectance

+

1st transmittance

+

2nd transmittance

+

4th transmittance

6th transmittance

+

specular

=

Final Crystal Reflectance

reflectance

+

1st transmittance

+

2nd transmittance

+

4th transmittance

6th transmittance

+

specular

=

final result

Questions?