Chromatic Aberration

CEDEC 2001
Tokyo, Japan
Mark J. Kilgard
Graphics Software Engineer
NVIDIA Corporation
Chromatic Aberration

- Refraction through surfaces can be wave-length dependent
- Examples:
  - Crystal sculptures
  - Soap bubbles
  - Poor quality lens
  - Prism
  - Rainbows
- Physics of chromatic aberration
  - Physically plausible rendering model is hard
  - But cool hacks are possible today!
Physics of Refraction

- Snell’s law, usually written

\[ n_1 \sin \theta_1 = n_2 \sin \theta_2 \]

where \( n_1 \) & \( n_2 \) are indexes of refraction of two mediums on either side of a refractive interface and \( \theta_1 \) & \( \theta_2 \) are the incoming and outgoing angles of incidence

\[ \eta = \frac{n_2}{n_1} \]
Wavelength Dependent Refraction

- The index of refraction for most translucent materials is wavelength dependent
  - \( n \) is a function of wavelength \( \lambda \)

\[
\theta_1 \quad n_{2\text{red}} > n_{2\text{green}} > n_{2\text{blue}} > n_1
\]

- Chromatic aberration (optics term) is the visual effect of wavelength dependent refraction
Real Refraction is Hard

- Correct refraction is very hard
  - Usually involves multiple interfaces
  - Best technique today’s are expensive global illumination techniques such as ray tracing
- But it’s hard to discern if refraction is “correct”
  - Non-intuitive effect
  - Harder to intuit correctness than reflections
    - Environment mapping fakes convincing reflections, so fake refraction likely to be convincing too
- So simplify
  - One refractive interface
  - Use environment maps
Sketch of a Chromatic Aberration Technique (1)

- Per-vertex, compute
  - Three refraction vectors given surface normal, view direction, & indices of refraction
    - one refraction vector for red, green, and blue
  - different indices of refraction for each
  - compute refraction vectors using vertex programs

- Sample single RGB cube map texture object
  - Extract respective color component for each refraction vector
  - Combine the components in the register combiners
Sketch of a Chromatic Aberration Technique (2)

- Also blend in conventional reflective environment mapping contribution
  - Again, access same RGB environment map
  - Most transparent surfaces refract and reflect
- Scalable technique for different GPUs
  - Single-texture GPU, 4 passes
  - Dual-texture GPU (TNT, GeForce 1/2), 2 passes
  - Quad-texture GPU (GeForce3), 1 pass
  - Use CPU math when vertex programs not available
- Credit for idea & demo: Simon Green, NVIDIA
2D View of Chromatic Aberration Rendering Technique

Cube map environment map

- View vector
- Surface normal vector
- Surface (refraction interface)
- Reflected view vector
- Red, green, & blue refracted view vectors
Example Refraction/Reflection
Environment Map Cube Map

Credit: Paul Debevec
Chromatic Aberration Example

Crystal bunny floating in the Uffizi Gallery, Florence

Credit: Simon Green, NVIDIA
Per-vertex Refraction Vector Computations

- Two refraction vector techniques
  - Physically-based vector approach
    \[ R = \left( \eta \hat{N} \cdot \hat{I} - \sqrt{\hat{I} - \eta^2 \left( 1 - (\hat{N} \cdot \hat{I})^2 \right)} \right) \hat{N} - \eta \hat{I} \]
  - RenderMan approach
    \[ k = 1 - \eta^2 \left( 1 - (\hat{I} \cdot \hat{N})^2 \right) \]
    \[ R = \begin{cases} k < 0, & (0,0,0) \\ k \geq 0, & \eta \hat{I} - \left( \eta \hat{I} \cdot \hat{N} + \sqrt{k} \right) \hat{N} \end{cases} \]
- Where \( \hat{I} \) = incident vector (to eye), \( \hat{N} \) = normal, and \( \eta \) = ratio of two indices of refraction; to compute the refraction vector \( R \)
Vertex Program Formulation

!!VP1.0 # refraction
# c[4],c[5],c[6] = upper 3x3 of transposed inverse modelview matrix
DP3  R11.x, c[4], v[NRML] ; # transform normal to eye-space
DP3  R11.y, c[5], v[NRML] ; # R11 = eye-space normal
DP3  R11.z, c[6], v[NRML] ; # (assume normalized)

# c[8],c[9],c[10],c[11] = 4x4 modelview matrix
DP4  R9.x, c[8], v[OPOS] ; # transform position to eye-space
DP4  R9.y, c[9], v[OPOS] ;
DP4  R9.z, c[10], v[OPOS] ;
DP4  R9.w, c[11], v[OPOS] ; # R9 = eye-space position

# R0 = eye-space eye vector
ADD    R0, -R9, c[59]; # c[59] = position in eye-space; (0,0,0,1)

DP3  R8.w, R0, R0 ;
RSQ   R8.w, R8.w ;
MUL   R8, R0, R8.w ; # R8 = normalized eye vector

DP3  R0.x, R11, -R8; # R0 = N • -E = Ndotl
First Refraction Based on $\eta_1$

# $c[58] = (\eta_1, \eta_1^*\eta_1, 0.0, 1.0)$

# $R1.x = 1-(N\cdot I)^2$
MAD R1.x, -R0.x, R0.x, c[58].w ;
# $R1.x = k = 1.0- \eta_1^*\eta_1*R1.x$
MAD R1.x, R1.x, c[58].y, -c[58].w ;

RSQ R2.x, R1.x ;
RCP R2.x, R2.x ;  # $R2.x = \sqrt{k}$
MAD R2.x, c[58].x, R0.x, R2.x ;
MUL R2, R11, R2.x ;
MAD R2, c[58].x, -R8, R2 ;  # $R = \eta_1*I-(\eta_1^*N\cdot I+\sqrt{k})*N$

DP3 o[TEX0].x, c[12], R2 ;  # rotate refraction ray
DP3 o[TEX0].y, c[13], R2 ;  # by cube map transform
DP3 o[TEX0].z, c[14], R2 ;
Second Refraction Based on $\eta_2$

# $c[57] = (\eta_2, \eta_2 \ast \eta_2, 0.0, 1.0 )$

# $R1.x = 1 - (N \cdot I)^2$
MAD $R1.x$, $-R0.x$, $R0.x$, $c[57].w$ ;
# $R1.x = k = 1.0 - \eta_2 \ast \eta_2 \ast R1.x$
MAD $R1.x$, $R1.x$, $c[57].y$, $-c[57].w$ ;

RSQ $R2.x$, $R1.x$ ;
RCP $R2.x$, $R2.x$ ; # $R2.x = sqrt(k)$
MAD $R2.x$, $c[57].x$, $R0.x$, $R2.x$ ;
MUL $R2$, $R11$, $R2.x$ ;
MAD $R2$, $c[57].x$, $-R8$, $R2$ ; # $R = \eta_2 \ast I - (\eta_2 \ast N \cdot I + sqrt(k)) \ast N$

DP3 $o[TEX1].x$, $c[12]$, $R2$ ; # rotate refraction ray
DP3 $o[TEX1].y$, $c[13]$, $R2$ ; # by cube map transform
DP3 $o[TEX1].z$, $c[14]$, $R2$ ;
Third Refraction Based on $\eta_3$

# $c[56] = (\eta_3, \eta_3*\eta_3, 0.0, 1.0)$

# $R1.x = 1-(Ndotl)^2$
MAD R1.x, -R0.x, R0.x, c[56].w ;

# $R1.x = k = 1.0 - \eta_3*\eta_3*R1.x$
MAD R1.x, R1.x, c[56].y, -c[56].w ;

RSQ R2.x, R1.x ;
RCP R2.x, R2.x ;  # $R2.x = sqrt(k)$
MAD R2.x, c[56].x, R0.x, R2.x ;
MUL R2, R11, R2.x ;
MAD R2, c[56].x, -R8, R2 ;  # $R = \eta_3*I-(\eta_3*Ndotl+sqrt(k))*N$

DP3 o[TEX2].x, c[12], R2 ;  # rotate refraction ray
DP3 o[TEX2].y, c[13], R2 ;  # by cube map transform
DP3 o[TEX2].z, c[14], R2 ;
Compute Simple Fresnel Term

# c[62] = (fresnel, fresnel, fresnel 1.0 )

# Fresnel approximation = (1-(Ndotl))^2
ADD  R0.x, c[62].w, R0.x;
MUL  R0.x, R0.x, R0.x;
MUL  o[COL0], R0.x, c[62];
Compute Reflection

# c[64].z = 2.0
MUL R0, R11, c[64].z;  # R0 = 2*N
DP3 R3.w, R11, R8;     # R3.w = N dot E
MAD R3, R3.w, R0, -R8;  # R3 = 2*N * (N dot E) - E

# transform reflected ray by cube map transform
DP3 o[TEX3].x, c[12], R3;
DP3 o[TEX3].y, c[13], R3;
DP3 o[TEX3].z, c[14], R3;
Transform Vertex to Clip Space

// transform vertex to clip space
DP4 o[HPOS].x, c[0], v[OPOS] ;
DP4 o[HPOS].y, c[1], v[OPOS] ;
DP4 o[HPOS].z, c[2], v[OPOS] ;
DP4 o[HPOS].w, c[3], v[OPOS] ;
END
Per-fragment Math to Combine Reflection and Refractions

- **Combine refractions**
  
  $$\text{RefractedColor} = \text{red}(\text{cubemap}(\text{tex0})) + \text{green}(\text{cubemap}(\text{tex1})) + \text{blue}(\text{cubemap}(\text{tex2}))$$

- **Compute reflected color**
  
  $$\text{ReflectedColor} = \text{cubemap}(\text{tex3})$$

- **Combine color based on Fresnel term**
  
  $$\text{color} = \text{primaryColor} \times \text{ReflectedColor} + (1-\text{primaryColor}) \times \text{RefractedColor}$$
Improvements, Optimizations, & Suggestions

- Dynamic environment mapping
  - Render environment map per-frame
- Vertex program optimizations
  - Combine scalar instructions together
  - Perform vector math in cube map space rather than in eye space to save DP3s
- Very cheap refraction approximation

\[ R = \hat{E} - \eta \hat{N} \]

Bends ray towards normal in proportion to \( \eta \)
Conclusions

- Chromatic aberration
  - Environment mapping-style technique
  - Convincing if not physically plausible
  - Requires cube maps
  - Accelerated by
    - Vertex programs
    - Multiple hardware texture units

- Inventor
  - Simon Green, NVIDIA