Real-time Graphics

6. Reflections, Refractions

Martin Samuelčík

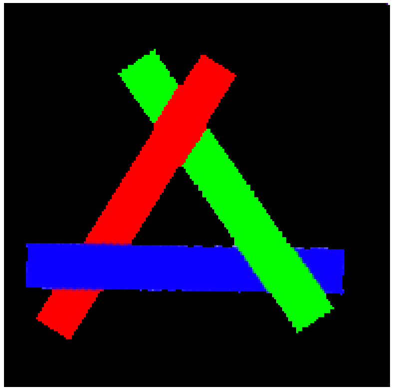
Juraj Starinský

Blending

- Simulating transparent materials
- Alpha value, RGBA model
- Using alpha test – fragment alpha value is tested against given constant
- Using blending of colors – fragment color is blended with color from color buffer using alpha
- Source of alpha – texture, constant, shader, …
- Classic blending
  - *glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)*

Blending

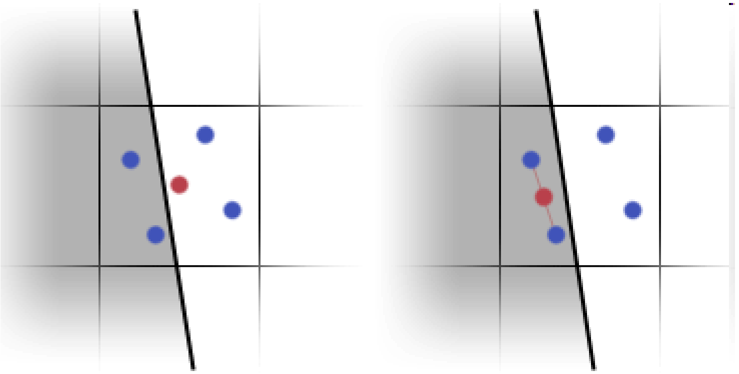
Blending - problems



- Ordering of triangles (expensive?)
- Depth buffer writing disabled
- Algorithm
  - Render opaque objects
  - Disable writing to depth buffer
  - Render transparent objects with blending
- Use alpha test
- Still problem, normal blending is not additive
- Use additive blending
  - *glBlendFunc(GL_SRC_ALPHA, GL_ONE)*

Alpha to coverage

- Part of multisample extension, for full screen antialiasing
  - GL_ARB_multisample
  - GLX_ARB_multisample
  - WGL_ARB_multisample



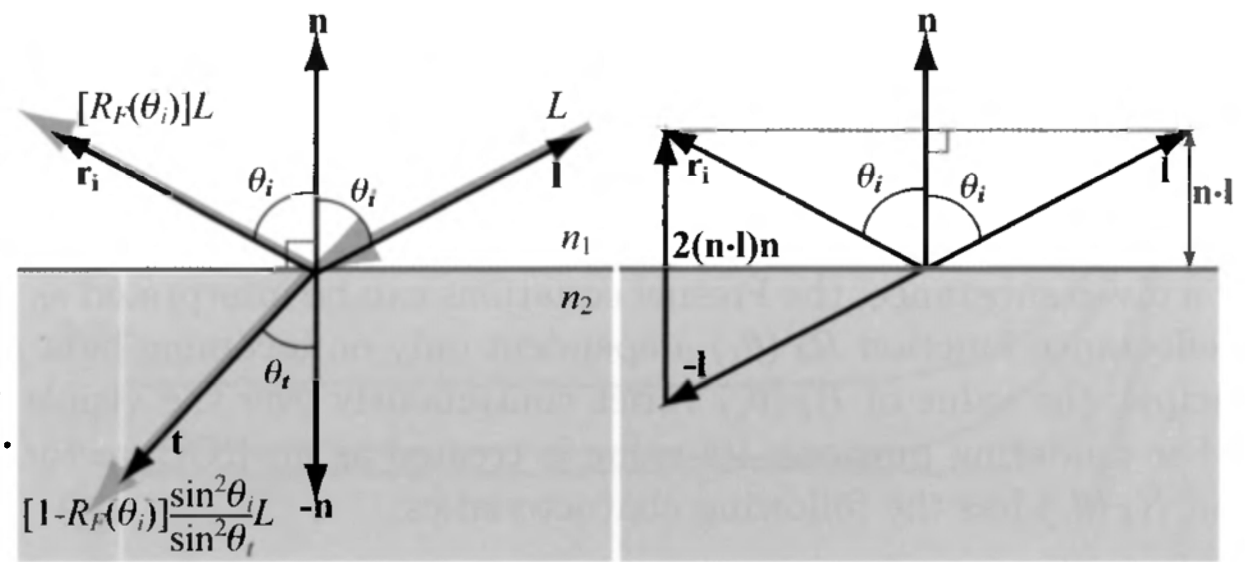
- Sampling fragment multiple times (4x, 8x,)
- Alpha to coverage -> alpha holds percentual number of covered samples
- Used when alpha represents coverage (grass, leafs)

Reflection & refraction

- Optical properties on surface – boundary between two substances
- Fresnel equations, Fresnel reflectance
- Snell's law

$$n_1 \sin(\theta_i) = n_2 \sin(\theta_t)$$

$$\mathbf{t} = (w - k)\mathbf{N} - n\mathbf{L},$$
$$w = n(\mathbf{L} \cdot \mathbf{N}),$$
$$k = \sqrt{1 + (w - n)(w + n)}.$$
$$n = n_1 / n_2$$



Fresnel reflectance

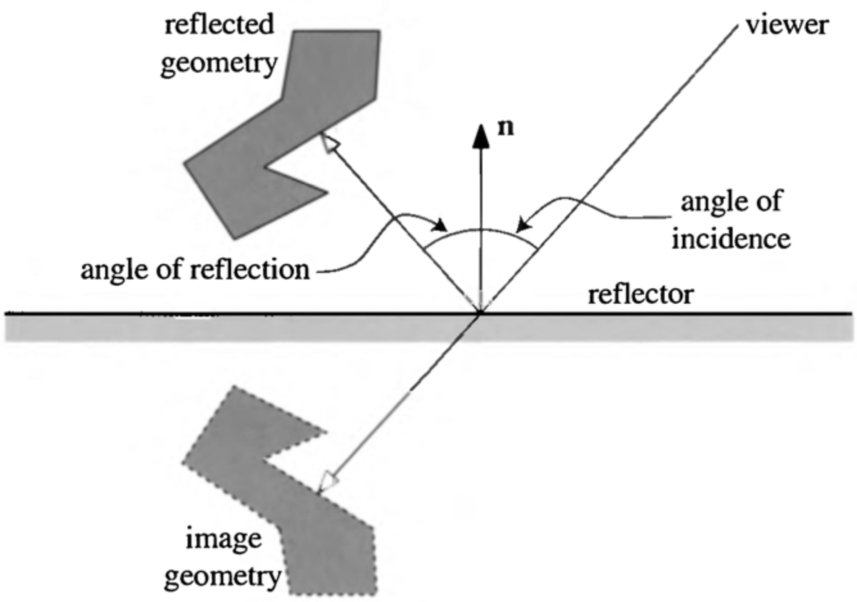
- Intensity $R_F(\theta_i)$ of reflected light based on angle θ_i
- Approximation: $R_F(\theta_i) \approx R_F(0^\circ) + (1 - R_F(0^\circ))(1 - \overline{\cos}\,\theta_i)^5.$
- Characteristic specular color $R_F(0^o)$
- Example:
  - CD box under small angle reflect small amount of light, cover under plastic is clearly visible
  - CD box under large angle reflect large amount of light, cover under plastic is not visible

n-refractive factor $R_F(0^\circ) = \left(\frac{n-1}{n+1}\right)^2$

Metal	$R_F(0^\circ)$ **(Linear)**
Gold	1.00,0.71,0.29
Silver	0.95,0.93,0.88
Copper	0.95,0.64,0.54
Iron	0.56,0.57,0.58
Aluminum	0.91,0.92,0.92

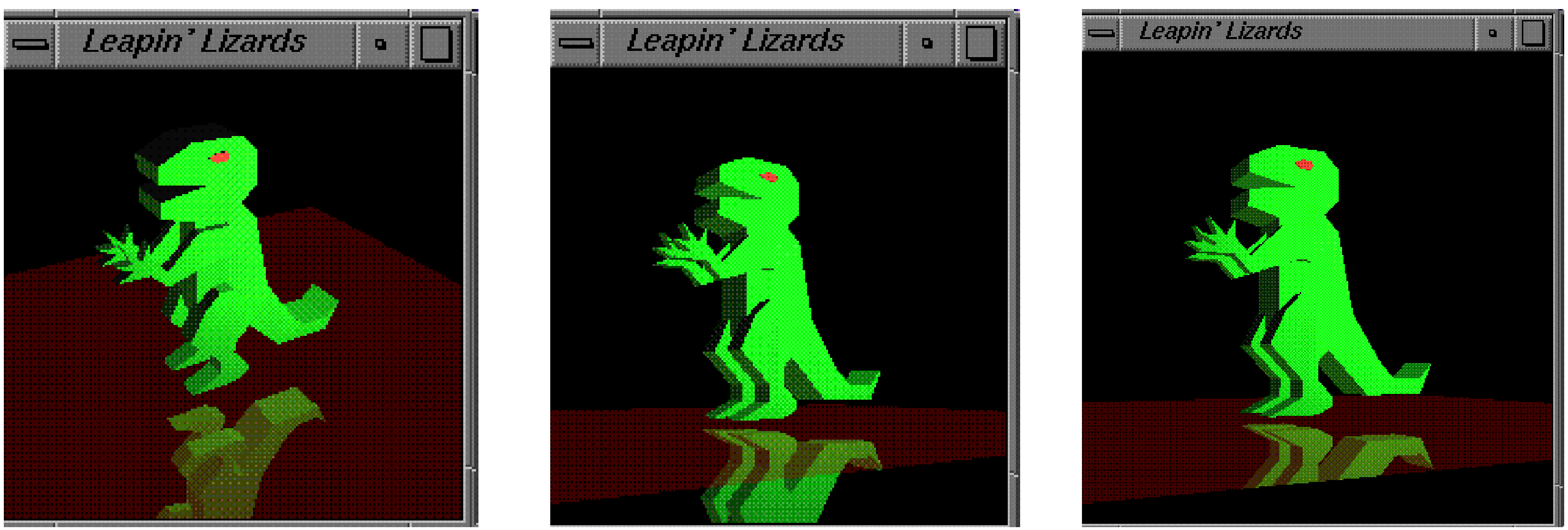
Planar reflections

- Law of reflection - states that the angle of incidence is equal to the angle of reflection
- Reflected geometry appears as geometry behind the reflector



Reflection - geometry

- Draw reflected object mirrored in relation to reflector
- Draw reflector with blending
- Draw object
- Remove parts of mirrored object - stencil buffer

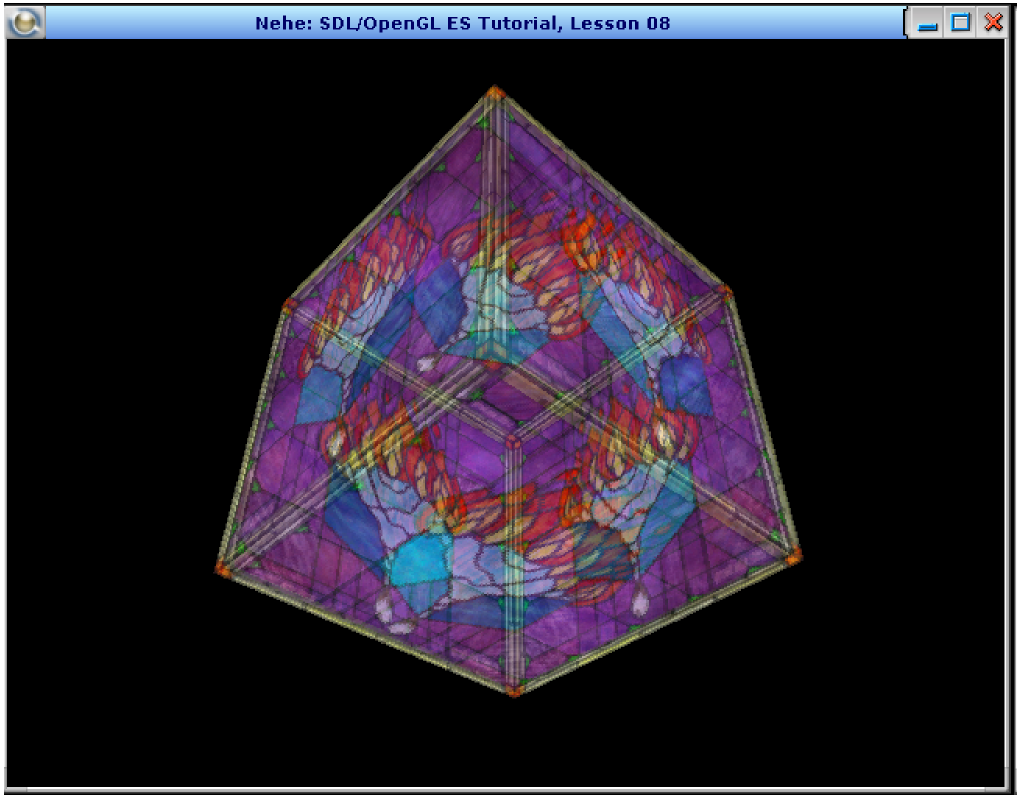


opengl.org

Reflection - geometry

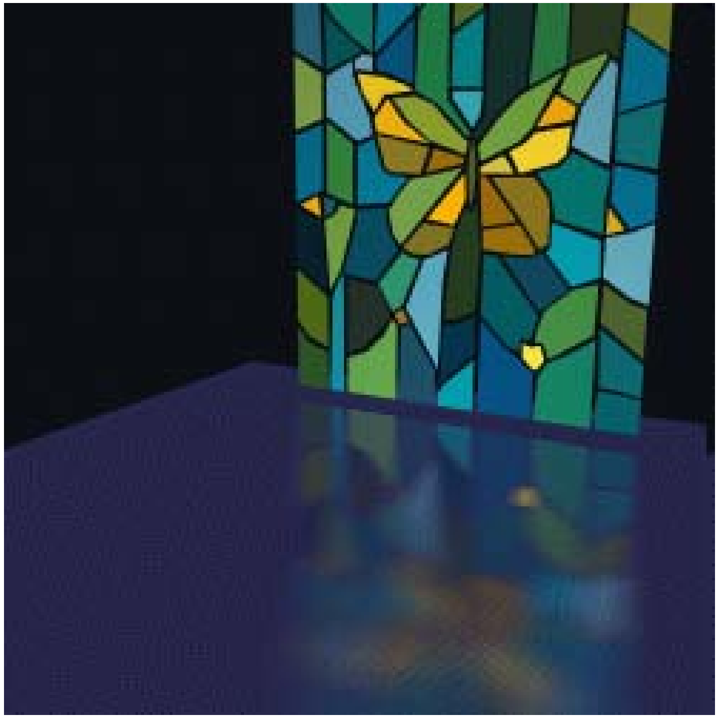
Refraction - geometry

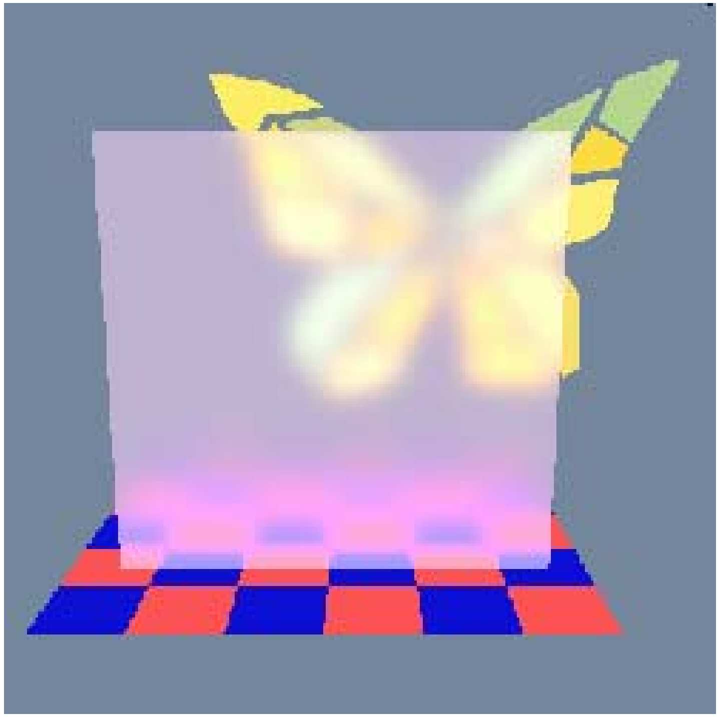
- Draw refracted geometry with special rotate & scale transformation based on refraction factor
- Use blending



Fuzzy reflection & refraction

- Accumulation buffer – jittering of mirror object
- Using fog for distance effect





Paul Diefenbach

Using textures

- Rendering to texture in several passes
- Render scene from mirrored position – reflections
- Render scene from slightly changed position – refractions
- Optimalization - using clip plane, culling, …
- "Looking through the glass"

Using textures

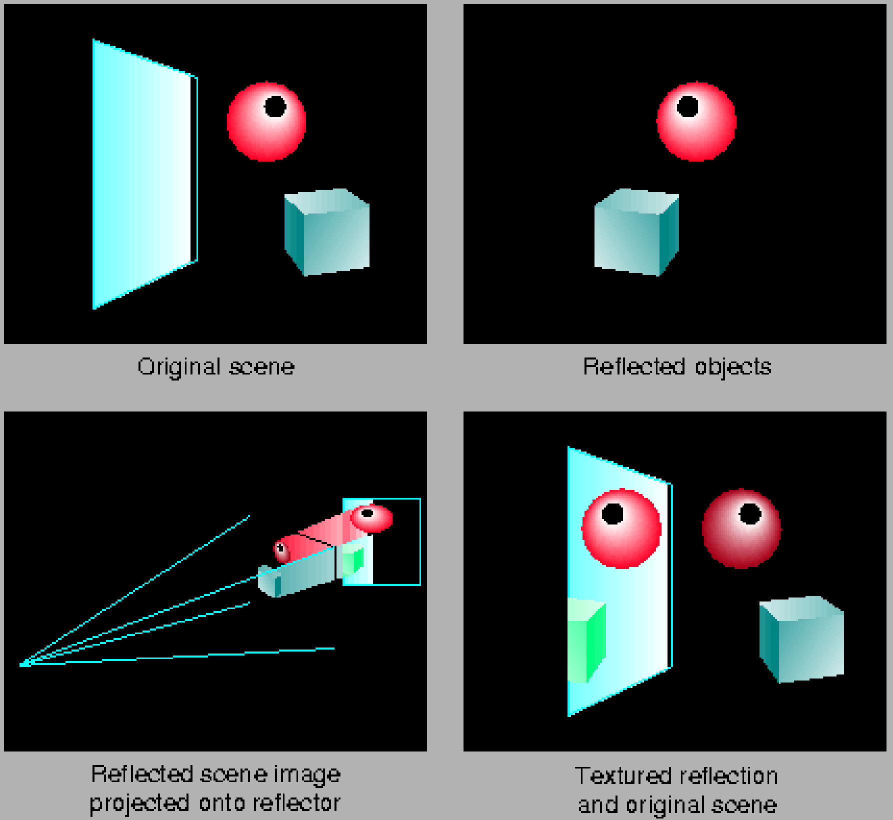
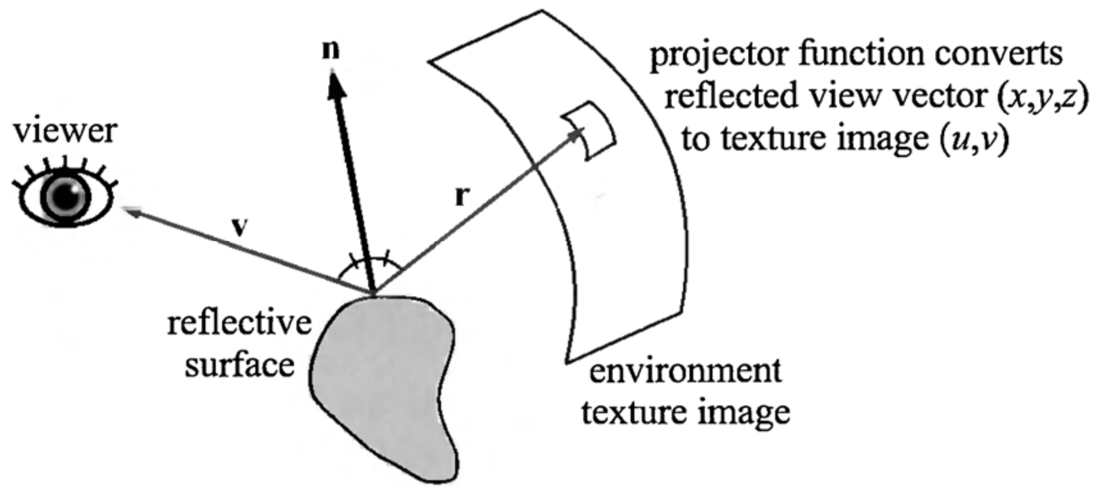


Figure 58. Masking Reflections Using Projective Texture

Environment mapping

- Reflection of objects with large distance from reflector
- Best performance for curved surfaces
- Using texture to represent far reflected objects
  - Cube mapping
  - Sphere mapping
  - Dual paraboloid
  - Rendering on fly
  - ...
- Skyboxes



Environment mapping

- Generate or load a two-dimensional image representing the environment.
- For each fragment that contains a reflective object, compute the normal at the location on the surface of the object.
- Compute the reflected view vector from the view vector and the normal.
- Use the reflected view vector to compute an index into the environment map that represents the incoming radiance in the reflected view direction.
- Use the texel data from the environment map as incoming radiance.

Cube mapping

- Cube texture – consists of 6 2D textures, POS_X, NEG_X, POS_Y, NEG_Y, POS_Z, NEG_Z
- GL_ARB_texture_cube_map, OpenGL 1.3
- GLSL
  - samplerCube – cube texture
  - textureCube - fetches texel from cube map based on reflected vector in world coordinates
- Getting cube map value for fragment
  - The coordinate of reflected vector with highest magnitude gives face
  - Remaining two coordinates are divided by third and scaled to range [0,1]
  - Example: (0.287, -0.944, 0.164) gives face NEG_Y, texture coordinates are s = (0.287/0.944*0.5) + 0.5, t = (0.164/0.944*0.5) + 0.5

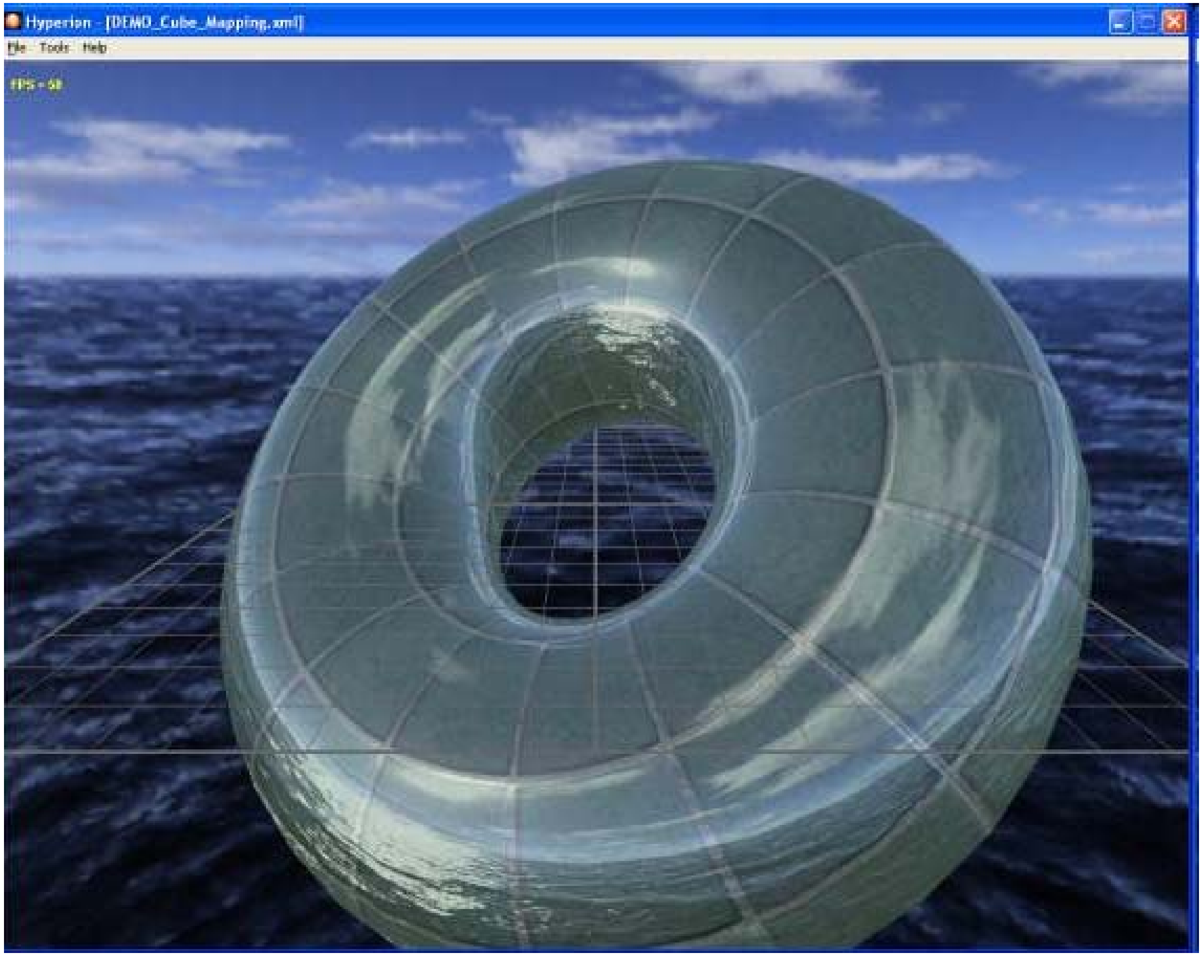
Cube mapping - GLSL

```
uniform mat4 ModelWorld4x4;
uniform vec3 CameraPos;

void main()
{
	gl_Position = ftransform();

	// Color map texture coordinates.
	// Increase a little bit the tiling by 2.
	gl_TexCoord[0] = gl_MultiTexCoord0 * 2.0;

	mat3 ModelWorld3x3 = GetLinearPart( ModelWorld4x4 );

	// find world space position.
	vec4 WorldPos = ModelWorld4x4 *  gl_Vertex;

	// find world space normal.
	vec3 N = normalize( ModelWorld3x3 * gl_Normal );

	// find world space eye vector.
	vec3 E = normalize( WorldPos.xyz - CameraPos.xyz );

	// calculate the reflection vector in world space.
	gl_TexCoord[1].xyz = reflect( E, N );
}
```

```
uniform samplerCube cubeMap;
uniform sampler2D colorMap;

const float reflect_factor = 0.5;

void main (void)
{
	vec4 output_color;

	// Perform a simple 2D texture look up.
	vec3 base_color = texture2D(colorMap, gl_TexCoord[0].xy).rgb;

	// Perform a cube map look up.
	vec3 cube_color = textureCube(cubeMap, gl_TexCoord[1].xyz).rgb;

	// Write the final pixel.
	gl_FragColor = vec4( mix(base_color, cube_color, reflect_factor), 1.0);
}
```

http://www.ozone3d.net/tutorials/

Cube mapping



http://www.ozone3d.net/tutorials/

Water rendering

- Render reflection texture (local reflection)
- Render refraction texture
- Add specular light from sun
- Add environmental map (global reflection)
- Total reflection is sum of reflection texture, sun and environmental map
- Use Fresnel term to combine reflection and rippled refraction

Water rendering

Valve

Water rendering

Valve

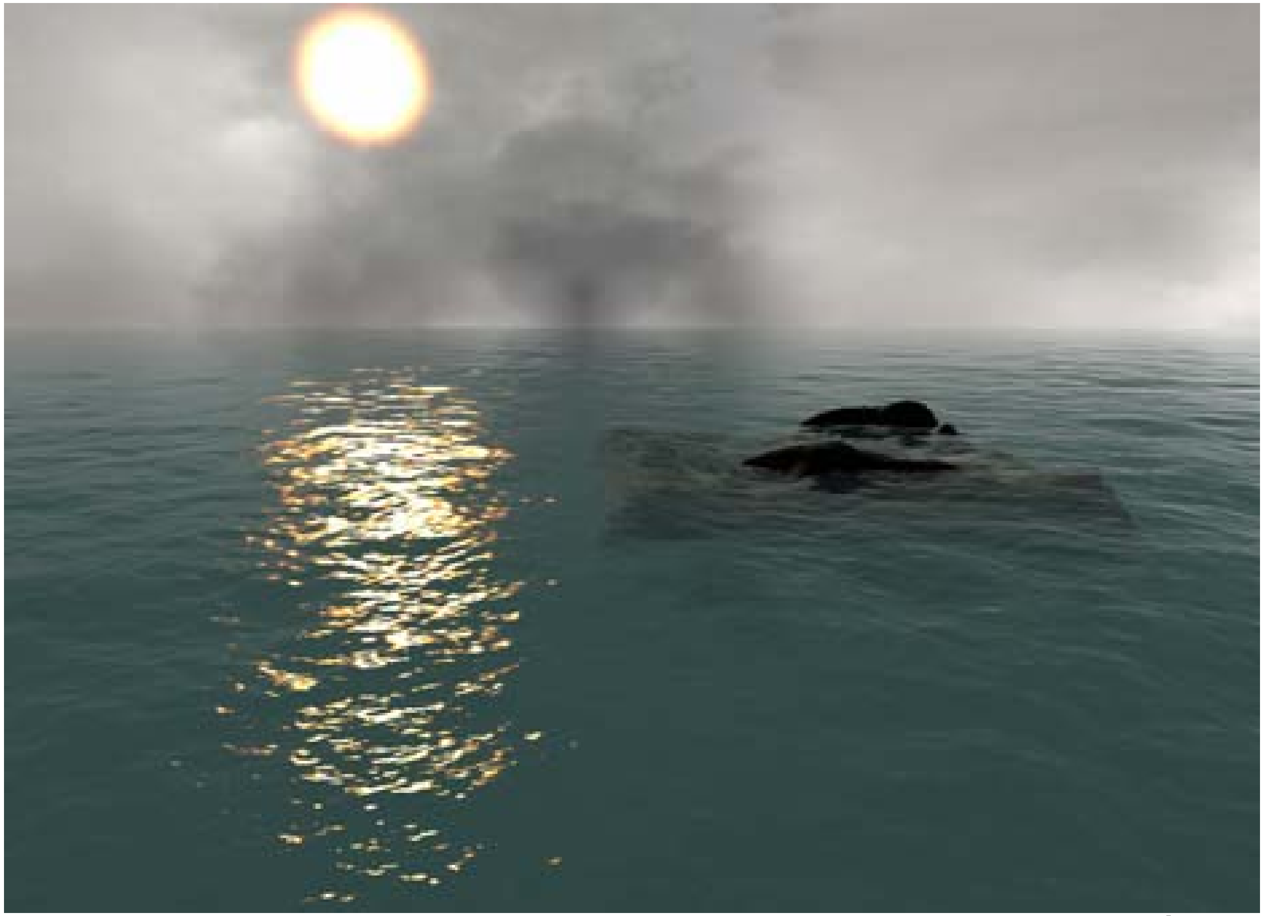
Water Shader

```
float4 main( PS_INPUT i ) : COLOR
{
   // Load normal and expand range
   float4 vNormalSample = tex2D( NormalSampler, i.vBumpTexCoord );
   float3 vNormal = vNormalSample * 2.0 - 1.0;

   float ooW = 1.0f / i.W;  // Perform division by W only once

   float2 vReflectTexCoord, vRefractTexCoord;

   float4 vN; // vectorize the dependent UV calculations (reflect = .xy, refract = .wz)
   vN.xy = vNormal.xy;
   vN.w = vNormal.x;
   vN.z = vNormal.y;
   float4 vDependentTexCoords = vN * vNormalSample.a * g_ReflectRefractScale;

   vDependentTexCoords += ( i.vReflectXY_vRefractYX * ooW );
   vReflectTexCoord = vDependentTexCoords.xy;
   vRefractTexCoord = vDependentTexCoords.wz;

   float4 vReflectColor = tex2D( ReflectSampler, vReflectTexCoord ) * vReflectTint; // Sample reflection
   float4 vRefractColor = tex2D( RefractSampler, vRefractTexCoord ) * vRefractTint; // and refraction

   float3 vEyeVect = texCUBE( NormalizeSampler, i.vTangentEyeVect ) * 2.0 - 1.0;

   float fNdotV = saturate( dot( vEyeVect, vNormal ) ); // Fresnel term
   float fFresnel = pow( 1.0 - fNdotV, 5 );

   if( g_bReflect && g_bRefract ) {
      return lerp( vRefractColor, vReflectColor, fFresnel );
   }
   else if( g_bReflect ) {
      return vReflectColor;
   } else if( g_bRefract ) {
      return vRefractColor;
   } else {
      return float4( 0.0f, 0.0f, 0.0f, 0.0f );
   }
}
```

Valve

Water rendering

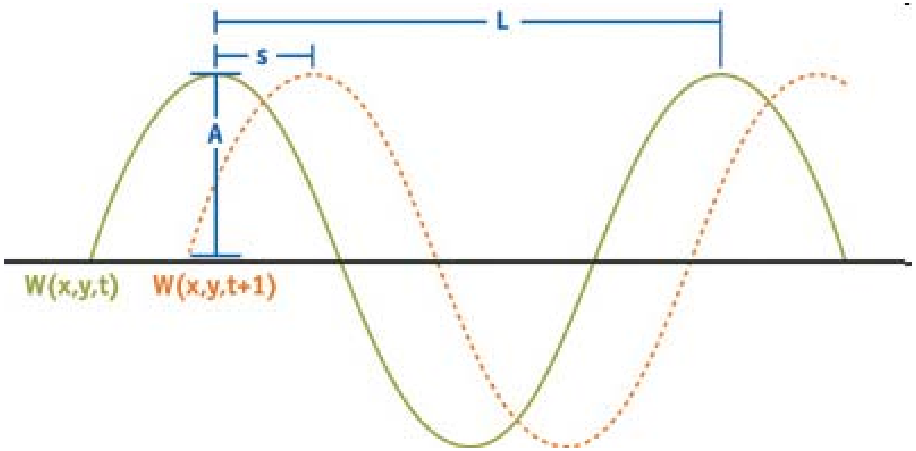
result = (1-fresnel)*refr + fresnel*(refl_local.rgb*refl_local.a + (1-refl_local.a)*(refl_global + refl_sun))



Claes Johanson

Water surface

- Representation
  - Geometry
  - Height field
  - Normal map
- Using Perlin noise textures
- Using wave equations

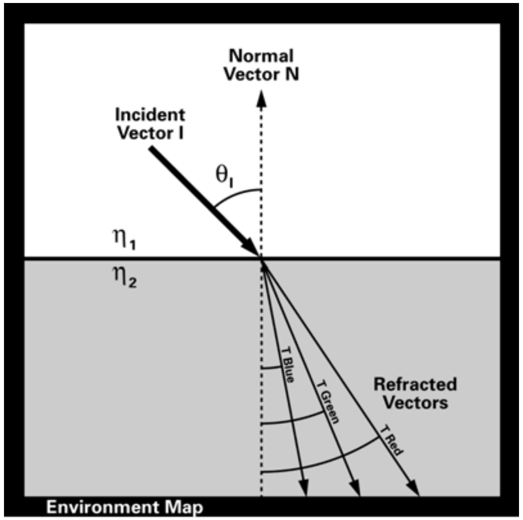


$$H(x,y,t) = \sum \left(A_i \times \sin\left(\mathbf{D}_i \cdot (x,y) \times w_i + t \times \varphi_i \right) \right),$$

nvidia.com

Refraction

- Given reflected ray, find color in environment map
- Given refracted ray, find color in environment map
- Color dispersion – compute refraction for each color channel separately
- Combine with Fresnel term



Refraction - Cg

http://developer.nvidia.com/CgTutorial/

```
void FS_reflection_refraction (float reflectionFactor : COLOR,
float3 R : TEXCOORD0,
float3 TRed : TEXCOORD1,
float3 TGreen : TEXCOORD2,
float3 TBlue : TEXCOORD3,
out float4 color : COLOR,
uniform samplerCUBE environmentMap0,
uniform samplerCUBE environmentMap1,
uniform samplerCUBE environmentMap2,
uniform samplerCUBE environmentMap3)
{
  // Fetch the reflected environment color
  float4 reflectedColor = texCUBE(environmentMap0, R);
  // Compute the refracted environment color
  float4 refractedColor;
  refractedColor.r = texCUBE(environmentMap1, TRed).r;
  refractedColor.g = texCUBE(environmentMap2, TGreen).g;
  refractedColor.b = texCUBE(environmentMap3, TBlue).b;
  refractedColor.a = 1;
  // Compute the final color
  color = lerp(refractedColor, reflectedColor, reflectionFactor);
}
```

```
void VS_reflection_refraction (float4 position : POSITION,
float3 normal : NORMAL,
out float4 oPosition : POSITION,
out float reflectionFactor : COLOR,
out float3 R : TEXCOORD0,
out float3 TRed : TEXCOORD1,
out float3 TGreen : TEXCOORD2,
out float3 TBlue : TEXCOORD3,
uniform float fresnelBias,
uniform float fresnelScale,
uniform float fresnelPower,
uniform float3 etaRatio,
uniform float3 eyePositionW,
uniform float4x4 modelViewProj,
uniform float4x4 modelToWorld)
{
  oPosition = mul(modelViewProj, position);
  // Compute position and normal in world space
  float3 positionW = mul(modelToWorld, position).xyz;
  float3 N = mul((float3x3)modelToWorld, normal);
  N = normalize(N);
  // Compute the incident, reflected, and refracted vectors
  float3 I = positionW – eyePositionW;
  R = reflect(I, N);
  I = normalize(I);
  TRed = refract(I, N, etaRatio.x);
  TGreen = refract(I, N, etaRatio.y);
  TBlue = refract(I, N, etaRatio.z);
  // Compute the reflection factor
  reflectionFactor = fresnelBias + fresnelScale * pow(1 + dot(I, N), fresnelPower);
}
```

Double refraction

- Chris Wyman, University of Iowa
- Approximate first intersection of refraction ray using depth buffer difference, similar for normal
- Use environment mapping in that intersection

$$\tilde{\mathbf{P}}_2 = \mathbf{P}_1 + \tilde{d}\vec{T}_1 \approx \mathbf{P}_1 + d\vec{T}_1$$

$$\tilde{d} = \frac{\theta_t}{\theta_i} d_{\vec{V}} + \left(1 - \frac{\theta_t}{\theta_i}\right) d_{\vec{N}}.$$

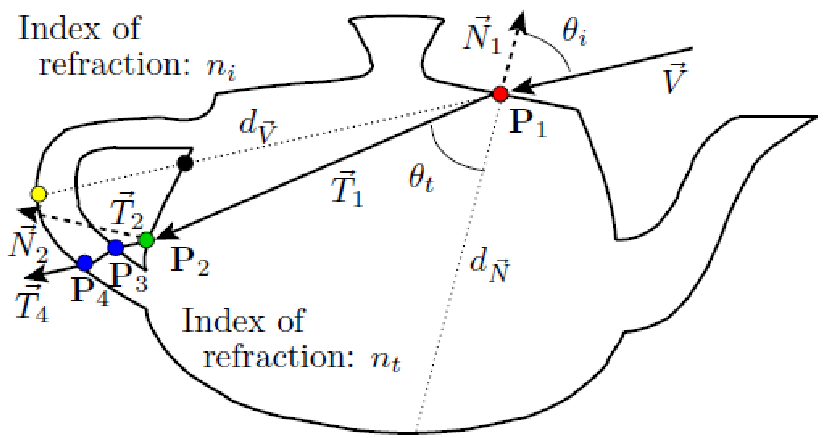


Figure 2: *Vector $\vec{V}$ hits the surface at $\mathbf{P}_1$ and refracts in direction $\vec{T}_1$ based upon the incident angle θ_i with the normal $\vec{N}_1$. Physically accurate computations lead to further refractions at $\mathbf{P}_2$, $\mathbf{P}_3$, and $\mathbf{P}_4$. Our method only refracts twice, approximating the location of $\mathbf{P}_2$ using distances $d_{\vec{N}}$ and $d_{\vec{V}}$.*

Double refraction

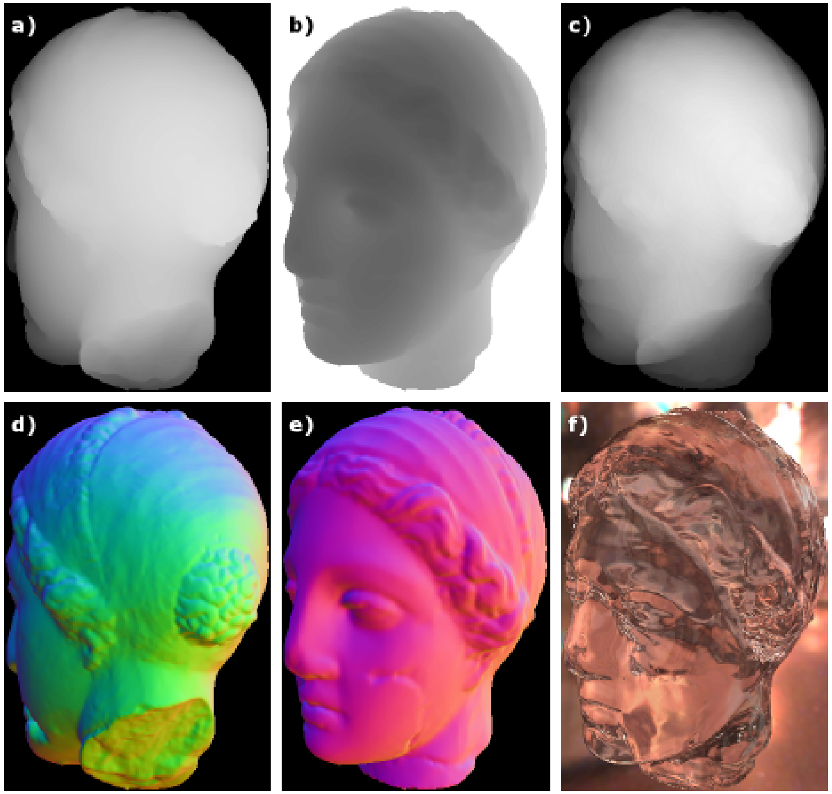


Figure 3: *Distance to back faces (a), to front faces (b), and between front and back faces (c). Normals at back faces (d) and front faces (e). The final result (f).*

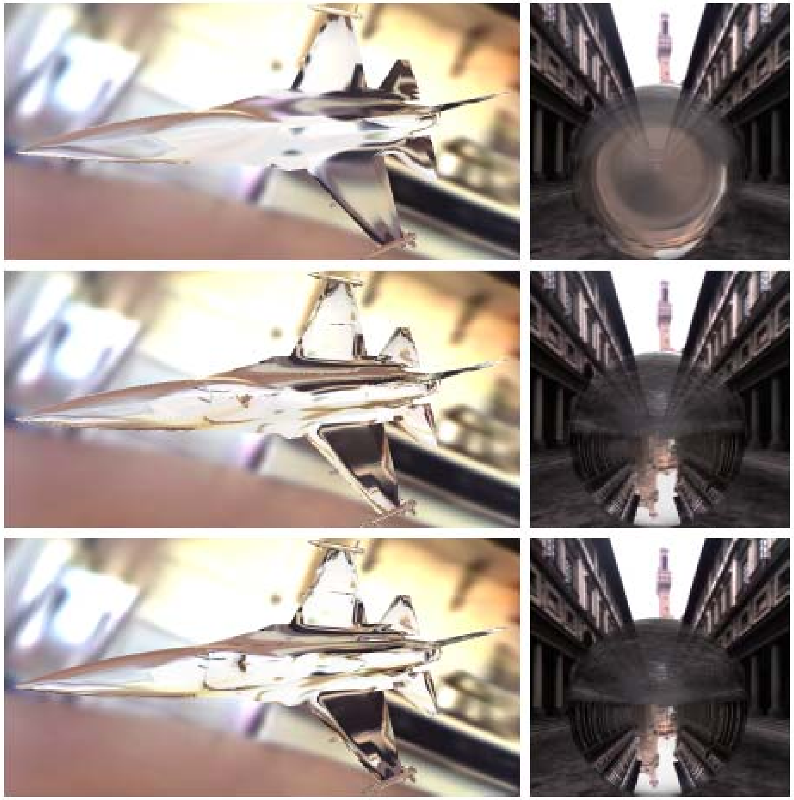
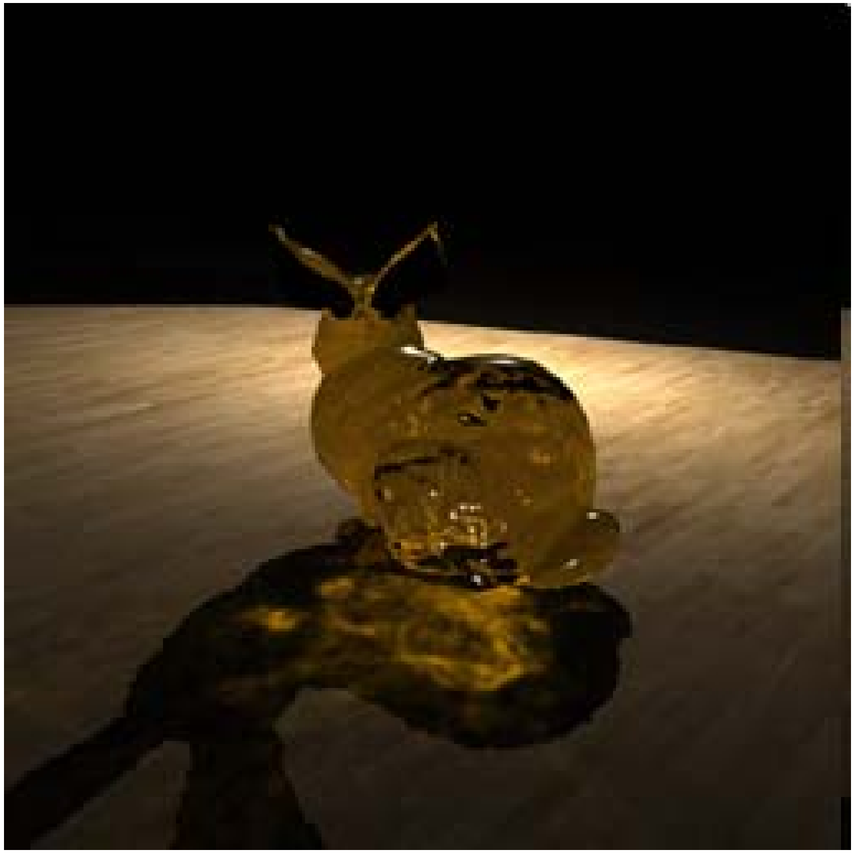


Figure 6: *Refraction through only one interface (top), using our technique (center), and ray traced (bottom). The indices of refraction are 1.2 for the jet and 1.5 for the ball.*

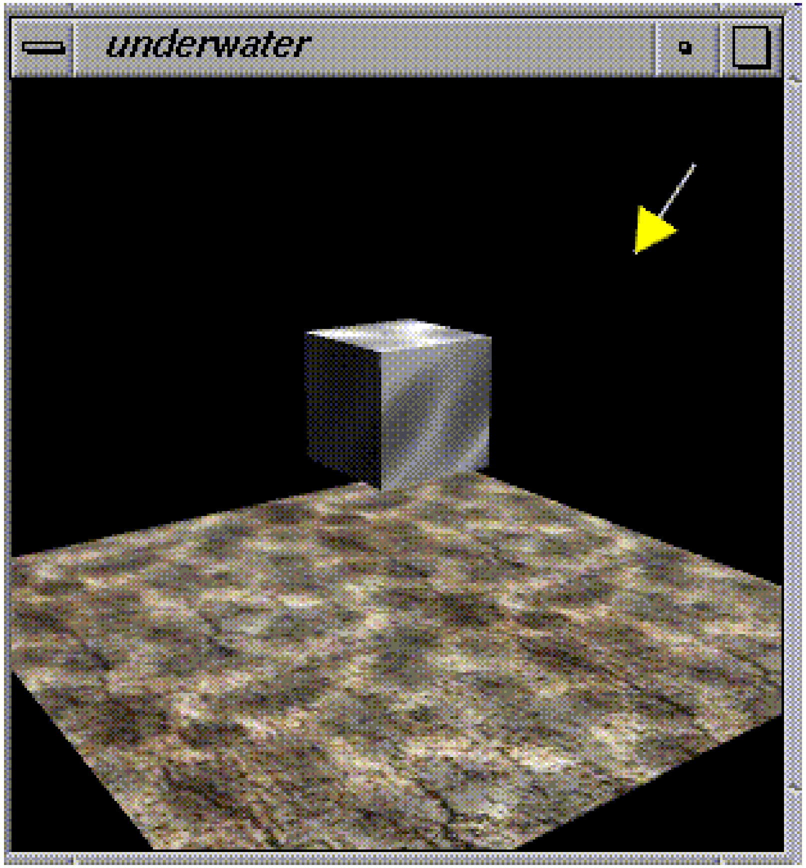
Other effects

- Subsurface scattering
  - Simulates scattering of refracted rays by material and exits the surface at a different point
  - Skin, wax, marble, jade, water, …
- Caustics
  - Envelope of light rays reflected or refracted by a curved surface or object
  - Produced by glass, water, …
- Sky & atmosphere
  - Sky boxes, sky dome, …
  - Simulating scattering of light rays in atmosphere

Caustics



http://graphics.cs.ucf.edu/caustics/



http://www.opengl.org/

Subsurface scattering

http://machinesdontcare.wordpress.com/2008/10/29/subsurface-scatter-shader/
http://eraser85.wordpress.com/2008/04/04/make-it-translucid-part-three/

Sources

- http://www.pages.drexel.edu/~pjd37/
- http://fileadmin.cs.lth.se/graphics/theses/projects/projgrid/projgrid-lq.pdf
- http://http.developer.nvidia.com/GPUGems/gpugems_ch01.html
- http://www.valvesoftware.com/publications/2004/GDC2004_Half-Life2_Shading.pdf
- http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node156.html
- http://www.humus.name/index.php?page=3D
- http://www.cs.uiowa.edu/~cwyman/pubs.html
- http://www.ozone3d.net/tutorials/
- http://www.pearsonhighered.com/assets/hip/us/hip_us_pearsonhighered/samplechapter/0321194969.pdf
- http://www.vterrain.org/Water/
- http://www.vterrain.org/Atmosphere/
- http://developer.nvidia.com/attach/6828
- http://eraser85.wordpress.com/category/opengl/

Questions?