GPU Programming for Video Games

Environment Mapping

Prof. Aaron Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

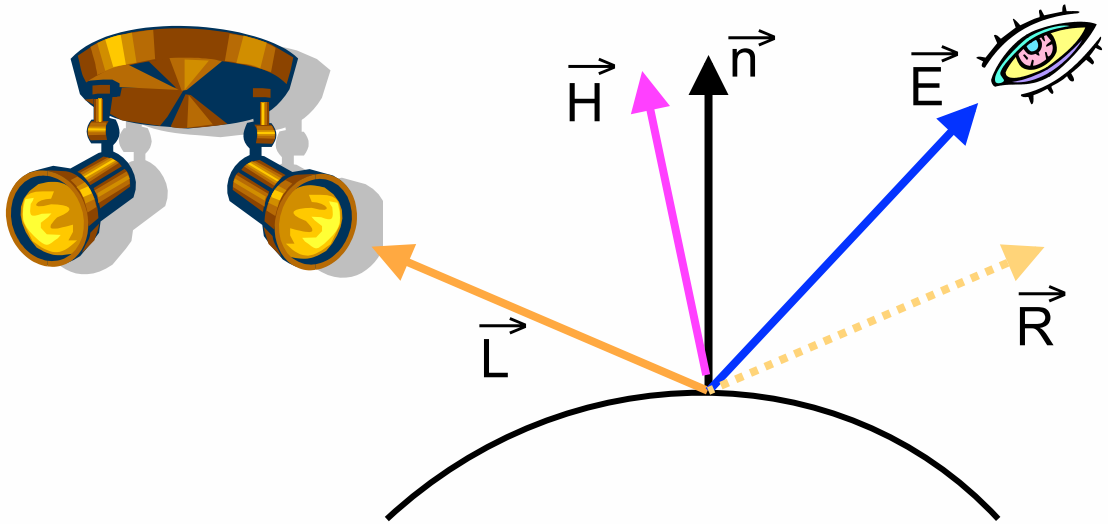
Georgia Institute of Technology

...of course, of course...

Using model & animation by Tom Higgens:
http://www.dootsy.co.uk

Georgia Institute of Technology

Specular models



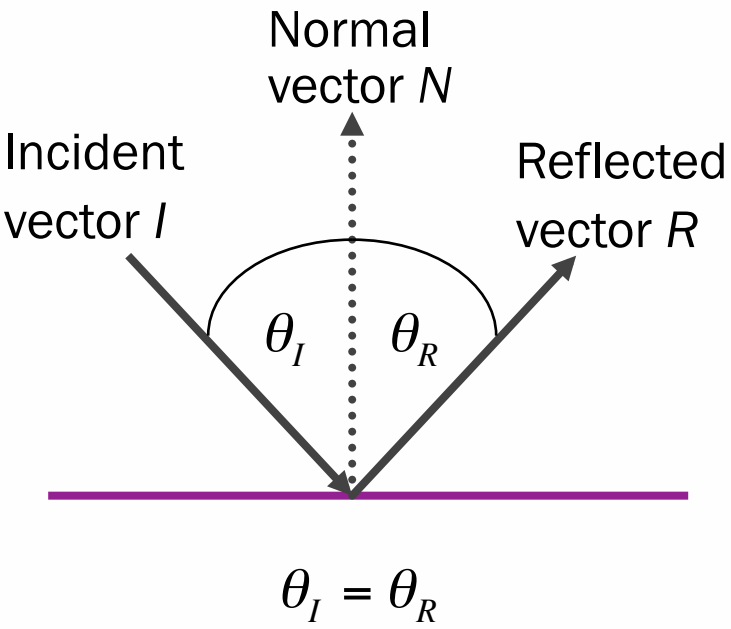
$$\vec{H} = \frac{\vec{E} + \vec{L}}{\left|\vec{E} + \vec{L}\right|}$$

Half-way vector

- Blinn-Phong model: $(\max(\vec{n} \bullet \vec{H}, 0))^S$
- Original Phong model: $(\max(\vec{R} \bullet \vec{E}, 0))^S$

Drawing by Hsien-Hsin Sean Lee

Reflection



$$\theta_I = \theta_R$$

```
R=reflect(I,N)
```

- N needs to be normalized
- R has same length as I

$$\vec{R} = \vec{I} - 2\vec{N}(\vec{N} \bullet \vec{I})$$

Georgia Institute of Technology

Compare models - setup

```
Shader "GPUXX/SpecdModelComp" {
  Properties {
    _BlinnPhongShininess ("Blinn-Phong Shininess", Range(0.01,1)) = 0.7
    _OriginalPhongShininess ("Original Phong Shininess", Range(0.01,1)) = 0.2
  }

                . . .


  uniform float _BlinnPhongShininess;
  uniform float _OriginalPhongShininess;

  struct a2v {
    float4 v: POSITION;
    float3 n: NORMAL;
  };

  struct v2f {
    float4 sv: SV_POSITION;
    float3 vWorldPos: TEXCOORD1;
    float3 nWorld: TEXCOORD2;
  };
```

Georgia Institute of Technology

Compare models – vertex program

```
v2f vert_specmodelcomp(a2v input) {
  v2f output;
  output.sv = mul(UNITY_MATRIX_MVP, input.v);
  // To transform normals, we want to use the inverse transpose
  // of upper left 3x3
  // Putting input.n in first argument is like
  // doing trans((float3x3)_World2Object) * input.n;
  output.vWorldPos = mul(_Object2World, input.v).xyz;
  output.nWorld = normalize(mul(input.n, (float3x3) _World2Object));
  return output;
}
```

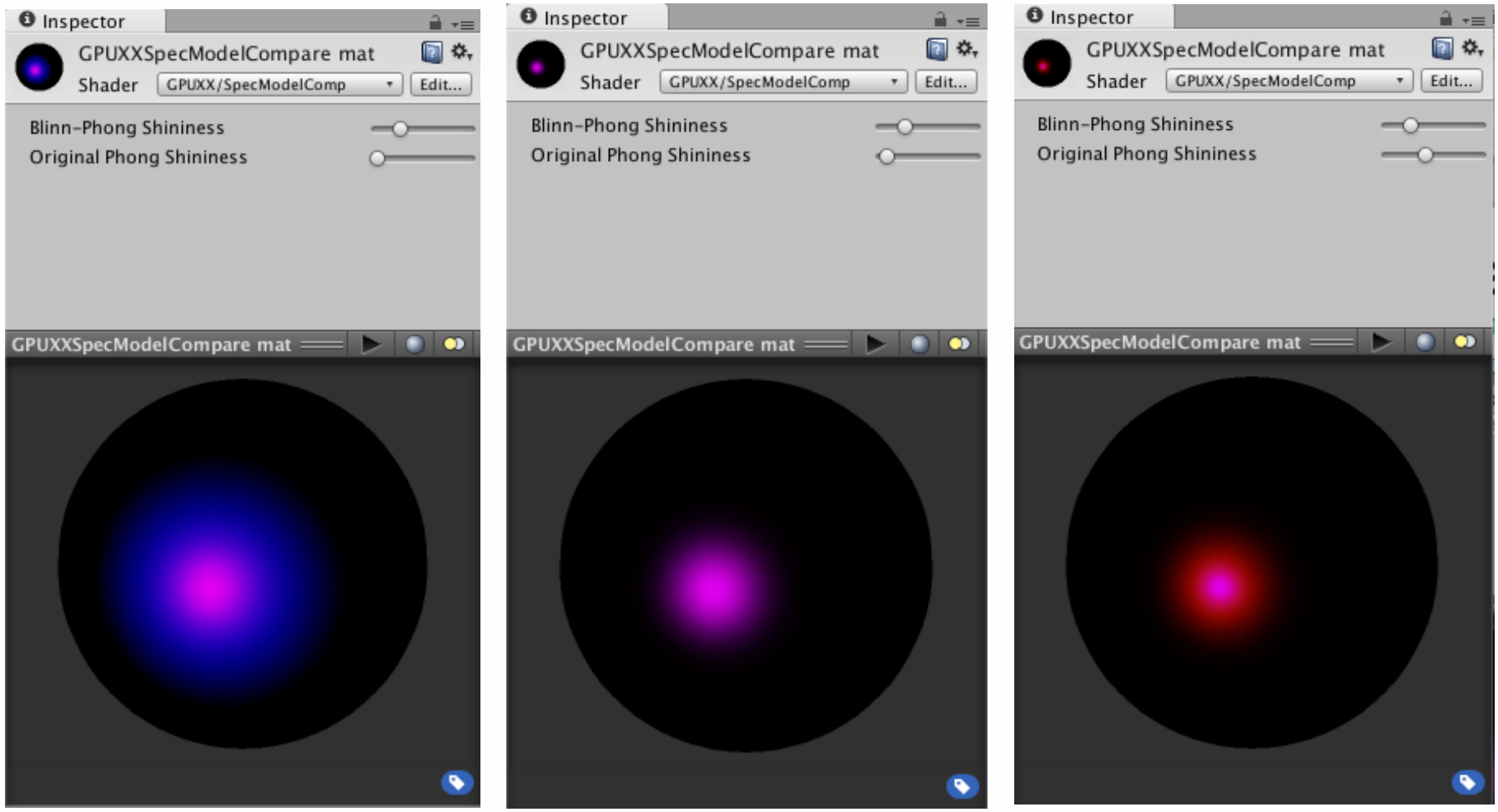
Georgia Institute of Technology

Compare models – fragment program

```
float4 frag_specmodelcomp(v2f input) : COLOR {
  // Weird Unity light convention
  float3 lightDir = normalize(_WorldSpaceLightPos0.xyz
                              - input.vWorldPos * _WorldSpaceLightPos0.w);
  float3 eyeDir = normalize(_WorldSpaceCameraPos.xyz - input.vWorldPos);
  float3 n = normalize(input.nWorld);
  float3 h = normalize(lightDir + eyeDir);
  // because reflect uses an "incident vector" we need a minus sign
  float3 r = normalize(reflect(-lightDir,n));

  float ndoth = max(0, dot(n, h));
  float rdoteye = max(0, dot(r, eyeDir));
  // colorcoding "Blinn-Phong as red" and "Phong Original" as blue;
  // to match Shininess settings, adjust until you see only purple
  float3 spec_blinnphong_cc = float3(0.9,0,0)
                              * pow(ndoth, _BlinnPhongShininess*128.0);
  float3 spec_phongorig_cc = float3(0,0,0.9)
                              * pow(rdoteye, _OriginalPhongShininess*128.0);
  float3 output = spec_blinnphong_cc + spec_phongorig_cc;
  return(float4(output,1));
}
```

Georgia Institute of Technology

Comparing both models on a sphere

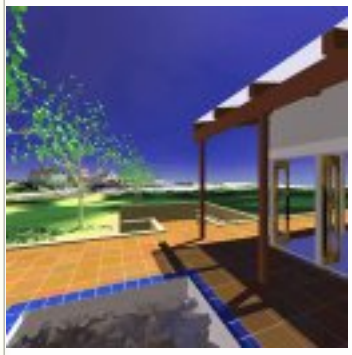


Matching spec powers

Horse model & animation by Dootsy Development

Cube maps

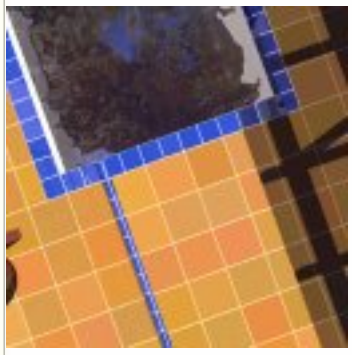
Back



Front

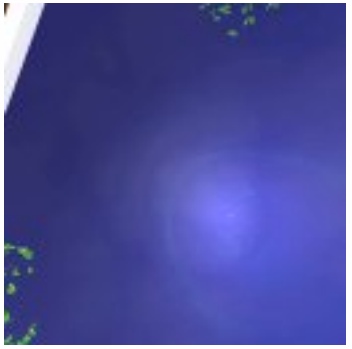
Right

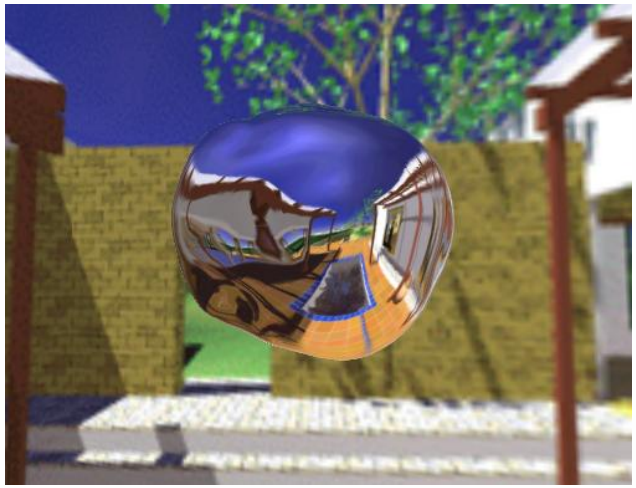
Bottom

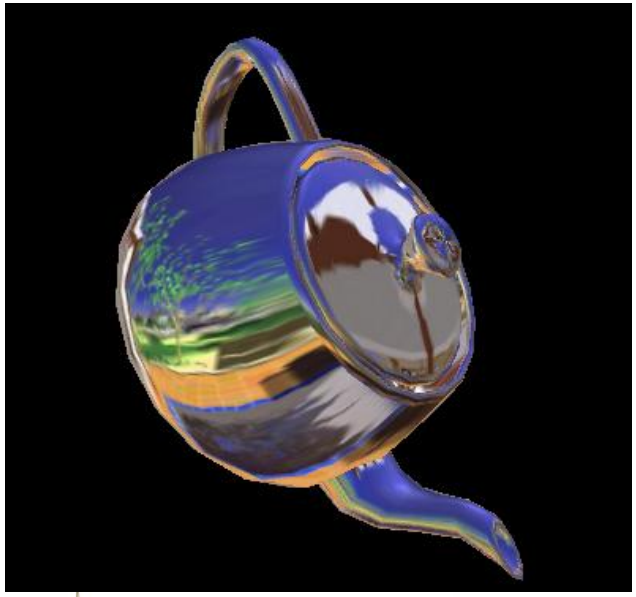


Left

Top





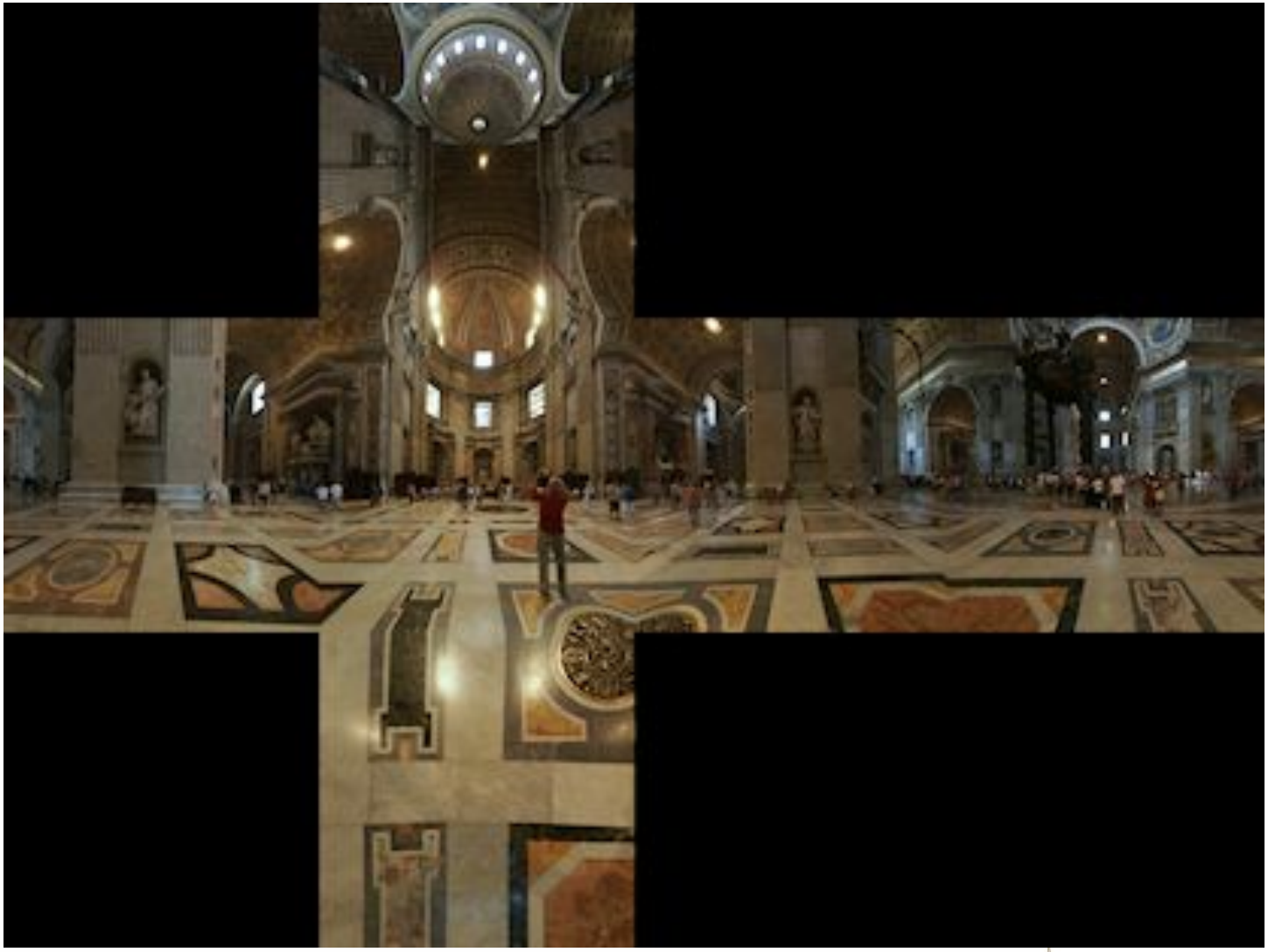


texCUBE`((samplerCUBE) envMap,(float3) vec)`

Images from "OpenGL Cube Map Texturing,"
developer.nvidia.com/object/cube_map_ogl_tutorial.html

Georgia Institute of Technology

St. Peter's Basilica

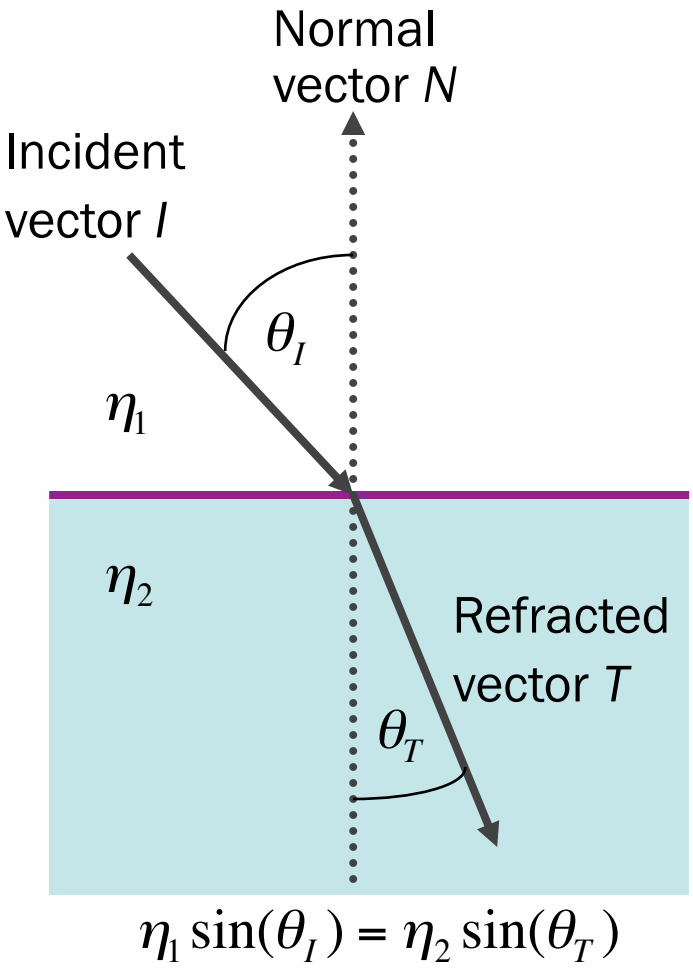


Cube Map by Emil Persson, aka "Humus"

From http://www.humus.name/index.php?page=Textures&ID=106

Georgia Institute of Technology

Refraction




```
T=refract(I,N,etaRatio)
```

etaRatio=η_1 / η_2

Georgia Institute of Technology

Different indices of refraction

Vacuum: 1.0
Air: 1.0003
Water: 1.333
Glass: 1.5 (ordinary window glass)
Plastic: 1.5
Diamond: 2.417

Data from "The Cg Tutorial," p. 184

Images from Thomas Kerwin, "Refraction in OpenGL,"
www.cse.ohio-state.edu/~kerwin/refraction.html

EnvMap (per vertex) – setup

```
Shader "GPUXX/EnvMapPerVertex" {
  Properties {
    _Cube ("Reflection Cubemap", CUBE) = "white" {}
    _etaRatio("Eta Ratio", Range(0.01,3)) = 1.5
    _crossfade("Crossfade", Range(0,1)) = 0
  }
          . . .

  uniform samplerCube _Cube;
  uniform float _etaRatio;
  uniform float _crossfade;

  struct a2v {
    float4 v: POSITION;
    float3 n: NORMAL;
  };

  struct v2f {
    float4 sv: SV_POSITION;
    float3 reflectWorld: TEXCOORD0;
    float3 refractWorld: TEXCOORD1;
  };
```

Georgia Institute of Technology

EnvMap (per vertex) – vertex program

```
v2f vert_envmappervertex(a2v input) {
  v2f output;
  output.sv = mul(UNITY_MATRIX_MVP, input.v);
  // To transform normals, we want to use the inverse transpose
  // of upper left 3x3
  // Putting input.n in first argument is like doing
  // trans((float3x3)_World2Object) * input.n;
  float3 vWorldPos = mul(_Object2World, input.v).xyz;
  float3 nWorld = normalize(mul(input.m, (float3x3) _World2Object));
  // incident is opposite the direction of eyeDir in our other programs
  float3 incidentWorld = normalize (vWorldPos
                                    - _WorldSpaceCameraPos.xyz);
  output.reflectWorld = reflect(incidentWorld, nWorld);
  output.refractWorld = refract(incidentWorld, nWorld,_etaRatio);
  return output;
}
```

Georgia Institute of Technology

EnvMap (per vertex) – fragment program

```
float4 frag_envmappervertex(v2f input) : COLOR {
  float4 reflectColor = texCUBE(_Cube, input.reflectWorld);
  float4 refractColor = texCUBE(_Cube, input.refractWorld);
  return(lerp(reflectColor, refractColor, _crossfade));
}
```

Artifacts of per-vertex computation:

Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Georgia Institute of Technology

Fresnel effect

- Some light reflects and some refracts
- Think about looking into water
  - At shallow angles, a lot of reflection and little refraction
  - Looking straight in, a lot of refraction and a little reflection
- Empirical approximation: $\vec{I} = -\vec{E}$

$$reflectCoeff = \max(0, \min(1, bias + scale(1 + \vec{I} \bullet \vec{n})^{power}))$$

$$C_{final} = reflectCoeff \times C_{reflected} + (1 - reflectCoeff) C_{refracted}$$

From "The Cg Tutorial," p. 189

EnvMap (per pixel) – setup

```
Shader "GPUXX/EnvMapPerPixel" {
  Properties {
    _Cube ("Reflection Cubemap", CUBE) = "white" {}
    _etaRatio("Eta Ratio", Range(0.01,3)) = 1.5
    _crossfade("Crossfade", Range(0,1)) = 0
    _fresnelBias ("Fresnel Bias", Range(0,1)) = 0.5
    _fresnelScale ("Fresnel Scale", Range(0,1)) = 0.5
    _fresnelPower ("Fresnel Power", Range(0,10)) = 0.5
  }

            ...

  uniform samplerCube _Cube;
  uniform float _etaRatio;
  uniform float _crossfade;
  uniform float _fresnelBias
  uniform float _fresnelScale
  uniform float _fresnelPower
```

Georgia Institute of Technology

EnvMap (per pixel) – structures

```
struct a2v {
    float4 v: POSITION;
    float3 n: NORMAL;
};

struct v2f {
    float4 sv: SV_POSITION;
    float3 nWorld: TEXCOORD0;
    float3 vWorldPos: TEXCOORD1;
};
```

Georgia Institute of Technology

EnvMap (per pixel) – vertex program

```
v2f vert_envmapperpixel(a2v input) {
  v2f output;
  output.sv = mul(UNITY_MATRIX_MVP, input.v);
  // To transform normals, we want to use
  // the inverse transpose of upper left 3x3
  // Putting input.n in first argument is like
  // doing trans((float3x3)_World2Object) * input.n;
  output.vWorldPos = mul(_Object2World, input.v).xyz;
  output.nWorld = normalize(mul(input.n,
                          (float3x3) _World2Object));
  return output;
}
```

Georgia Institute of Technology

EnvMap (per pixel) – fragment program

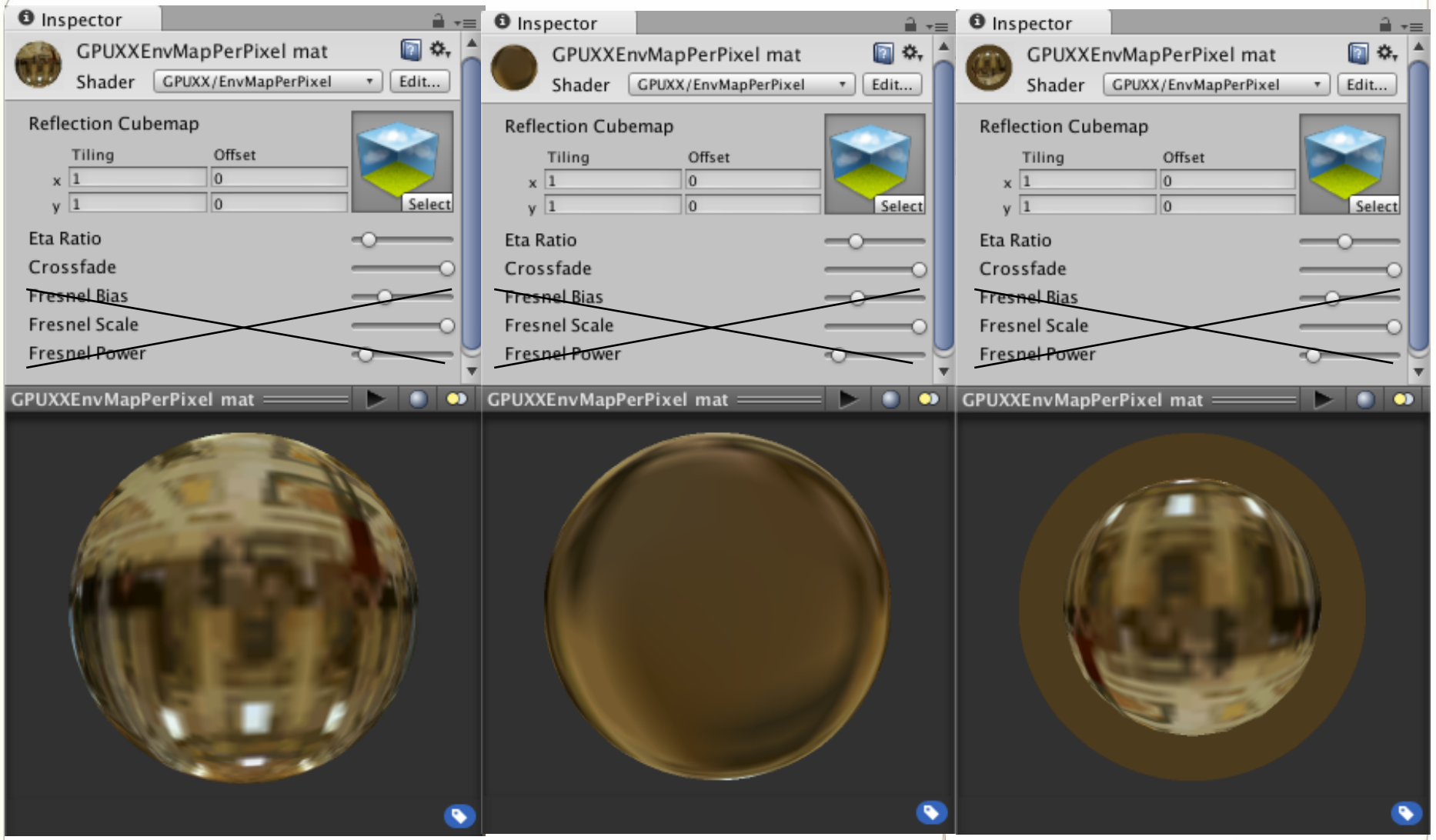
```
float4 frag_envmapperpixel(v2f input) : COLOR {
  // incident is opposite the direction of eyeDir in our other programs
  float3 incidentWorld = normalize(input.vWorldPos
                                   - _WorldSpaceCameraPos.xyz);
  float3 reflectWorld = reflect(incidentWorld, input.nWorld);
  float3 refractWorld = refract(incidentWorld, input.nWorld,_etaRatio);

  float4 reflectColor = texCUBE(_Cube, reflectWorld);
  float4 refractColor = texCUBE(_Cube, refractWorld);
  float reflectFactor = saturate(_fresnelBias + _fresnelScale *
          pow(1 + dot(incidentWorld, input.nWorld), _fresnelPower));

  // Diagnostic: uncomment next lines to code
  // refraction as yellow & reflection as blue
  // refractColor = float4(1,1,0,1);
  // reflectColor = float4(0,0,1,1);

  // Comment out one of the following two lines
  // and leave the other as desired
  // return(lerp(reflectColor, refractColor, _crossfade));
  return(lerp(refractColor, reflectColor, reflectFactor));
}
```

Georgia Institute of Technology

Pure reflection

Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Pure refraction



Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Georgia Institute of Technology

Low EtaRatio

Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

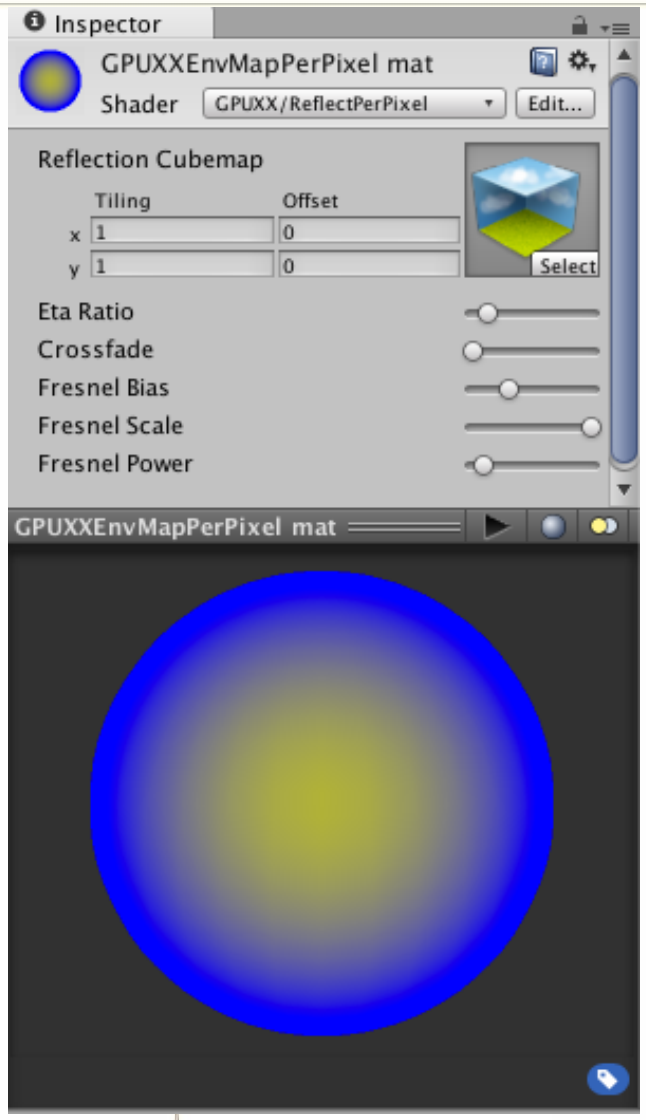
Medium EtaRatio

Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

High EtaRatio

Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Fresnel effect visualization



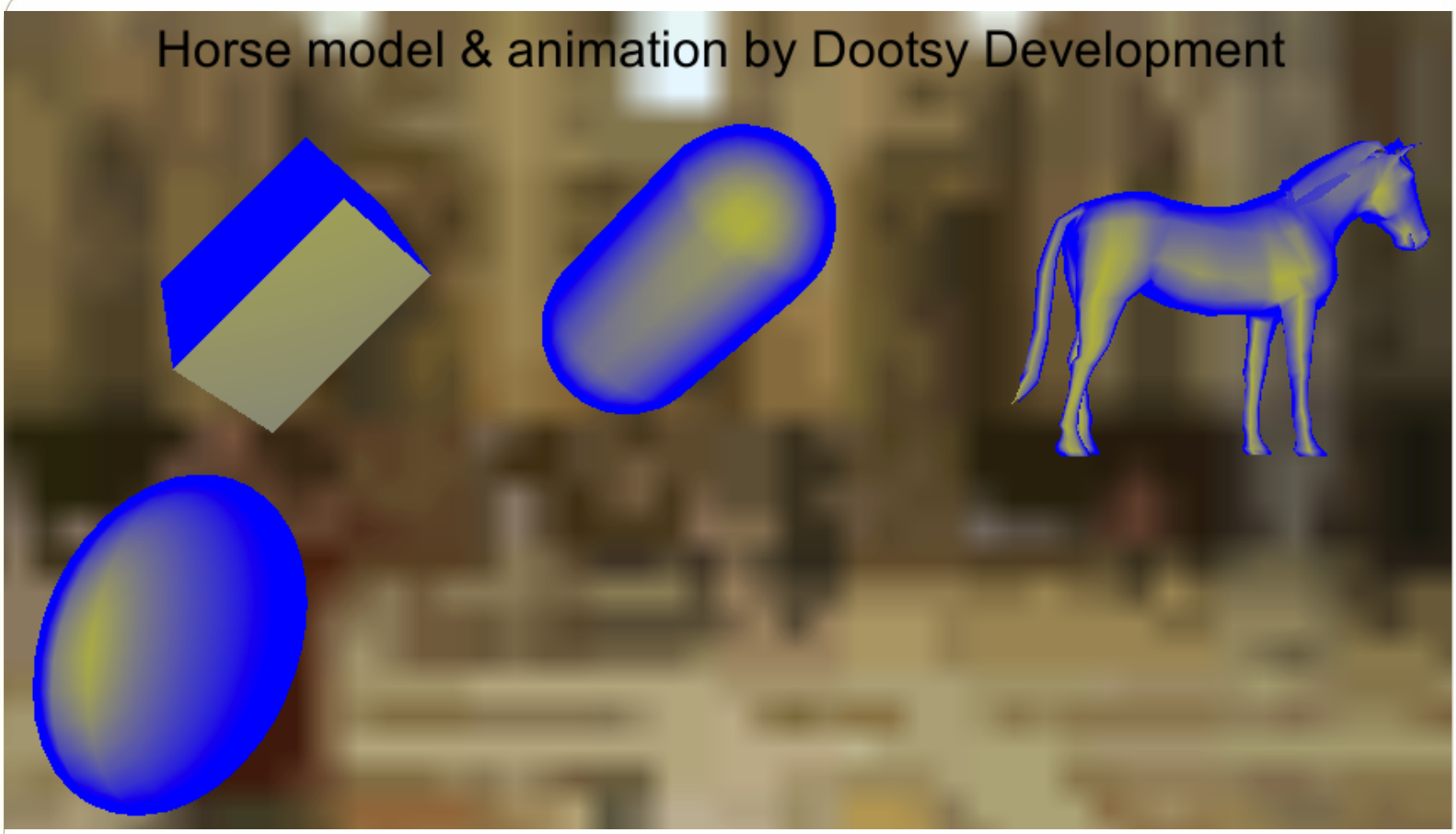
Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Georgia Institute of Technology

Fresnel effect

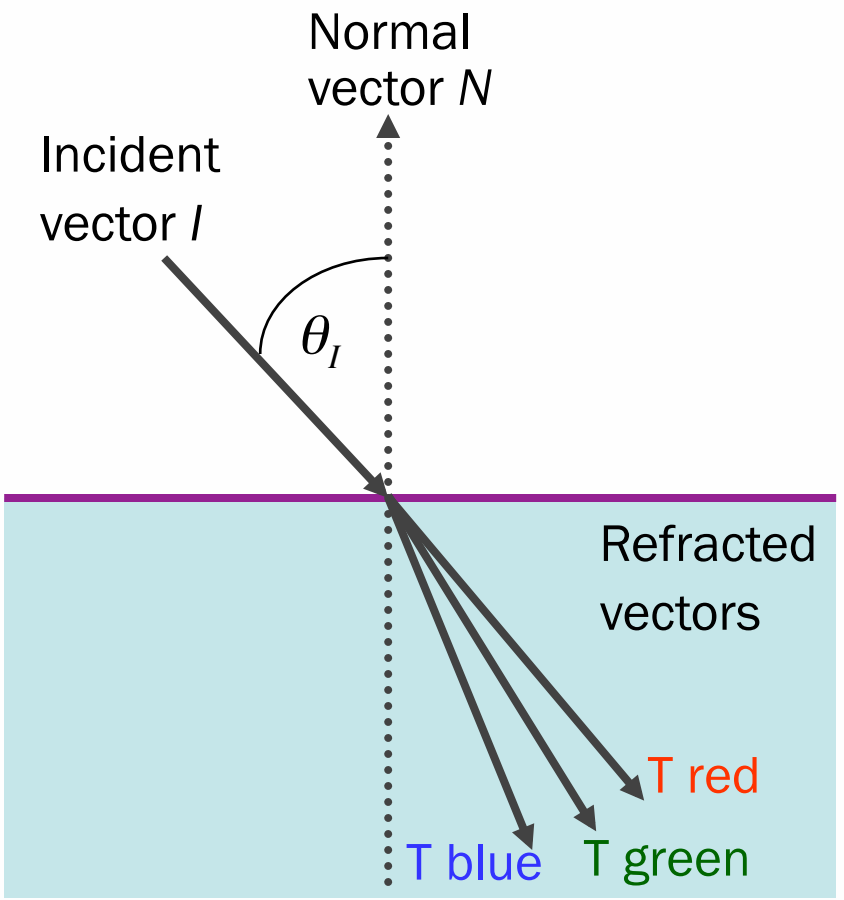
Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Fresnel effect (diagnostic)



Cube Map by Emil Persson, aka "Humus"
From http://www.humus.name/index.php?page=Textures&ID=106

Chromatic dispersion



Images from Thomas Kerwin, "Refraction in OpenGL,"
www.cse.ohio-state.edu/~kerwin/refraction.html