Real-time graphics has come a long way

- **Virtua Fighter** (SEGA Corporation)
  - NV1
  - 50K triangles/sec
  - 1M pixel ops/sec
  - 1995

- **Dead or Alive 3** (Tecmo Corporation)
  - Xbox (NV2A)
  - 100M triangles/sec
  - 1G pixel ops/sec
  - 2001

- **Dawn** (NVIDIA Corporation)
  - GeForce FX (NV30)
  - 200M triangles/sec
  - 2G pixel ops/sec
  - 2003

The Cg Tutorial

- Can get Cg Toolkit, example code, etc. here:

Nice framework for experimentation

Image from “Teaching Cg” Powerpoint presentation:
Cube maps

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

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

Reflection and refraction

Incident vector \( I \)
Reflected vector \( R \)
Reflected vector \( R = \text{reflect}(I, N) \)

Incident vector \( I \)
Normal vector \( N \)
Refraction \( \eta_1 \sin(\theta_1) = \eta_2 \sin(\theta_2) \)
Refracted vector \( T = \text{refract}(I, N, \text{etaRatio}) \)
etaRatio = \( \frac{\eta_1}{\eta_2} \)

Cg vertex shader for reflective mapping

```c
void C7E1v_reflection(float4 position : POSITION, float2 texCoord : TEXCOORD0, float3 normal : NORMAL, out float4 oPosition : POSITION, out float2 oTexCoord : TEXCOORD0, out float3 R : TEXCOORD1, uniform float3 eyePositionW, uniform float4x4 modelViewProj, uniform float4x4 modelToWorld) {
    oPosition = mul(modelViewProj, position);
    oTexCoord = texCoord;
// 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 and reflected vectors
    float3 I = positionW - eyePositionW;
    R = reflect(I, N);
}
```

From “The Cg Tutorial,” p. 177

Cg pixel shader for reflective mapping

```c
void C7E2f_reflection(float2 texCoord : TEXCOORD0, float3 R : TEXCOORD1, out float4 color : COLOR, uniform float reflectivity, uniform sampler2D decalMap, uniform samplerCUBE environmentMap) {
// Fetch reflected environment color
    float4 reflectedColor = texCUBE(environmentMap, R);
// Fetch the decal base color
    float4 decalColor = tex2D(decalMap, texCoord);
    color = lerp(decalColor, reflectedColor, reflectivity);
}
```

From “The Cg Tutorial,” p. 180
Different indices of refraction

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

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

Cg vertex shader for refractive mapping

```cpp
void C7E3v_refraction(float4 position : POSITION,
float2 texCoord : TEXCOORD0, float3 normal : NORMAL,
out float4 oPosition : POSITION,
out float2 oTexCoord : TEXCOORD0,
out float3 T : TEXCOORD1,
uniform float etaRatio,
uniform float3 eyePositionW,
uniform float4x4 modelViewProj, uniform float4x4 modelToWorld)
{
oPosition = mul(modelViewProj, position);
tTexCoord = texCoord;
// 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 and refracted vectors
float3 I = positionW - eyePositionW;
T = refract(I, N, etaRatio);
}
```

From “The Cg Tutorial,” p. 184

Cg pixel shader for refractive mapping

```cpp
void C7E4f_refraction(float2 texCoord : TEXCOORD0,
float3 T : TEXCOORD1,
out float4 color : COLOR,
uniform float transmittance,
uniform sampler2D decalMap,
uniform samplerCUBE environmentMap)
{
// Fetch the decal base color
float4 decalColor = tex2D(decalMap, texCoord);
// Fetch refracted environment color
float4 refractedColor = texCUBE(environmentMap, T);
// Compute the final color
color = lerp(decalColor, refractedColor, transmittance);
}
```

From “The Cg Tutorial,” p. 188

Chromatic dispersion

Without CD
- Incident vector I
- Normal vector N
- Refracted vectors

With CD
- T blue
- T red
- T green

Images from Thomas Kerwin, “Refraction in OpenGL,” www.cse.ohio-state.edu/~kerwin/refraction.html
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:
  \[ \text{reflectCoeff} = \max(0, \min(1, \text{bias} + \text{scale}(1 + I \cdot N)\text{power})) \]
  \[ C_{\text{final}} = \text{reflectCoeff} \times C_{\text{reflected}} + (1 - \text{reflectCoeff})C_{\text{refracted}} \]

From “The Cg Tutorial,” p. 189

Bump mapping

- 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:
  \[ \text{reflectCoeff} = \max(0, \min(1, \text{bias} + \text{scale}(1 + I \cdot N)\text{power})) \]
  \[ C_{\text{final}} = \text{reflectCoeff} \times C_{\text{reflected}} + (1 - \text{reflectCoeff})C_{\text{refracted}} \]

From “The Cg Tutorial,” p. 189

Bump mapping examples

- Top row from Wikipedia entry on “bump mapping”
- Bottom row from Søren Dreijer, “Bump Mapping Using Cg (3rd Edition),”

Images from Paul Baker, “Simple Bumpmapping,”
www.paulsprojects.net/tutorials/simplebump/simplebump.html

Shader effect movies

- Bump mapping demo with the Cimg library
  http://video.google.com/videoplay?docid=1570416667092534064

- Bump mapping and reflective textures
  - (HLEH - Half Life mod???)
  http://www.youtube.com/watch?v=FmpyHc6hXc4

- Bump mapping on the Nintendo DS
  http://www.youtube.com/watch?v=6ypt5Jf-eof
Storing normals in textures

- Textures don’t have to store color; we can store other things as well, like normals
  - Use r, g, b components to store, x, y, z of normal
- Problem: Textures take [0,1] values; normals need [-1,1] values
- Easy solution: “Range Compression”

\[
\text{colorComponent} = 0.5 \times \text{normalComponent} + 0.5;
\]

\[
\text{normalComponent} = 2 \times (\text{colorComponent} - 0.5);
\]

Creating normal map from height field

- Height field \(H(u,v)\)

\[
\text{normal} = \begin{pmatrix}
H_z - H_r
\end{pmatrix}
\begin{pmatrix}
H_y - H_r
\end{pmatrix}
\begin{pmatrix}
H_y - H_r
\end{pmatrix}
\]

- In flat regions, normal is \((0,0,1)\), i.e. pointing “up”

Cg vertex shader for bump mapping

```cpp
void C8E1v_bumpWall(float4 position : POSITION,
                     float2 texCoord : TEXCOORD0,
                     out float4 oPosition : POSITION,
                     out float2 oTexCoord : TEXCOORD0,
                     out float3 lightDirection : TEXCOORD1,
                     uniform float3 lightPosition, // Object space
                     uniform float4x4 modelViewProj)
{
    oPosition = mul(modelViewProj, position);
    oTexCoord = texCoord;
    // Difference vectors for object-space light direction
    lightDirection = lightPosition - position.xyz;
}
```

Cg pixel shader for bump mapping

```cpp
float3 expand(float3 v) { return (v-0.5)*2; }

void C8E2f_bumpSurf(float2 normalMapTexCoord : TEXCOORD0,
                    float3 lightDir : TEXCOORD1,
                    out float4 color : COLOR,
                    uniform sampler2D normalMap,
                    uniform samplerCUBE normalizeCube)
{
    float3 lightTex = texCUBE(normalizeCube, lightDir).xyz;
    float3 light = expand(lightTex);
    // Sample and expand the normal map texture
    float3 normalTex = tex2D(normalMap, normalMapTexCoord).xyz;
    float3 normal = expand(normalTex);
    // Diffuse lighting
    color = dot(normal, light);
}
```

From “The Cg Tutorial,” p. 202

From “The Cg Tutorial,” p. 203

From “The Cg Tutorial,” p. 205

From “The Cg Tutorial,” p. 206