Shaders and Visual Realism

"Toy Story" - Pixar, 1995 – Using the Pixar Renderman Language
(Programmable Photorealistic Offine Rendering)

Part I

VISUAL REALISM IN GAMES
Part II

**SHADER PROGRAMMING**

Most of the following images were taken from
"Shaders for Game Programmers and Artists" by Sébastien St-Laurent,
Copyright 2004 by Course Technology.
Polygons (Triangles)

Vertices

Vertex Transformation
Translating and Scaling

Object vertices in Object coordinates: $T_{\text{local}}$
**World Coordinates**

Object vertices in World Space coordinates:

\[ \mathbf{T}_{\text{world}} \cdot \mathbf{T}_{\text{local}} \]

**Camera Coordinates**

Object vertices in Camera Space coordinates:

\[ \mathbf{T}_{\text{camera}}^{-1} \cdot \mathbf{T}_{\text{world}} \cdot \mathbf{T}_{\text{local}} \]

**Face Culling and Clipping**

Culled if facing away

Clipped if outside frustum
View Projection

Object vertices in Screen Space coordinates:
Projection Matrix \( \cdot T^{-1}_{\text{camera}} \cdot T_{\text{world}} \cdot T_{\text{local}} \)

Rasterization

Fragment Coloring and Blending

Coloring of fragments based on interpolated information from nearby vertices (e.g., Vertex colors, Vertex UV coordinates, Vertex Normals)

Alpha Blending and Depth Tests

Z-buffer keeps track of the front most fragments
Pixels (what we see on the screen)

PC 3D Rendering in Hardware

<table>
<thead>
<tr>
<th>Year</th>
<th>Graphics Card</th>
<th>Milestones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1987</td>
<td>IBM VGA</td>
<td>Provides a pixel frame buffer that the CPU has to fill</td>
</tr>
<tr>
<td>1996</td>
<td>3dfx Voodoo</td>
<td>Rasterizes and textures pre-transformed vertices (triangles)</td>
</tr>
<tr>
<td>1999</td>
<td>nVidia GeForce 256</td>
<td>Applies both transformation and lighting to vertices (T&amp;L) – fixed pipeline</td>
</tr>
<tr>
<td>2001</td>
<td>nVidia GeForce 3</td>
<td>Configurable pixel shader and programmable vertex shader</td>
</tr>
<tr>
<td>2003</td>
<td>nVidia GeForce FX</td>
<td>Fully programmable pixel and vertex shaders</td>
</tr>
</tbody>
</table>
// Simplest Vertex Shader
// input vertex
struct VertIn {
  float4 pos   : POSITION;
  float4 color : COLOR0;
};
// output vertex
struct VertOut {
  float4 pos   : POSITION;
  float4 color : COLOR0;
};
// vertex shader main entry
VertOut main(VertIn IN, uniform float4x4 modelViewProj) {
  VertOut OUT;
  OUT.pos = mul(modelViewProj, IN.pos); // calculate output coords
  OUT.color = IN.color; // copy input color to output
  return OUT;
}
**Pixel Shader**

// Small Pixel Shader (Grayscale Converter)
// Input pixel
struct PixIn {
float3 color : COLOR0;
float3 texcoord : TEXCOORD0;
};
// output pixel
struct PixOut {
float3 color : COLOR0;
};
// vertex shader main entry
PixOut main(PixIn IN, uniform sampler2D texture : TEXUNIT0) {
PixOut OUT;
float3 color = tex2D(texture, IN.texcoord).rgb;
OUT.color = dot(color, float3(0.299, 0.587, 0.114)).xxx;
return OUT;
}

**Categories of Shader Programs**

- Vertex Skinning
- Vertex Displacement Mapping
- Screen Effects
- Light and Surface Models
- Non-photorealistic Rendering

**Scene Effects**

- Pixel shader renders to a temporary texture that it then processes with filters before returning the color values.
Scene Effects: Glow

Scene Effects: Depth of Field

Scene Effects: Distortion
Screen Effects: High Dynamic Range + Bloom

Lighting Models

- Shaders calculate new color values by applying various lighting models, involving parameters such as surface normals (N), light angle (L), reflected light angle (R) and view angle (V).

Lighting Models: Per-Pixel Lighting
Non-Photorealistic Rendering