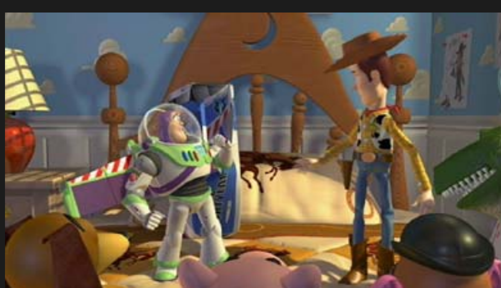
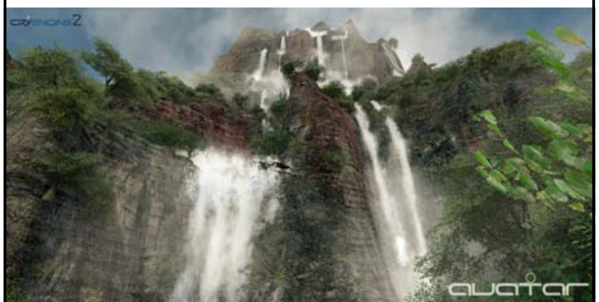
Shaders and Visual Realism



"Toy Story" - Pixar, 1995 – Using the Pixar RenderMan Language
(Programmable Photorealistic Off-line Rendering)

Part I

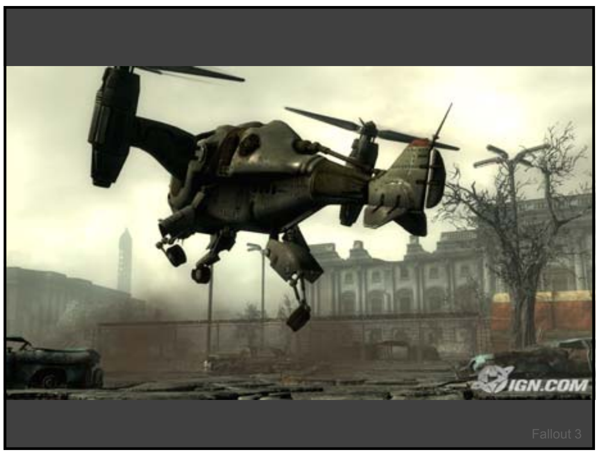
VISUAL REALISM IN GAMES



Codemasters: Dirt 2

Codemasters: Dirt 2

Fallout 3

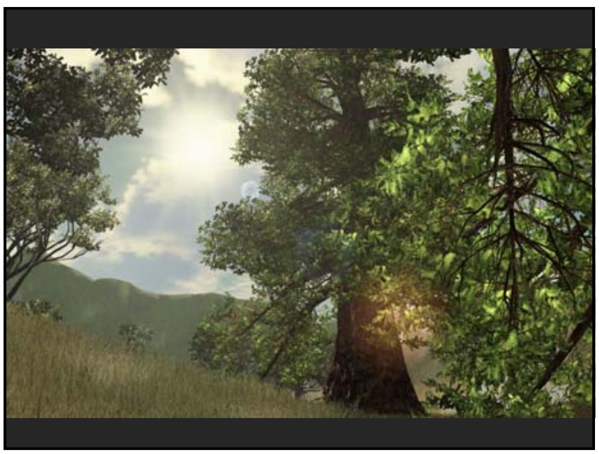


Fallout 3

"Bioshock"


"Unreal Engine 3.0"


POWERED BY
UNREAL
TECHNOLOGY
speedtree



“Alan Wake”

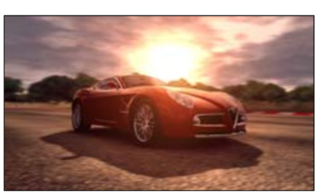
"Alan Wake"

"Alan Wake"

"Alan Wake"

"Unreal Tournament 2007"

"Crysis"



"Test Drive Unlimited"

"Test Drive Unlimited"

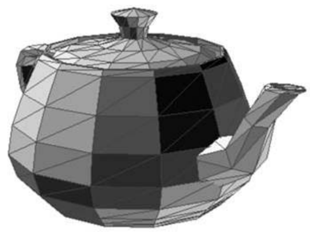
Part II

SHADER PROGRAMMING

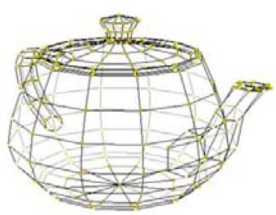
Most of the following images were taken from
"Shaders for Game Programmers and Artists" by Sebastien St-Laurent,

3D Object (Shaded and colored)

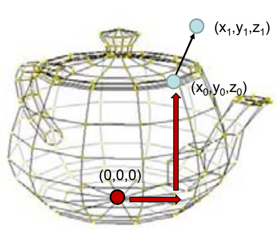
Polygons (Triangles)



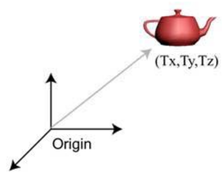
Vertices

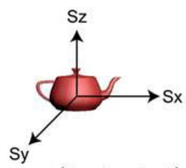


Vertex Transformation



Translating and Scaling

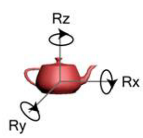




$$\text{Translation} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ Tx & Ty & Tz & 0 \end{pmatrix}$$

$$\text{Scale} = \begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Rotating

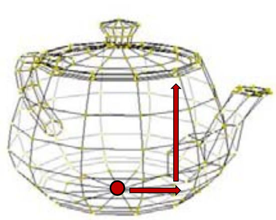


$$\text{Rx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{Ry} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rz} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

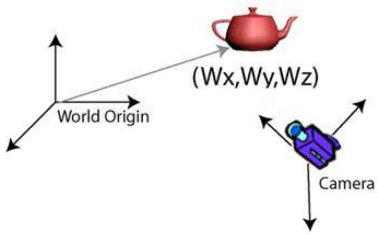
Vertex Transformation



Object vertices in Object coordinates:

T_{local}

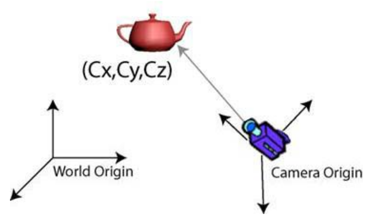
World Coordinates



Object vertices in World Space coordinates:

$T_{world} \bullet T_{local}$

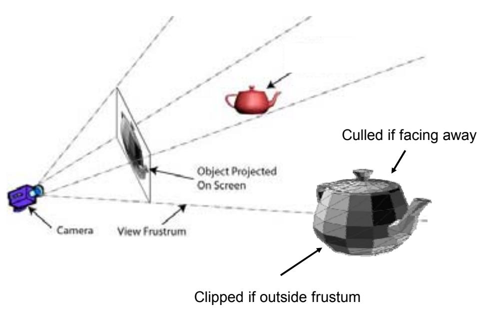
Camera Coordinates



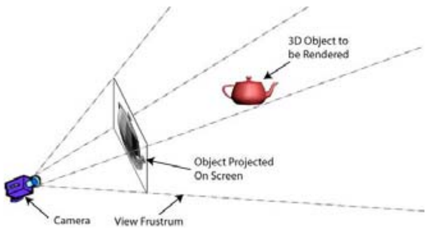
Object vertices in Camera Space coordinates:

$T^{-1}_{camera} \bullet T_{world} \bullet T_{local}$

Face Culling and Clipping



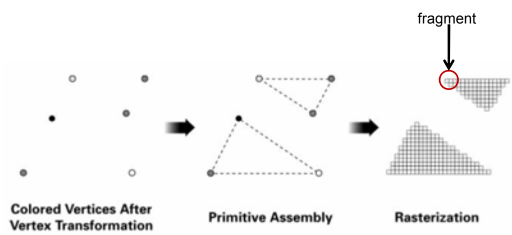
View Projection



Object vertices in Screen Space coordinates:

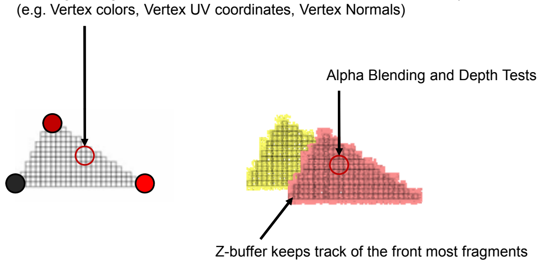
Projection Matrix • T^{-1}_{camera} • T_{world} • T_{local}

Rasterization



Fragment Coloring and Blending

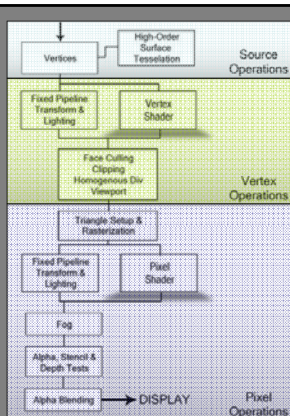
Coloring of fragments based on interpolated information from nearby vertices (e.g. Vertex colors, Vertex UV coordinates, Vertex Normals)

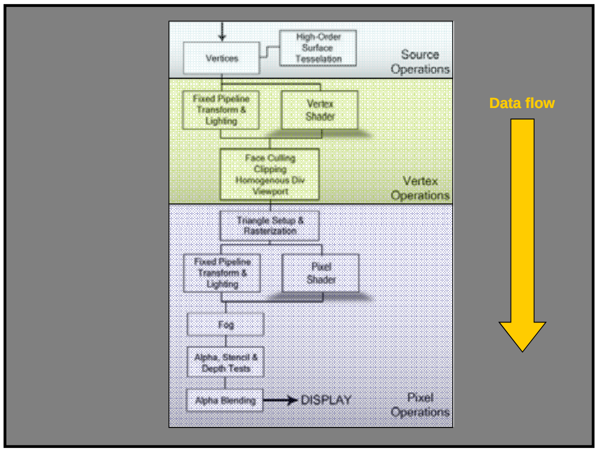


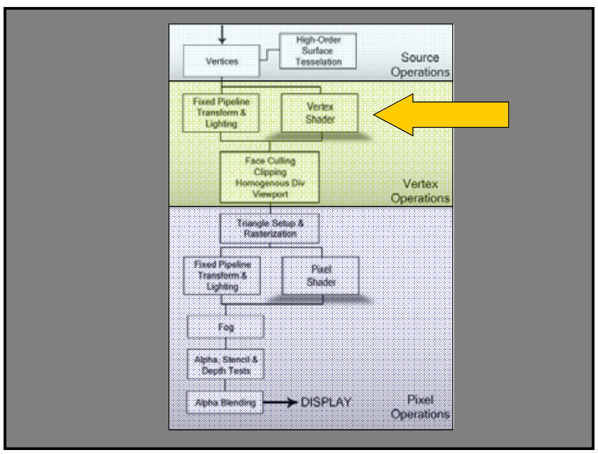
Pixels (what we see on the screen)

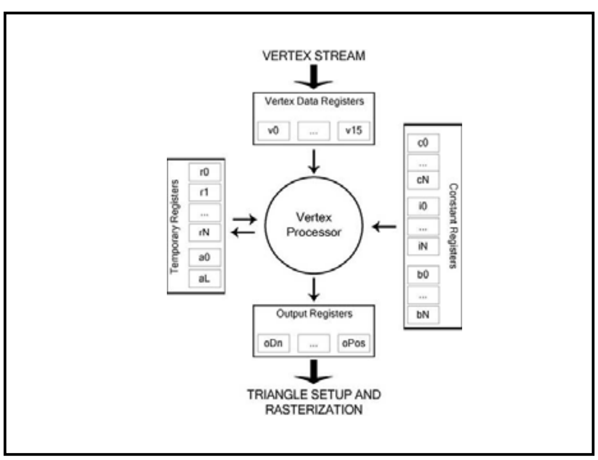
PC 3D Rendering in Hardware

Year	Graphics Card	Milestone
1987	IBM VGA	Provides a pixel frame buffer that the CPU has to fill
1996	3dfx Vodoo	Rasterizes and textures pre-transformed vertices (triangles)
1999	nVidia GeForce 256	Applies both transformation and lighting to vertices (T&L) – fixed pipeline
2001	nVidia GeForce 3	Configurable pixel shader and programmable vertex shader
2003	nVidia GeForce FX	Fully programmable pixel and vertex shaders



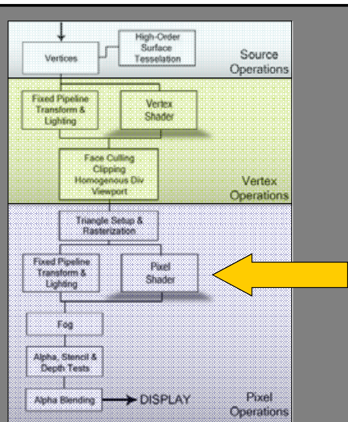

Vertices
High-Order Surface Tesselation
Source Operations
Fixed Pipeline Transform & Lighting
Vertex Shader
Face Culling Clipping Homogenous Div Viewport
Vertex Operations
Triangle Setup & Rasterization
Fixed Pipeline Transform & Lighting
Pixel Shader
Fog
Alpha, Stencil & Depth Tests
Alpha Blending
DISPLAY
Pixel Operations
Data flow

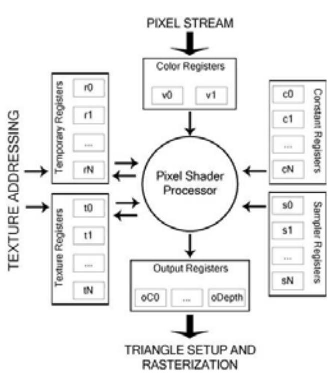

Vertices
High-Order Surface Tesselation
Source Operations
Fixed Pipeline Transform & Lighting
Vertex Shader
Face Culling Clipping Homogenous Div Viewport
Vertex Operations
Triangle Setup & Rasterization
Fixed Pipeline Transform & Lighting
Pixel Shader
Fog
Alpha, Stencil & Depth Tests
Alpha Blending
DISPLAY
Pixel Operations


VERTEX STREAM
Vertex Data Registers
v0
...
v15
Temporary Registers
r0
r1
...
rN
a0
aL
Vertex Processor
Constant Registers
c0
...
cN
i0
...
iN
b0
...
bN
Output Registers
oDn
...
oPos
TRIANGLE SETUP AND RASTERIZATION

Vertex Shader

```
// 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(0.299,0.587,0.184)).xxx
        return OUT;
}
```

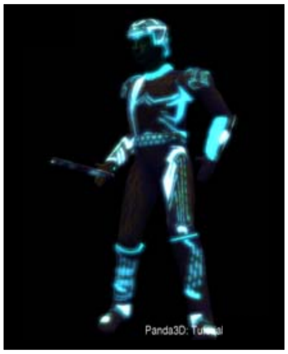
Categories of Shader Programs

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

Sceen Effects

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

Sceen Effects: Glow



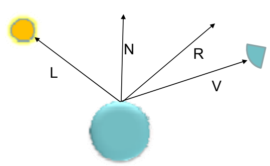
Sceen Effects: Depth of Field

Sceen Effects: Distortion

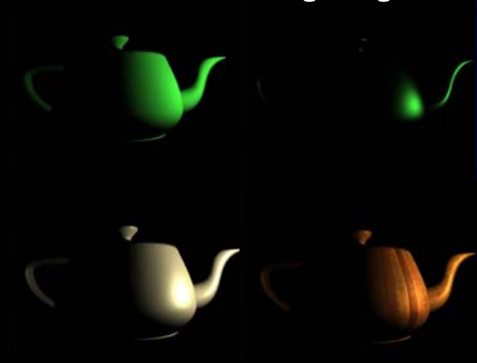
Sceen Effects: High Dynamic Range + Bloom

Lighting Models

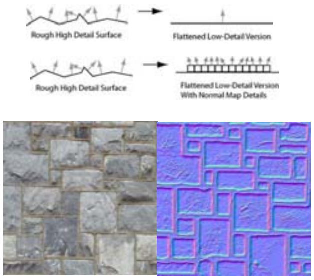
- Shaders calculate new color values by applying various lighting models, involving parameters such as surface normals (N), light angle (L), reflected light angle (R) and view angle (V).



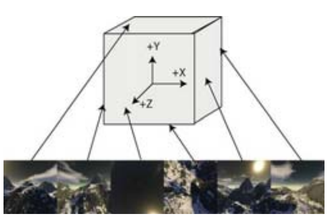
Lighting Models: Per-Pixel Lighting



Lighting Models: Normal Mapping



Lighting Models: Environment Reflection



Lighting Models: Shadows

Non-Photorealistic Rendering

Non-Photorealistic Rendering