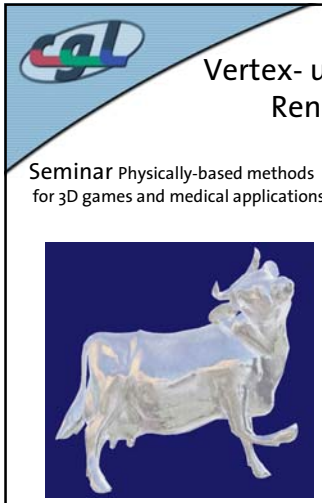
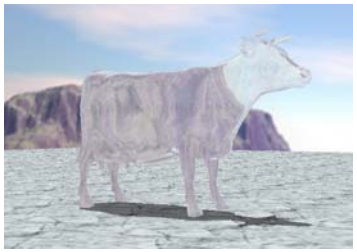


Vertex- und Pixelshader für Echtzeit-Rendering von Glas und Wasser

Seminar Physically-based methods for 3D games and medical applications



Wintersemester 02/03

Thomas Rusterholz

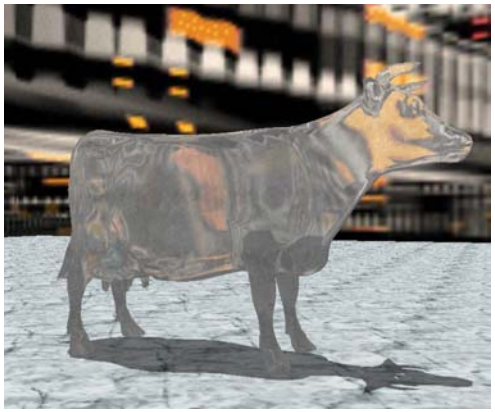
Quellen (1)

- NVIDIA GForce 3/4 Hardware
  - Vertex- und Pixelshader
  - www.nvidia.com/developer
- OpenGL
  - www.opengl.org
  - Extensions: http://oss.sgi.com/projects/ogl-sample/registry/

Quellen (2)

- MPI (Max-Planck-Institut)
  - A Vertex Program for Interactive Rendering of Realistic Shallow Water, Bastian Goldluecke, Marcus A. Magnor, Graphics – Optics – Vision, Max-Planck Institut für Informatik, Saarbrücken, Germany

Motivation (1)

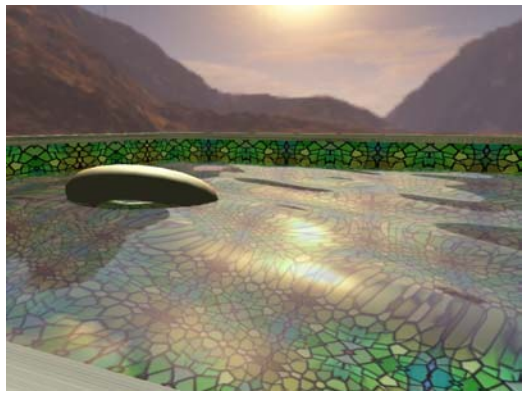


CGL

Motivation (2)

CGL

Motivation (3)



MPI

Tools

- Grafik Algorithmen in Hardware
- Effizient berechenbares physikalisches Modell

Übersicht

- Hardware
- Pipeline
- Vertex Shaders
- Register Combiners

Hardware (1)

- GPU = Graphics Processing Unit
  - Kennen nur wenige, auf Computergrafik ausgelegte Instruktionen
  - Instruktionen werden dafür sehr schnell ausgeführt

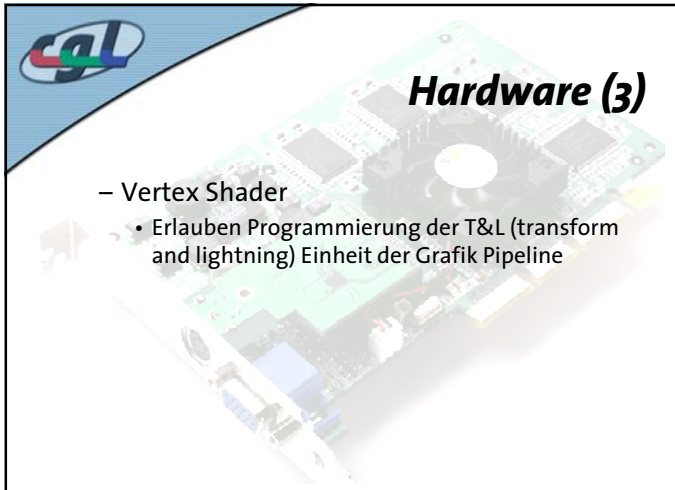
Hardware (2)

- CPU-intensive Berechnungen auf die GPU verlagern
  - Erzielen von hohen Frameraten
  - Interessant für Echzeitsimulationen und Spiele

Hardware (3)

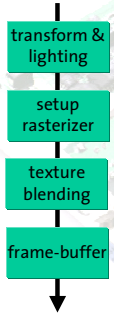
- NVIDIA GForce 3/4 Hardware
  - Vertex Shaders
  - Register Combiners

Hardware (3)



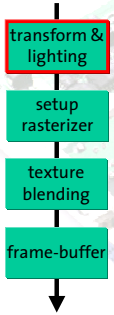
- Vertex Shader
  - Erlauben Programmierung der T&L (transform and lightning) Einheit der Grafik Pipeline

Pipeline (1)



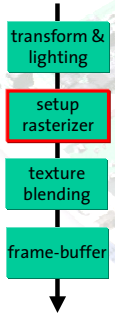
- Jede Einheit führt spezifische „fest verdrahtete“ Funktionen aus
  - Eventuell mit verschiedenen Operationsmodi

Pipeline (2)



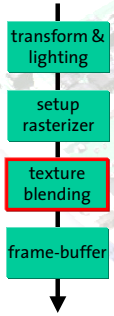
- Transform
  - Skalierung, Rotation und Translation
- Lightning
  - Beleuchtungsmodell wird an jedem Vertex ausgewertet

Pipeline (3)



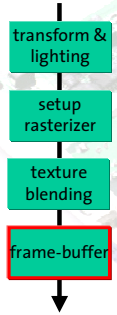
- Setup Rasterizer
  - Produziert Pixel und Fragments aus Vertices und Dreiecken

Pipeline (4)



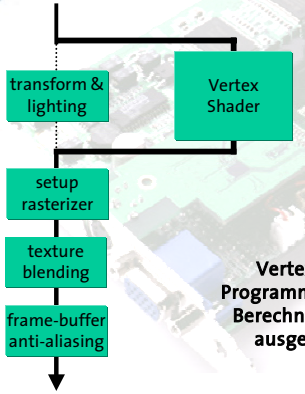
- Texture Blending
  - Mehrere Texturen werden kombiniert.

Pipeline (5)



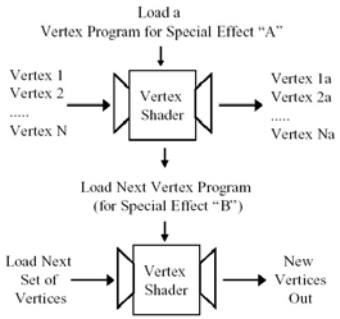
- Frame-Buffer
  - Der Display-Controller liest die Farbwerte aus dem Frame-Buffer und schickt sie weiter an den Treiber der Grafikkarte.

Pipeline (6)

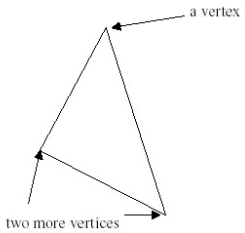


Vertex Shader geben dem Programmierer die Kontrolle über Berechnungen, die auf Vertices ausgeführt werden sollen.

Vertex Shaders (1)



Vertex Shaders (2)



Example: Vertex Data

position:	{X, Y, Z, W}
color:	{Red, Green, Blue, Alpha}
texture1:	{S, T, R, Q}
texture2:	{S, T, R, Q}
.....	
texture-n:	{S, T, R, Q}
fog:	{F}
specularity:	{P}

Vertex Shaders (3)

Ein komplettes Beispiel:

```
!!VP1.0
# c[0-3]  = modelview projection (composite) matrix
DP4   o[HPOS].x, c[0], v[OPOS]; # Compute position.
DP4   o[HPOS].y, c[1], v[OPOS];
DP4   o[HPOS].z, c[2], v[OPOS];
DP4   o[HPOS].w, c[3], v[OPOS];
MOV   o[COL0], v[3];                # Forward color.
END
```


Vertex Shaders (3)

Ein komplettes Beispiel:

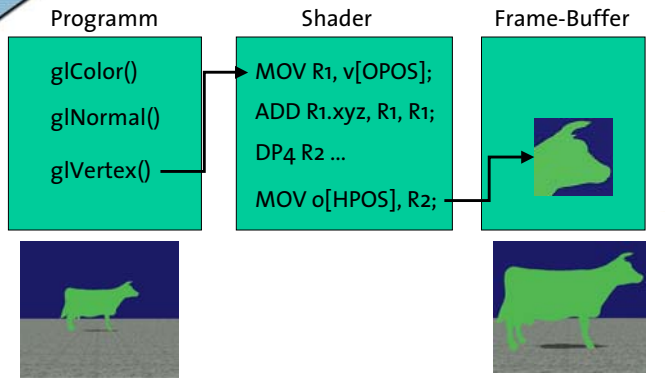
```
!!VP1.0
# c[0-3]  = modelview projection (composite) matrix
DP4   o[HPOS].x, c[0], v[OPOS]; # Compute position.
DP4   o[HPOS].y, c[1], v[OPOS];
DP4   o[HPOS].z, c[2], v[OPOS];
DP4   o[HPOS].w, c[3], v[OPOS];
MOV   o[COL0], v[3];                # Forward color.
END
```


Vertex Shaders (4)

Skalierung mit Faktor 2:

```
!!VP1.0
# c[0-3]  = modelview projection (composite) matrix
MOV   R1, v[OPOS];
ADD   R1.xyz, R1, R1;       # Scale by 2.
DP4   R2.x, c[0], R1;       # Compute position.
DP4   R2.y, c[1], R1;
DP4   R2.z, c[2], R1;
DP4   R2.w, c[3], R1;
MOV   o[HPOS], R2;
MOV   o[COL0], v[3];        # Forward color.
END
```

Vertex Shaders (5)

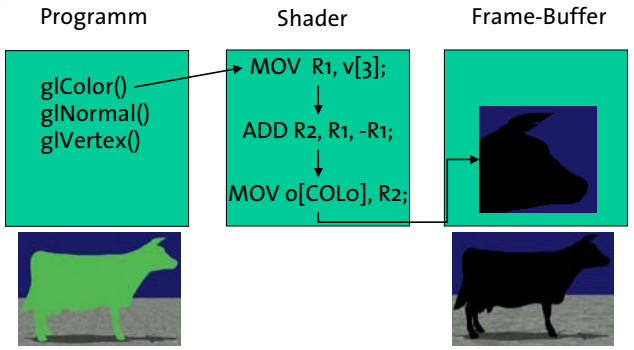


Vertex Shaders (6)

Spielen mit der Farbe:

```
!!VP1.0
# c[0-3]  = modelview projection (composite) matrix
DP4   o[HPOS].x, c[0], v[OPOS]; # Compute position.
DP4   o[HPOS].y, c[1], v[OPOS];
DP4   o[HPOS].z, c[2], v[OPOS];
DP4   o[HPOS].w, c[3], v[OPOS];
MOV   R1, v[3];
ADD   R2, R1, -R1;              # Make color black.
MOV   o[COL0], R2;              # Forward color.
END
```

Vertex Shaders (7)



Vertex Shaders (8)

Setup des Vertex Shaders:

```
!!VP1.0
MOV    R1, v[OPOS];
ADD    R1.xyz, R1, R1;
DP4...
...
```

```
glh_init_extension("GL_NV_vertex_program");
glGenProgramsNV(1, &_vertexProgId );
glBindProgramNV(GL_VERTEX_PROGRAM_NV, _vertexProgId);
glLoadProgramNV(GL_VERTEX_PROGRAM_NV, _vertexProgId,
    strlen(prog),(GLubyte *)prog);
glBindProgramNV(GL_VERTEX_PROGRAM_NV, _vertexProgId);
glTrackMatrixNV(GL_VERTEX_PROGRAM_NV, 0,
GL_MODELVIEW_PROJECTION_NV, GL_IDENTITY_NV);
glEnable(GL_VERTEX_PROGRAM_NV);
```

Setup

- glGenProgramsNV(sizei n, uint *ids)
- glLoadProgramNV(enum target, uint id,sizei len,const ubyte *program)
- glBindProgramNV(enum target, uint id)

Parameterübergabe (1)

- glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, index, x, y, z, w)
- glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, index, x, y, z, w)

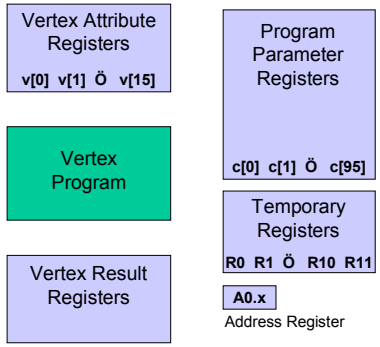
Parameterübergabe (2)

- „Tracking“ von Matrizen
  - TrackMatrixNV(GL_VERTEX_PROGRAM_NV, uint address, enum matrix, enum transform)

Registerset (1)

- 128 Programminstruktionen (SIMD)
- Alle Register sind 4-Komponenten Floating Point Vektor-Register

Registerset (2)



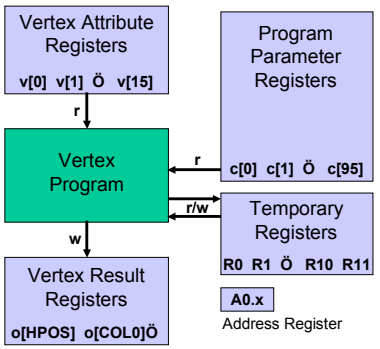
Vertex Input Register

Attribute Register	Mnemonic Name	Typical Meaning
v[0]	v[OPOS]	object position
v[1]	v[WGHT]	vertex weight
v[2]	v[NRML]	normal
v[3]	v[COL0]	primary color
v[4]	v[COL1]	secondary color
v[5]	v[FOGC]	fog coordinate
v[6]	-	-
v[7]	-	-
v[8]	v[TEX0]	texture coordinate 0
v[9]	v[TEX1]	texture coordinate 1
v[10]	v[TEX2]	texture coordinate 2
v[11]	v[TEX3]	texture coordinate 3
v[12]	v[TEX4]	texture coordinate 4
v[13]	v[TEX5]	texture coordinate 5
v[14]	v[TEX6]	texture coordinate 6
v[15]	v[TEX7]	texture coordinate 7

Vertex Output Register

Register Name	Description	Component Interpretation
o[HPOS]	Homogeneous clip space position	(x,y,z,w)
o[COL0]	Primary color (front-facing)	(r,g,b,a)
o[COL1]	Secondary color (front-facing)	(r,g,b,a)
o[BFC0]	Back-facing primary color	(r,g,b,a)
o[BFC1]	Back-facing secondary color	(r,g,b,a)
o[FOGC]	Fog coordinate	(f,*,*,*)
o[PSIZ]	Point size	(p,*,*,*)
o[TEX0]	Texture coordinate set 0	(s,t,r,q)
o[TEX1]	Texture coordinate set 1	(s,t,r,q)
o[TEX2]	Texture coordinate set 2	(s,t,r,q)
o[TEX3]	Texture coordinate set 3	(s,t,r,q)
o[TEX4]	Texture coordinate set 4	(s,t,r,q)
o[TEX5]	Texture coordinate set 5	(s,t,r,q)
o[TEX6]	Texture coordinate set 6	(s,t,r,q)
o[TEX7]	Texture coordinate set 7	(s,t,r,q)

Registerset (3)



Instruktionsset (1)

- SIMD Instruktionsset mit 17 Intruktionen
- Allgemeine Befehle wie MOV, MUL, MAD (Multiply-Add)
- Spezielle Befehle wie RCP (Reciprocal), RSQ (Reciprocal Square Root), DP3, DP4 (Dot Product)...

Instruktionsset (2)

Instruction Format:

Opcode dst, [-]s0 [,[-]s1 [,[-]s2]]; #comment

Instruction name | Destination Register | Source0 Register | Source1 Register | Source2 Register

Example:

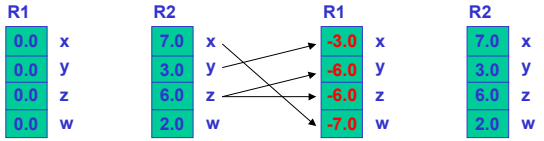
```
MOV  r1, r2
```

Instruktionsset (3)

Source registers can be negated and ì swizzled":

```
MOV    R1, -R2.yzzx;
```

before **after**



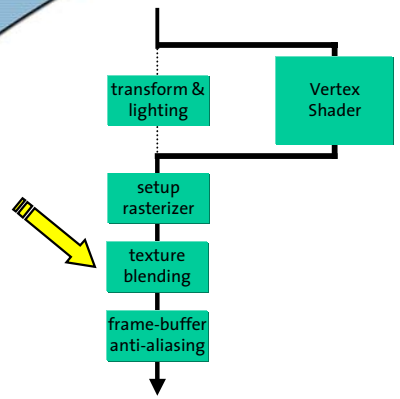
Zusammenfassung Vertex Shaders

- 1 Vertex Input und 1 Vertex Output
- Keine topologische Information vorhanden
- Kann dynamisch geladen und aktiviert werden

Register Combiners (1)

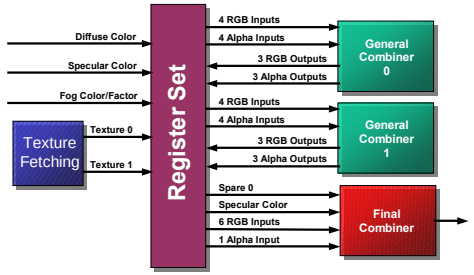
- Register Combiners
  - Erlauben Programmierung der Texture Blending Einheit der Grafik Pipeline

Register Combiners (2)



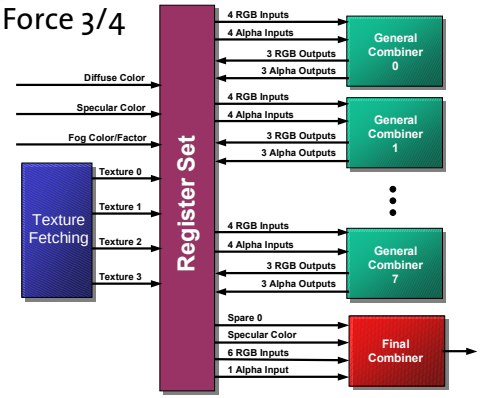
Register Combiners (3)

GForce 2



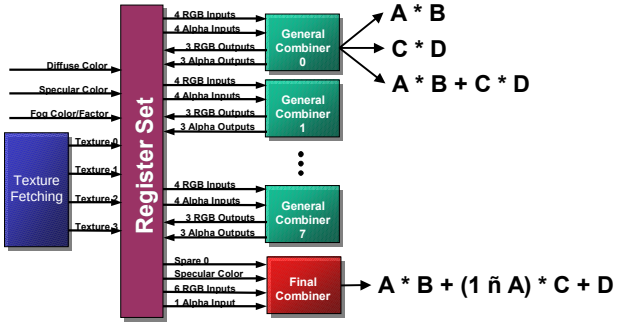
Register Combiners (4)

GForce 3/4



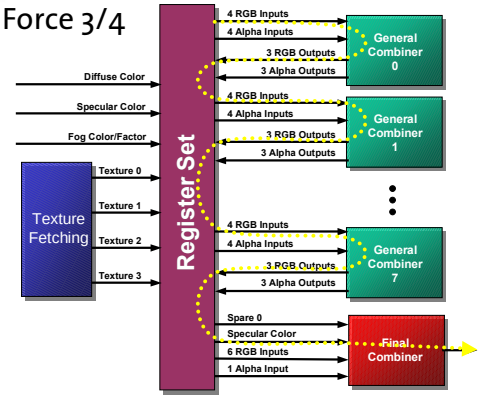
Register Combiners (4)

GForce 3/4



Register Combiners (5)

GForce 3/4



Register Combiners (7)

glCombinerParameteriNV(GL_NUM_GENERAL_COMBINERS_NV, 1);

// combiner 0

// a*b+c*d

glCombinerInputNV(GL_COMBINER0_NV, GL_RGB, GL_VARIABLE_A_NV, GL_TEXTURE0_ARB, GL_UNSIGNED_IDENTITY_NV, GL_RGB);

...

Register Combiners (8)

// (stage, portion, abOutput, cdOutput, sumOutput, scale, bias, abDotProduct, cdDotProduct, muxSum)

glCombinerOutputNV(GL_COMBINER0_NV, GL_RGB, GL_DISCARD_NV, GL_DISCARD_NV, GL_SPARE0_NV, GL_NONE, GL_NONE, GL_FALSE, GL_FALSE, GL_FALSE);

Register Combiners (9)

// final combiner

// output: Frgb = A*B + (1-A)*C + D

// (variable, input, mapping, componentUsage);

glFinalCombinerInputNV(GL_VARIABLE_A_NV, GL_SPARE0_NV, GL_UNSIGNED_IDENTITY_NV, GL_RGB);

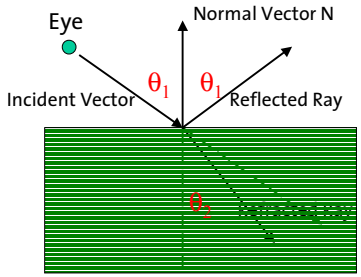
...

Übersicht

- Raytracing
- Demo Glas
- Tiefe
- Demo Wasser
- Ausblick

Raytracing (1)

Snell's law gives the angles of reflection and refraction.

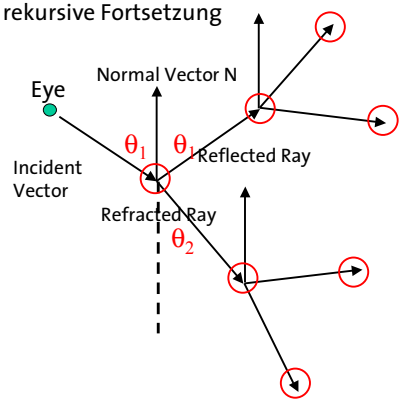


η (eta): refraction ratio

$\sin \theta_2 = \eta \sin \theta_1$

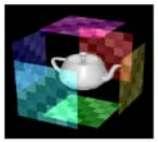
Raytracing (2)

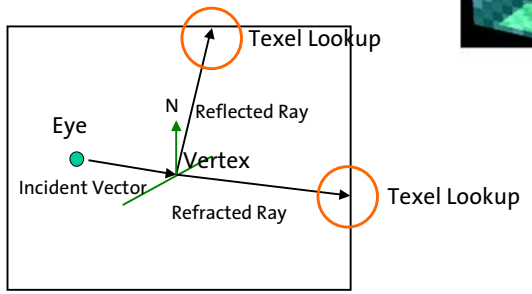
Raytracing: rekursive Fortsetzung



Raytracing (3)

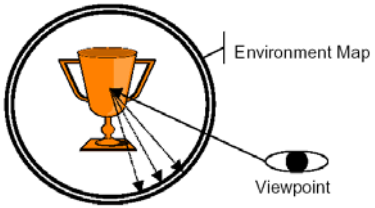
Approximation erster Ordnung
Look-Up in die Environment Map





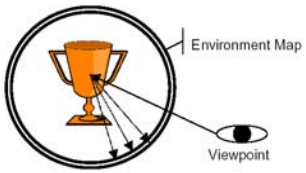
Raytracing (4)

Cubemap



Raytracing (5)

Problem?

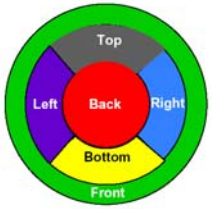


Raytracing (6)

Problem?

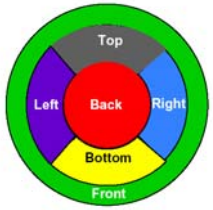
Raytracing (7)

Problem?



Raytracing (8)

Von vorne



Raytracing (9)

Fresnel's Law bestimmt den Koeffizienten **r** für die Gewichtung der Farben aus Reflektion und Brechung.

Approximation für Fresnel's Law:

$$r = bias + scale * (1 + I \cdot N)^{power}$$

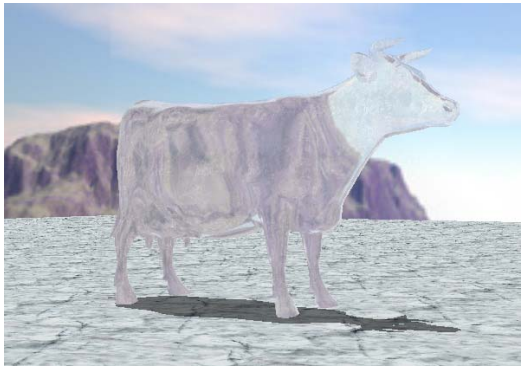
Raytracing (10)

Die beiden Farbwerte $C_{reflection}$ und $C_{refraction}$ werden gemäss Fresnel's Law übereinander geblendet.

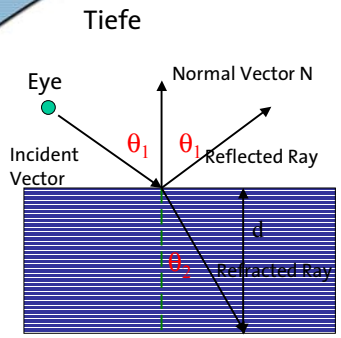
Mit Koeffizient **r** aus Fresnel:

$$C = r * C_{reflection} + (1 - r) * C_{refraction}$$

Demo Glas



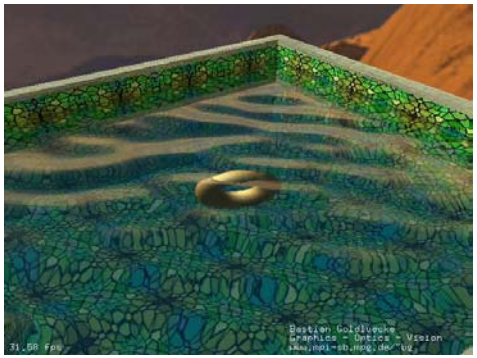
Raytracing (11)



p = Prozentsatz an Licht, der die Oberfläche erreicht

$$p = \exp\left(-\frac{C * d}{\cos\theta_2}\right)$$

Demo Wasser



Ausblick (1)

- NVIDIA CineFx Architektur
- GL_ARB_fragment_program Extension
  - Pixelshader statt Texture Shaders und Register Combiners

Ausblick (2)

- GL_ARB_fragment_program
  - Ähnlich wie GL_NV_vertex_program
  - Zusätzlich trigonometrische Funktionen sin, cos + Exponentialfunktion

CineFx (1)

- Vertex Shader
  - Branches
  - 256 statt 128 Instruktionen
  - 256 statt 96 Konstanten

CineFx (2)

- Pixel Shader
  - Keine Branches
  - Instruktionsset ähnlich Vertex Shaders

CineFx (3)

Elder Scrolls III, Bethesda Softworks, Inc.

CineFx (4)

CineFx Architecture Demo


Fragen...