Grundlagen der Spieleprogrammierung

Teil I: 3D-Graphik

Kapitel 7: Fortgeschrittene Techniken

Peter Sturm
Universität Trier

Outline

1. Übersicht und Motivation
2. Mathematische Grundlagen
3. Das Ideal: Photorealistisch (Raytracing, Radiosity)
4. Die Realität: DirectX und OpenGL (Übersicht)
5. Schritt 1: Modellierung und Drahtgitter
6. Schritt 2: Texturen
7. Schritt 4: Licht, Filter, etc.
8. Schritt 5: Fortgeschrittene Techniken (Vertex-, Pixel-Shader, …)
9. 3D-Hardware
10. 3D-Engines im Überblick, Cg von nvidia
11. Spielekonsolen
12. Zusammenfassung und Ausblick

Motivation

- Bisher
  - Primär dreieck-basierter Zugang
  - Color Maps für die farbliche Gestaltung der Oberfläche
  - Bump Maps für die "Rauhheit" der Objektoberfläche
  - Environmental Maps für Reflektionen der Objektumgebung
  - Mip Maps, Filter, Komprimierung, etc.
- Fortgeschrittene Techniken
  - Verbesserung der Gesamtdarstellung
  - Unterstützung für Licht und Schatten
  - Verlagerung von Rechenleistung auf GPU
    - Vertex Shader
    - Pixel Shader

Licht

- Direct3D Lichtmodell
  - Ambient Light
    - Konstante Hintergrundbeleuchtung
  - Diffuse Light
    - Objektabhängige Verbreitung einfallenden Lichtes
  - Specular Light
    - Objektabhängige Reflektion des Lichtes in Richtung Kamera
  - Emissive Light
    - Licht-emittierende Objekte

Ambientes Licht

Ambient Lighting = C_a*[G_a + sum(L_{ai}*Attenuation$_i$*SpotFactor$_i$)]

Formula for ambient lighting is incorrect. It should be Ambient Lighting = Ca*[Ga + sum(Lai)*Atti*Spoti] Where Att and Spot are attenuation and spot factors, computed for i-light.

The parameters are defined in the following table.

Parameter	Default value	Type	Description
C_a	(0,0,0,0)	D3DCOLORVALUE	Material ambient color.
G_a	(0,0,0,0)	D3DCOLORVALUE	Global ambient color.
sum	N/A	N/A	Summation of the ambient light from the light objects.
L_{ai}	(0,0,0,0)	D3DVECTOR	Light ambient color of the ith light.
Atten$_i$	(0,0,0,0)	D3DCOLORVALUE	Light attenuation of the ith light. See Attenuation and Spotlight Factor.
Spot$_i$	(0,0,0,0)	D3DVECTOR	Spotlight factor of the ith light. See Attenuation and Spotlight Factor.

The value for C_a is either:

- vertex color1, if AMBIENTMATERIALSOURCE = D3DMCS_COLOR1, and the first vertex color is supplied in the vertex declaration.
- vertex color2, if AMBIENTMATERIALSOURCE = D3DMCS_COLOR2, and the second vertex color is supplied in vertex declaration.
- material ambient color

Diffuse Light

Diffuse Lighting = sum[C_d*L_d*(N'L_{dir})*Atten*Spot]

Parameter	Default value	Type	Description
sum	N/A	N/A	Summation of each light's diffuse component.
C_d	(0,0,0,0)	D3DCOLORVALUE	Diffuse color.
L_d	(0,0,0,0)	D3DCOLORVALUE	Light diffuse color.
N	N/A	D3DVECTOR	Vertex normal
L_{dir}	N/A	D3DVECTOR	Direction vector from object vertex to the light.
Atten	N/A	FLOAT	Light attenuation. See Attenuation and Spotlight Factor.
Spot	N/A	FLOAT	Spotlight factor. See Attenuation and Spotlight Factor.

The value for C_d is either:

- vertex color1, if DIFFUSEMATERIALSOURCE = D3DMCS_COLOR1, and the first vertex color is supplied in the vertex declaration.
- vertex color2, if DIFFUSEMATERIALSOURCE = D3DMCS_COLOR2, and the second vertex color is supplied in the vertex declaration.
- material diffuse color

Specular Light

- Standardmäßig ausgeschaltet (`D3DRS_SPECULARENABLE`)

Specular Lighting = C_s*sum[L_s*$(N \cdot H)^P$*Atten*Spot]

The table identifies the variables, their types and their ranges.

Parameter	Default value	Type	Description
C_s	(0,0,0,0)	D3DCOLORVALUE	Specular color.
sum	N/A	N/A	Summation of each light's specular component.
N	N/A	D3DVECTOR	Vertex normal.
H	N/A	D3DVECTOR	Half way vector. See the section on the halfway vector.
P	0.0	FLOAT	Specular reflection power. Range is 0 to +infinity
L_s	(0,0,0,0)	D3DCOLORVALUE	Light specular color.
Atten	N/A	FLOAT	Light attenuation value. See Attenuation and Spotlight Factor.
Spot	N/A	FLOAT	Spotlight factor. See Attenuation and Spotlight Factor.

The value for C_s is either:

- vertex color1, if SPECULARMATERIALSOURCE = D3DMCS_COLOR1, and the first vertex color is supplied in the vertex declaration.
- vertex color2, if SPECULARMATERIALSOURCE = D3DMCS_COLOR2, and the second vertex color is supplied in the vertex declaration.
- material specular color

Emissive Light

Emissive Lighting = C_e

Where:

Parameter	Default value	Type	Description
C_e	(0,0,0,0)	D3DCOLORVALUE	Emissive color.

The value for C_e is either:

- vertex color1, if EMISSIVEMATERIALSOURCE = D3DMCS_COLOR1, and the first vertex color is supplied in the vertex declaration.
- vertex color2, if EMISSIVEMATERIALSOURCE = D3DMCS_COLOR2, and the second vertex color is supplied in the vertex declaration.
- material emissive color

Note If either EMISSIVEMATERIALSOURCE option is used, and the vertex color is not provided, the material emissive color is used.

Example

In this example, the object is colored using the scene ambient light and a material ambient color. The code is shown below.

```
// create material
D3DMATERIAL9 mtrl;
ZeroMemory( &mtrl, sizeof(mtrl) );
mtrl.Emissive.r = 0.0f;
mtrl.Emissive.g = 0.75f;
mtrl.Emissive.b = 0.0f;
mtrl.Emissive.a = 0.0f;
m_pd3dDevice->SetMaterial( &mtrl );
m_pd3dDevice->SetRenderState(D3DRS_EMISSIVEMATERIALSOURCE, D3DMCS_MATERIAL);
```

Attenuation - Abschwächung

The attenuation of a light depends on the type of light and the distance between the light and the vertex position. To calculate attenuation, use one of the following three equations.

$$Atten = 1/(att0_i + att1_i * d + att2_i * d^2)$$

Where:

Parameter	Default value	Type	Description
$att0_i$	0.0	FLOAT	Linear attenuation factor. Range is 0 to +infinity
$att1_i$	0.0	FLOAT	Squared attenuation factor. Range is 0 to +infinity
$att2_i$	0.0	FLOAT	Exponential attenuation factor. Range is 0 to +infinity
d	N/A	FLOAT	Distance from vertex position to light position

- Atten = 1, if the light is a directional light.
- Atten = 0, if the distance between the light and the vertex exceeds the light's range.

The att0, att1, att2 values are specified by the *Attenuation0*, *Attenuation1*, and *Attenuation2* members of D3DLIGHT9.

The distance between the light and the vertex position is always positive.

$$d = | L_{dir} |$$

Spotlight -Faktor

$$spot_i = \begin{cases} 1 & \text{for non- spotlights or if } rho_i > \cos(\frac{theta_i}{2}) \\ 0 & \text{if } rho_i \le \cos(\frac{phi_i}{2}) \\ \left[\frac{rho_i - \cos(\frac{phi_i}{2})}{\cos(\frac{theta_i}{2}) - \cos(\frac{phi_i}{2})}\right]^{falloff} & \text{otherwise} \end{cases}$$

Parameter	Default value	Type	Description
rho_i	N/A	FLOAT	cosine(angle) for spotlight i
phi_i	0.0	FLOAT	Penumbra angle of spotlight i in radians. Range: [$theta_i$, pi)
$theta_i$	0.0	FLOAT	Umbra angle of spotlight i in radians. Range: [0, pi)
falloff	0.0	FLOAT	Falloff factor. Range: (-infinity, +infinity)

Where:

$$rho = norm(L_{dcs}) \cdot norm(L_{dir})$$

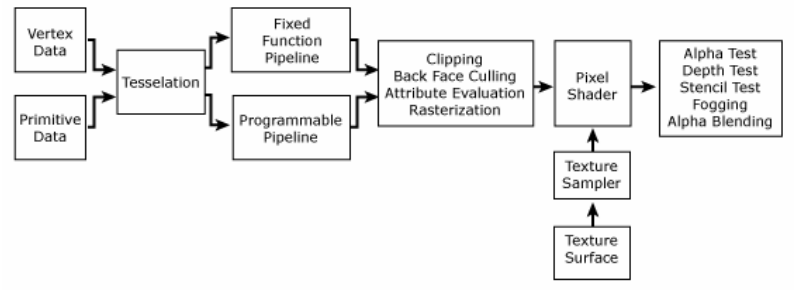
and:

Parameter	Default value	Type	Description
L_{dcs}	N/A	D3DVECTOR	The negative of the light direction in camera space
L_{dir}	N/A	D3DVECTOR	Direction vector from vertex position to the light position

Beispiel

Shader

- Verlagerung von Rechenleistung in die GPU
- Vertex Shader
  - Berechnungen auf Knotenebene
- Pixel Shader
  - Berechnungen auf Pixelebene



Vertex Shader

- Kleine Assemberprogramme
- Anbindbar an einen Vertex Stream
- Ausführung für jeden Vertex eines Streams
  - Vordefinierte Input- und Hilfsregister
  - Ergebnis steht in vordefinierten Ausgaberegistern
  - Repertoir an Shader-Instruktionen

Input- und Output-Register

- Input

Name	Register type	Count	Data type	Dimension	I/O permissions	Read port	Read / Instruction	Rel-Address	Defaults	Requires DCL
a0	address	1	integer	4-D vector (1)	write / use	1	3	no	none	no
c#	float constant	96 (2)	floating-point	4-D vector	define / read	1	3	a0.x	(0, 0, 0, 0)	no
r#	temporary	12	floating-point	4-D vector	write / read	3	3	no	none	no
v#	input	16	floating-point	4-D vector	read	1	3	no	(0, 0, 0, 1)	yes

- Output

Name	Register type	Count	Data type	Dimension	I/O permissions	Rel-Address	Default	Require DCL
oD#	diffuse / specular (3)	2	floating-point	4-D vector	write	no	none	no
oFog	fog	1	floating-point	scalar	write	no	none	no
oPos (4)	position	1	floating-point	4-D vector	write	no	none	no
oPts	point size	1	floating-point	scalar	write	no	none	no
oT#	texture coordinate	8	floating-point	4-D vector	write	no	none	no

Instruktionen

Name	Description	Instruction slots	Setup	Arithmetic	Macro-ops	New
add	Add two vectors	1		x		x
dcl_usage	Declare input vertex registers. See Registers - vs_1_1	0	x			x
def	Define constants	0	x			x
dp3	Three-component dot product	1		x		x
dp4	Three-component dot product	1		x		x
dst	Distance	1		x		x
exp	Full precision 2^x	10			x	x
expp	Partial precision 2^x	1			x	x
frc	Fractional component	3			x	x
lit	Calculate lighting	1		x		x
log	Full precision $\log_2(x)$	10			x	x
logp	Partial precision $\log_2(x)$	1			x	x
m3x2	3x2 matrix multiply	2			x	x
m3x3	3x3 matrix multiply	3			x	x
m3x4	3x4 matrix multiply	4			x	x
m4x3	4x3 matrix multiply	3			x	x
m4x4	4x4 matrix multiply	4			x	x
mad	Multiply and add	1		x		x
max	Maximum	1		x		x
min	Minimum	1		x		x
mov	Move	1		x		x
mul	Multiply	1		x		x
nop	No operation	0		x		x
rcp	Reciprocal	1		x		x
rsq	Reciprocal square root	1		x		x
sge	Greater than or equal compare	1		x		x
slt	Less than compare	1		x		x
sub	Subtract	1		x		x
vs	Version	0	x			x

Beispiel

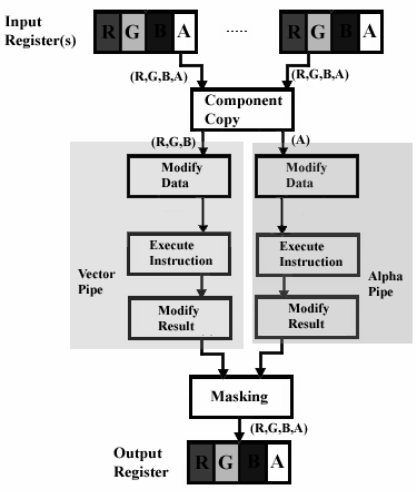
Ripple.vsh - Notepad

File Edit Format View Help

```
vs.1.1
; Constants:
;
;  c0-c3  - View+Projection matrix
;
;  c4.x   - time
;  c4.y   - 0
;  c4.z   - 0.5
;  c4.w   - 1.0
;
;  c7.x   - pi
;  c7.y   - 1/2pi
;  c7.z   - 2pi
;  c7.w   - 0.05
;
;  c10    - first 4 taylor coefficients for sin(x)
;  c11    - first 4 taylor coefficients for cos(x)

dcl_position v0

; Decompress position
mov r0.x, v0.x
mov r0.y, c4.w       ; 1
mov r0.z, v0.y
mov r0.w, c4.w       ; 1

; Compute theta from distance and time
mov r4.xz, r0        ; xz
mov r4.y, c4.y       ; y = 0
dp3 r4.x, r4, r4     ; d2
rsq r4.x, r4.x
rcp r4.x, r4.x       ; d
mul r4.xyz, r4, c4.x     ; scale by time

; Clamp theta to -pi..pi
add r4.x, r4.x, c7.x
mul r4.x, r4.x, c7.y
frc r4.xy, r4.x
mul r4.x, r4.x, c7.z
add r4.x, r4.x,-c7.x

; Compute first 4 values in sin and cos series
mov r5.x, c4.w       ; d^0
mov r4.x, r4.x       ; d^1
```

Pixel Shader

- Kleine Assemberprogramme
- Können bei der Bestimmung der Pixelfarbe intervenieren
- Grobarchitektur ähnlich Vertex Shader
- 2 Instruktionstypen
  - Arithmetische Instruktionen
  - Texturinstruktionen



Register

		Versions			
Name	**Type**	**1_1**	**1_2**	**1_3**	**1_4**
cn	Constant register	8	8	8	8
rn	Temporary register	2	2	2	6
tn	Texture register	4	4	4	6
vn	Color register	2	2	2	2 in phase 2

- **Constant registers** contain constant data organized in four fixed-point values. Data can be loaded into a constant register using SetPixelShaderConstant or it can be defined using def. Constant registers are not usable by texture address instructions. The only exception is the texm3x3spec instruction, which uses a constant register to supply an eye-ray vector.

Pixel shader versions	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_sw**	**2_x**	**3_0**	**3_sw**
Constant Registers	x	x	x	x	x	x	x	x	x

- **Temporary registers** are used to store intermediate results, as four fixed-point values. r0 additionally serves as the pixel shader output. The value in r0 at the end of the shader is the pixel color for the shader.

  Shader pre-processing will fail CreatePixelShader on any shader that attempts to read from a temporary register that has not been written by a previous instruction. D3DXAssembleShader will fail similarly, assuming validation is enabled (do not use D3DXASM_SKIPVALIDATION).

Pixel shader versions	**1_1**	**1_2**	**1_3**	**1_4**	**2_0**	**2_sw**	**2_x**	**3_0**	**3_sw**
Temporary Registers	x	x	x	x	x	x	x	x	x

- **Texture registers**

  For pixel shader version 1_1 to 1_3, texture registers contain texture data, organized in four fixed-point values. Texture data is loaded into a texture register when a texture is sampled. Texture sampling uses texture coordinates to look up, or sample, a color value at the specified (u,v,w,q) coordinates while taking into account the texture stage state attributes. The texture coordinate data is interpolated from the vertex texture coordinate data and is associated with a specific texture stage. There is a default one-to-one association between texture stage number and texture coordinate declaration order. By default, the first set of texture coordinates defined in the vertex format is associated with texture stage 0.

  For these pixel shader versions, texture registers behave just like temporary registers when used by arithmetic instructions.

  For pixel shader version 1_4, texture registers (t#) contain read-only texture coordinate data. This means that the texture

Instruktionslimits

Instruction type	Version				
	1_1	1_2	1_3	1_4 phase 1	1_4 phase 2
Version	*	*	*	*	N/A
Constant definition	*	*	*	*	N/A
Phase	N/A	N/A	N/A	*	*
Arithmetic	8	8	8	8	8
Texture address	4	4	4	6	6
Total	12	12	12	14	14

Arithmetische Instruktionen

Arithmetic instructions			1_1	1_2	1_3	1_4
add	Add two vectors	1	x	x	x	x
bem	Apply a fake bump environment-map transform	2				x
cmp	Compare source to 0	1		x	x	x
cnd	Compare source to 0.5	1	x	x	x	x
dp3	Three-component dot product	1	x	x	x	x
dp4	Four-component dot product	1		x	x	x
lrp	Linear interpolate	1	x	x	x	x
mad	Multiply and add	1	x	x	x	x
mov	Move	1	x	x	x	x
mul	Multiply	1	x	x	x	x
nop	No operation	0	x	x	x	x
sub	Subtract	1	x	x	x	x

Texture Instructions

Texture instructions			1_1	1_2	1_3	1_4
tex	Sample a texture	1	x	x	x	
texbem	Apply a fake bump environment-map transform	1	x	x	x	
texbeml	Apply a fake bump environment-map transform with luminance correction	1+1	x	x	x	
texcoord	Interpret texture coordinate data as color data	1	x	x	x	
texcrd	Copy texture coordinate data as color data	1				x
texdepth	Calculate depth values	1				x
texdp3	Three-component dot product between texture data and the texture coordinates	1		x	x	
texdp3tex	Three-component dot product and 1-D texture lookup	1		x	x	
texkill	Cancels rendering of pixels based on a comparison	1	x	x	x	x
texld	Sample a texture	1				x
texm3x2depth	Calculate per-pixel depth values	1			x	
texm3x2pad	First row matrix multiply of a two-row matrix multiply	1	x	x	x	
texm3x2tex	Final row matrix multiply of a two-row matrix multiply	1	x	x	x	
texm3x3	3x3 matrix multiply	1		x	x	
texm3x3pad	First or second row multiply of a three-row matrix multiply	1	x	x	x	
texm3x3spec	Final row multiply of a three-row matrix multiply	1	x	x	x	
texm3x3tex	Texture look up using a 3x3 matrix multiply	1	x	x	x	
texm3x3vspec	Texture look up using a 3x3 matrix multiply, with non-constant eye-ray vector	1	x	x	x	
texreg2ar	Sample a texture using the alpha and red components	1	x	x	x	
texreg2gb	Sample a texture using the green and blue components	1	x	x	x	
texreg2rgb	Sample a texture using the red, green and blue components	1		x	x	

Where 1 + 1 counts as 1 arithmetic instruction + 1 texture instruction.