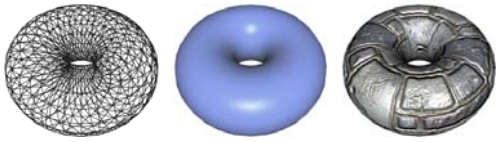
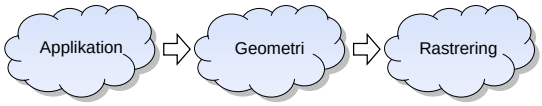
Shaders



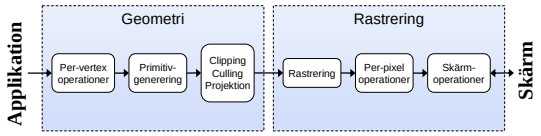
Martin Fitger
d00-mfi@d.kth.se

VT 2008, DH2323 / DH2640 / NA8740

Renderingssystem



Renderingssystem



Renderingssystem

- CPU-intensiv
- Hög parallellitet
- Stor efterfrågan på interaktiva applikationer
- Datorspel

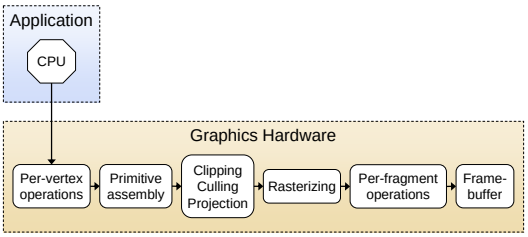
Hårdvara för 3D-rendering

- Avlasta CPUn
- Fysisk implementering av pipeline
- Hög parallellitet
- Utvecklas snabbare än Moore's Law.

Hårdvara för 3D-rendering

- Tidigt 90-tal tillgängligt på konsumentnivå
- Till en början endast rastrering
- Begränsat antal fasta funktioner
- Icke-flexibel
- "Syntetiska" bilder

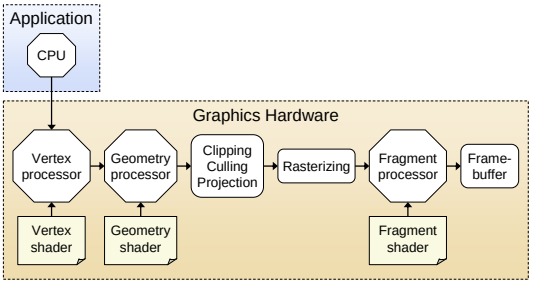
Fixed-Function-Pipeline



Programmerbar pipeline

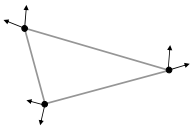
- Vid millennieskiftet
- Shaders
- Godtyckliga effekter
- Flexibel
- Realistiska bilder

Programmerbar pipeline



Vertex Shader

- Exekveras en gång per vertex
- Transformationer (vertex, normal)
- Generering av texturkoordinater
- Belysning
- etc.

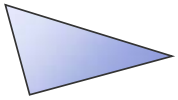


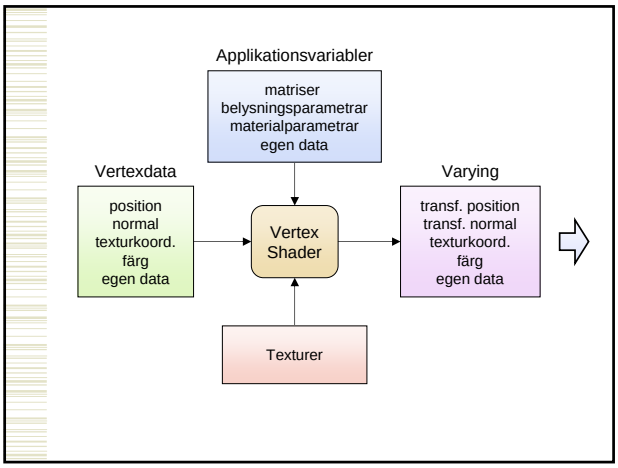
Geometry Shader

- Exekveras en gång per primitiv (linje, triangel)
- Generering av nya vertex/primitiver
- Användningsområden:
  - Tesselering (parametriserade ytor)
  - Point-sprites
  - Skuggvolymer

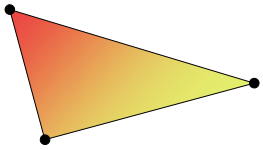
Fragment Shader

- Exekveras en gång per fragment (pixel)
- Texturläsning
- Färgblandning
- Belysning
- etc.

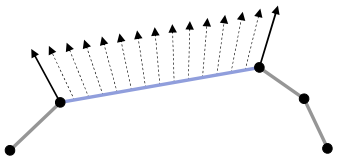


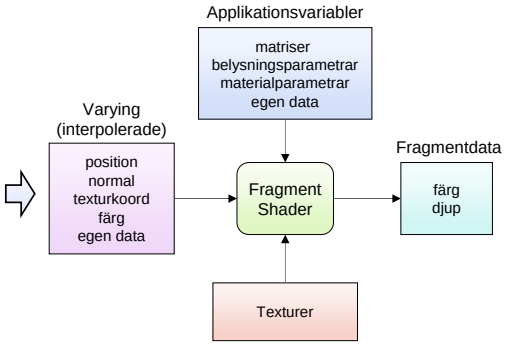


Interpolering av varying



Interpolering av varying





Shaderspråk

Assembler

- Effektiva program
- Tidsödande
- Kompatibilitetsproblem

Shaderspråk

D3D Vertex Shader Assembler

```
dp3 r7.w, VECTOR_VERTEXTOLIGHT, VECTOR_VERTEXTOLIGHT
rsq VECTOR_VERTEXTOLIGHT.w, r7.w
dst r7, r7.wwww, VECTOR_VERTEXTOLIGHT.wwww
mul r6, r5, ATTENUATION.w
```

OpenGL Vertex Program Assembler

```
MOV R1, R2.yzwx;
MUL R1.xyz, R2, R3;
ADD R1, R2, -R3;
RCP R1, R2.w;
```

Shaderspråk

Högnivåspråk

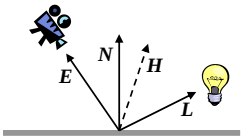
- GLSL, HLSL, Cg
- Liknar C
- Extra datatyper: vektorer, matriser mm.
- Inbyggda vektor- och grafikoperationer

Shaderspråk

GLSL Vertex Shader

```
varying float LightIntensity;

void main()
{
    vec3 n = normalize(gl_NormalMatrix * gl_Normal);

    LightIntensity = max(
        dot(n, gl_LightSource[0].position), 0.0);

    gl_Position = ftransform();
}
```

Exempel:
Phong's reflektionsmodell



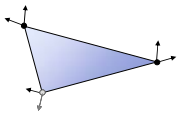
$$\boldsymbol{I} = k_{ambient} + k_{diffuse}(\boldsymbol{N} \bullet \boldsymbol{L}) + k_{specular}(\boldsymbol{N} \bullet \boldsymbol{H})^{shininess}$$

$$\boldsymbol{H} = (\boldsymbol{L} + \boldsymbol{E}) / |(\boldsymbol{L} + \boldsymbol{E})|$$

Exempel:
Phong's reflektionsmodell

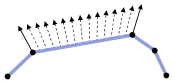
Alternativ 1 (Gouraud shading)

- Utför beräkningar i vertex shader
- Interpolera resulterande färgen



Alternativ 2 (Phong shading)

- Interpolera normaler etc.
- Utför beräkningar i fragment shader



Alternativ 1
(Gouraud shading)
Vertex Shader

```
// Applikationsvariabler
uniform vec3  Ambient;
uniform vec3  Diffuse;
uniform vec3  Specular;
uniform float Shininess;

// Interpoleringsvariabler
varying vec3 v_color;

void main()
{
    // Beräkna vektorer
    vec3 E = normalize(-vec3(gl_ModelViewMatrix * gl_Vertex));
    vec3 N = gl_NormalMatrix * gl_Normal;
    vec3 L = gl_LightSource[0].position.xyz;
    vec3 H = normalize(L + E);

    // Matt reflektion
    float diff = max(dot(L, N), 0.0);

    // Speglad reflektion
    float spec = pow(max(dot(H, N), 0.0), Shininess);

    // Beräkna färg
    v_color = Ambient + (Diffuse * diff) + (Specular * spec);

    // Transformera vertex
    gl_Position = ftransform();
}
```

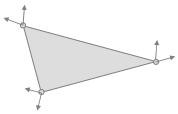
Alternativ 1
(Gouraud shading)
Fragment Shader

```
// Interpoleringsvariabler
varying vec3 v_color;

void main()
{

    // Skriv ut färg
    gl_FragColor = vec3(v_color, 1.0);

}
```

Exempel:
Phong's reflektionsmodell

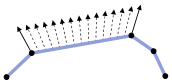
Alternativ 1 (Gouraud shading)

- Utför beräkningar i vertex shader
- Interpolera resulterande färgen



Alternativ 2 (Phong shading)

- Interpolera normaler etc.
- Utför beräkningar i fragment shader



Alternativ 2
(Phong shading)

Vertex Shader

```
// Interpoleringsvariabler
varying vec3 v_E;  // Eye vector
varying vec3 v_N;  // Normal vector

void main()
{
    // Beräkna och passa vidare vektorer
    v_E = normalize(-vec3(gl_ModelViewMatrix * gl_Vertex));
    v_N = normalize(gl_NormalMatrix * gl_Normal);

    // Transformera vertex
    gl_Position = ftransform();
}
```

Alternativ 2
(Phong shading)

Fragment Shader

```
// Applikationsvariabler
uniform vec3  Ambient;
uniform vec3  Diffuse;
uniform vec3  Specular;
uniform float Shininess;

// Interpoleringsvariabler
varying vec3 v_E;  // Eye vector
varying vec3 v_N;  // Normal vector

void main()
{
    // Beräkna vektorer
    vec3 E = normalize(v_E);
    vec3 N = normalize(v_N);
    vec3 L = gl_LightSource[0].position.xyz;
    vec3 H = normalize(L + E);

    // Matt reflektion
    float diff = max(dot(L, N), 0.0);

    // Speglad reflektion
    float spec = pow(max(dot(H, N), 0.0), Shininess);

    // Beräkna färg
    vec3 color = Ambient + (Diffuse * diff) + (Specular * spec);

    // Skriv ut färg
    gl_FragColor = vec3(color, 1.0);
}
```

Exempel:
Phong's reflektionsmodell

Alternativ 1
(Gouraud shading)

Alternativ 2
(Phong shading)

Inte bara shading

- Generella processorer
- Billigt
- Hög prestanda (flera 100 miljarder flops)
- Andra beräkningsområden

Inte bara shading

- Beräkningsintensiva problem
- Hög parallellitet
- Problemet måste anpassas
- Användningsområden:
  - Fysiksimulering
  - Bioinformatik
  - Lösa ekvationssystem
  - mm

Shaderutveckling

- Högnivåspråk underlättar utvecklingen
- Krävs programmerare för att skriva shaders
- Shading hör till grafikers område

Shaderutveckling

Verktyg

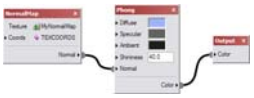
- FX Composer (Nvidia)
- RenderMonkey (ATI)
- Maya/3DSMax

FX Composer

Shaderutveckling

Visuell programmering

- Grafer
- Automatisk generering av kod
- Finns som verktyg till spelmotorer

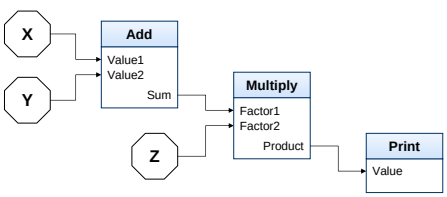


Visuell programmering

```
print( (X + Y) * Z )
```

Visuell programmering

```
print( (X + Y) * Z )
```



Demonstration