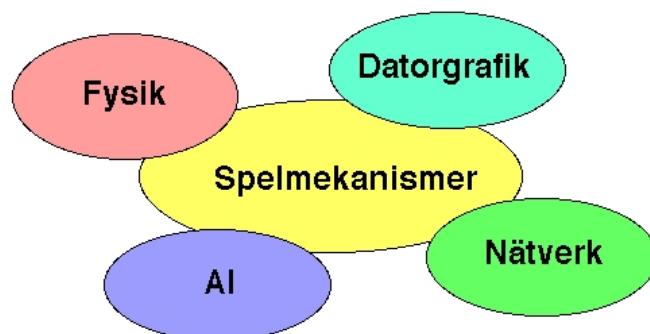


# **TSBK 03**

## **Teknik för avancerade datorspel**

**Ingemar Ragnemalm, ISY**



Ingemar  
Ragnemalm  
[ingis@isy.liu.se](mailto:ingis@isy.liu.se)

## **Föreläsning 4**

### **Avancerade shaders**

- Multipass-shaders
- Filter, färltning
- Flyttalsbuffrar
- High dynamic range
- Bump mapping med utvidgningar

Ingemar  
Ragnemalm  
[ingis@isy.liu.se](mailto:ingis@isy.liu.se)

# Multipass-shaders

**Enkla shaders kan göras i ett pass.**

**De flesta avancerade behöver flera pass.**

**Rendera till textur, sedan från denna textur till ännu en...**

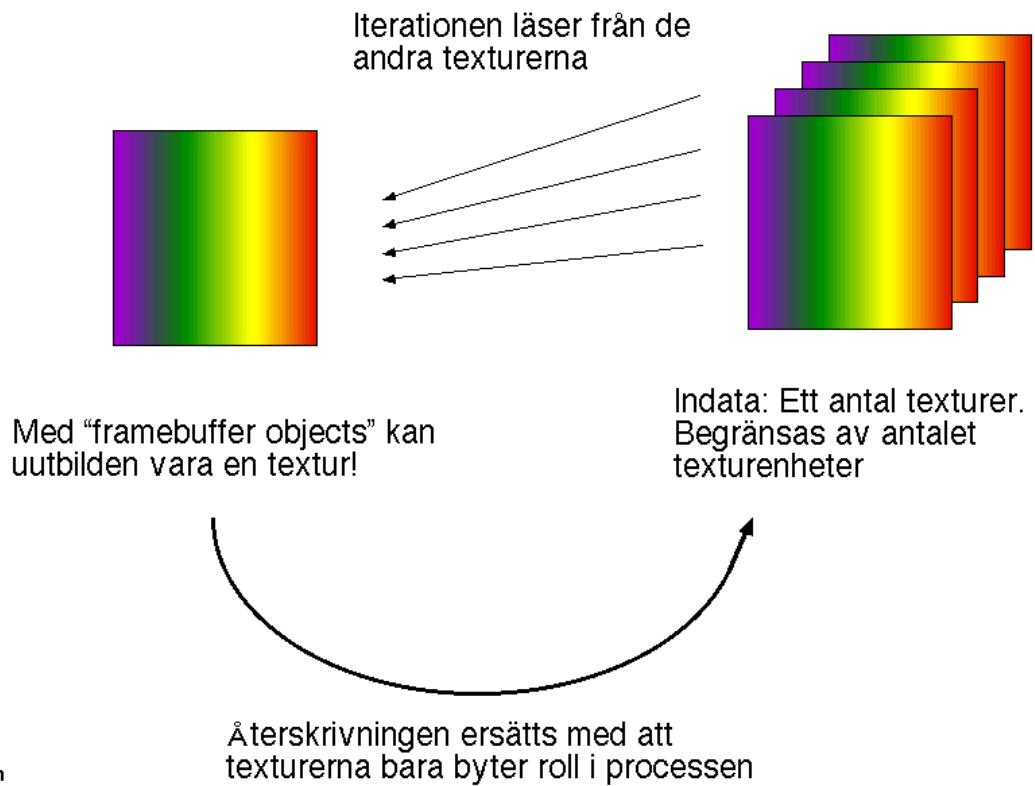
Ingemar  
Ragnemalm  
ingis@isy.liu.se

## Geometri

**Oftast bara en enkel rektangel tvärs över viewporten under ping-ponging!**

```
glBegin(GL_QUADS);
    glTexCoord2f(0, 0);
    glVertex2i(-1, -1);
    glTexCoord2f(0, 1);
    glVertex2i(-1, 1);
    glTexCoord2f(1, 1);
    glVertex2i(1, 1);
    glTexCoord2f(1, 0);
    glVertex2i(1, -1);
glEnd();
```

# “Ping-pong”-ing



Ingemar  
Ragnemalm  
ingis@isy.liu.se

## ping-pong-teknik

### Välj källa:

```
glBindTexture(GL_TEXTURE_2D, tx1);
```

### Välj destination:

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, tx2, 0);
```

### Välj shader:

```
glUseProgramObjectARB(shaderProgramObject);
```

**Rendera! Upprepa!**

Ingemar  
Ragnemalm  
ingis@isy.liu.se

# Filtrering, faltning

**Vanligt problem: applicera filter i form av faltningskärnor**

**Alla typer av linjära filter:**

- **Medelvärdesbildning (smoothing)**
  - Gradient, embossing

## Faltning (Convolution)

$$(f \otimes g)(t) = \sum f(\tau) \cdot g(t-\tau)$$

**Observera att ena signalen “vänds”.  
(Annars blir det korrelation, en annan operation.)**

**Detta kvittar för vanliga LP-filter.**

## 5x5 faltningskärna för LP-filter

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

/256

## Sobeloperator

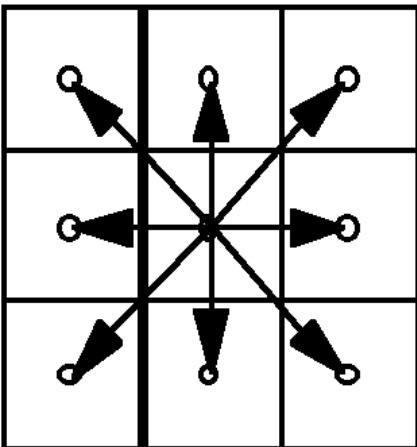
-1	0	1
-2	0	2
-1	0	1

## Gradientfilter

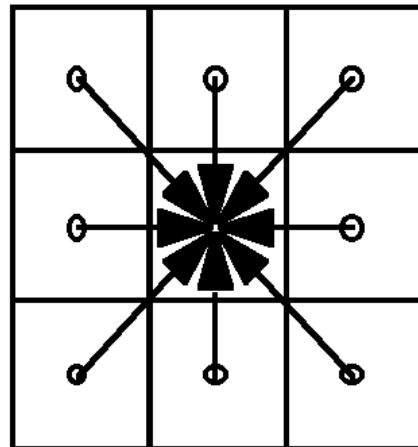
-1	1
----	---

-1
1

# Implementation av fästning



**Scatter**



**Gather**

# Implementation av fästning

**Scatter:** Dålig för GPUer! Kan inte alls utföras i shader. Kan utföras i t.ex. CUDA, men mindre effektivt än gather.

**Gather:** Enkel och effektiv på GPU!  
Stämmer bra med grundmodellen med fast utdata-fragment!

# Separerbara filter

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

$$\begin{array}{c} \boxed{1} \quad \boxed{2} \quad \boxed{1} \\ \otimes \end{array} \quad \begin{array}{c} \boxed{1} \quad \boxed{2} \quad \boxed{1} \\ \otimes \end{array}$$

=

$$\begin{array}{c} \boxed{1} \\ \boxed{2} \\ \boxed{1} \end{array} \quad \otimes \quad \begin{array}{c} \boxed{1} \\ \boxed{2} \\ \boxed{1} \end{array}$$

**Kan ge betydande acceleration för vissa filter.**

Ingemar  
Ragnemalm  
ingis@isy.liu.se

# Separerbara filter

Vad är flaskhalsen?

Beräkningar?  
25 MUL blev 12  
24 ADD blev 8

Texturaccesser? 25 blev 12.

Bildgenomgångar? 1 blev 4.

Troligast texturaccesser! Minnesaccesser är dyra, beräkningar billiga!

Testa!

Ingemar  
Ragnemalm  
ingis@isy.liu.se

**Alternativ metod för LP-filter:**

**Nedsampling!**

**Sampla ner med 2x2 per steg med linjär  
filtrering.**

**Snabbt - utnyttjar GPU:s inbyggda filter!**

**Mindre exakt, mindre frihet än färltning.**

Ingemar  
Ragnemalm  
ingis@isy.liu.se

## **Flyttalsbuffrar**

**Framebuffer normalt RGBA 8888, 32  
bitar.**

**“Varför skulle man någonsin behöva  
mer?”**

**Men shaderprogrammen jobbar  
uttryckligen med flyttal.**

**En textur/FBO kan allokeras med  
flyttal!**

Ingemar  
Ragnemalm  
ingis@isy.liu.se

# Flyttalsbuffrar

Tre olika precisioner:

**FP32: s23.8**

**FP24: s16.7**

**FP16: s10.5**

**“double” finns (nästan) inte.  
FP32 kan ha dåliga prestanda.  
Använd FP16 när den räcker!**

Ingemar  
Ragnemalm  
ingis@isy.liu.se

## Allokering av flyttalsbuffer

```
glTexImage2D(GL_TEXTURE_2D, 0,  
GL_RGBA32F_ARB, texSize, texSize,  
0, GL_RGBA, GL_FLOAT, data);
```

(**data = NULL** skapar en tom textur.)

Ingemar  
Ragnemalm  
ingis@isy.liu.se

# High Dynamic Range

**Flyttalsbuffrar har enormt mycket större dynamik än 8 bitars heltal!**

**Detta ger möjlighet att rendera med högre dynamik. Dock kan man inte visa det på skärmen...**

Ingemar  
Ragnemalm  
ingis@isy.liu.se

# High Dynamic Range

**Postprocessing i shaders används för att ta tillvara den rikare informationen.**

**Typiska HDR-effekter:**

- Blooming (glow)
- Tone mapping (jfr histogramutjämning mfl histogramoperationer)

Ingemar  
Ragnemalm  
ingis@isy.liu.se

# Blooming

I vanlig grafik blir områden ljusare än 1  
“clampade” till 1.0, “urfrätta”.

Vi kan använda den “urfrätta” delen  
på bättre sätt än att kasta bort den!

Vi vill sprida ljuset till omgivningen för  
att ge ett “sken”. Lågpassfilter!

Ingemar  
Ragnemalm  
[ingis@isy.liu.se](mailto:ingis@isy.liu.se)

# Blooming

## Algoritm:

- Rendera scenen normalt, till flyttalstextur.
- Trunkera denna (till ny textur) så vi enbart behåller värden över 1.0.
  - Lågpassfiltrera detta.
  - Lägg ihop resultatet med originalscenen (med lämplig viktning).

Ingemar  
Ragnemalm  
[ingis@isy.liu.se](mailto:ingis@isy.liu.se)

# Blooming

**Normal rendering:** Görs hur man önskar.

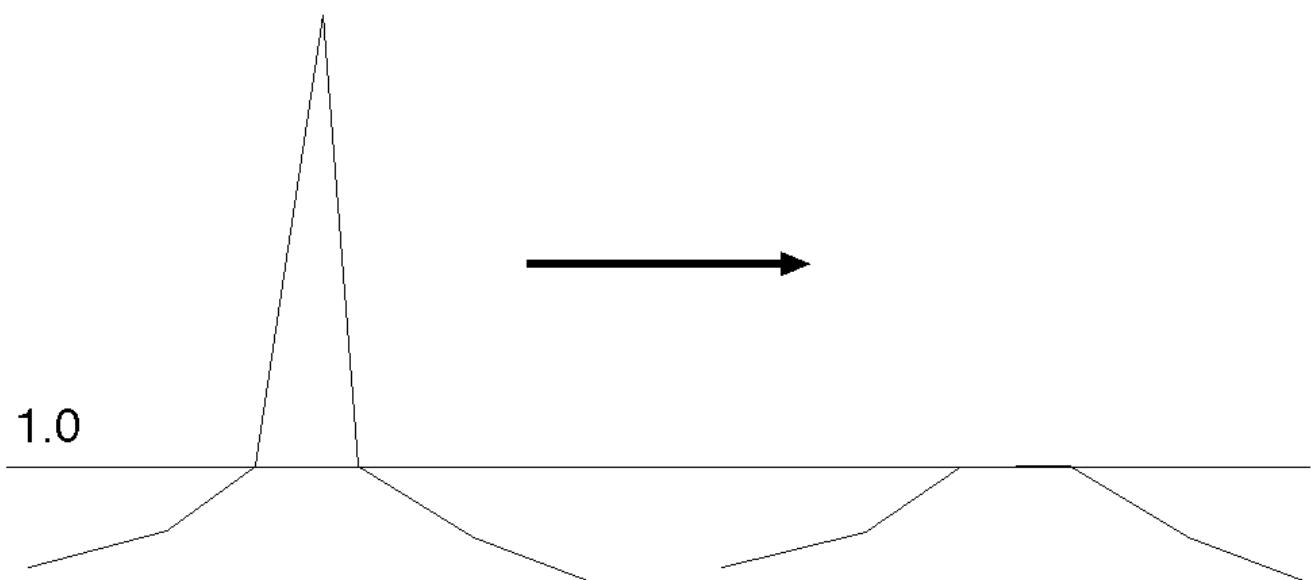
**Trunkering kan göras med enkel shader.**

**Lågpassfilter görs med flera pass LP-filter i shader.**

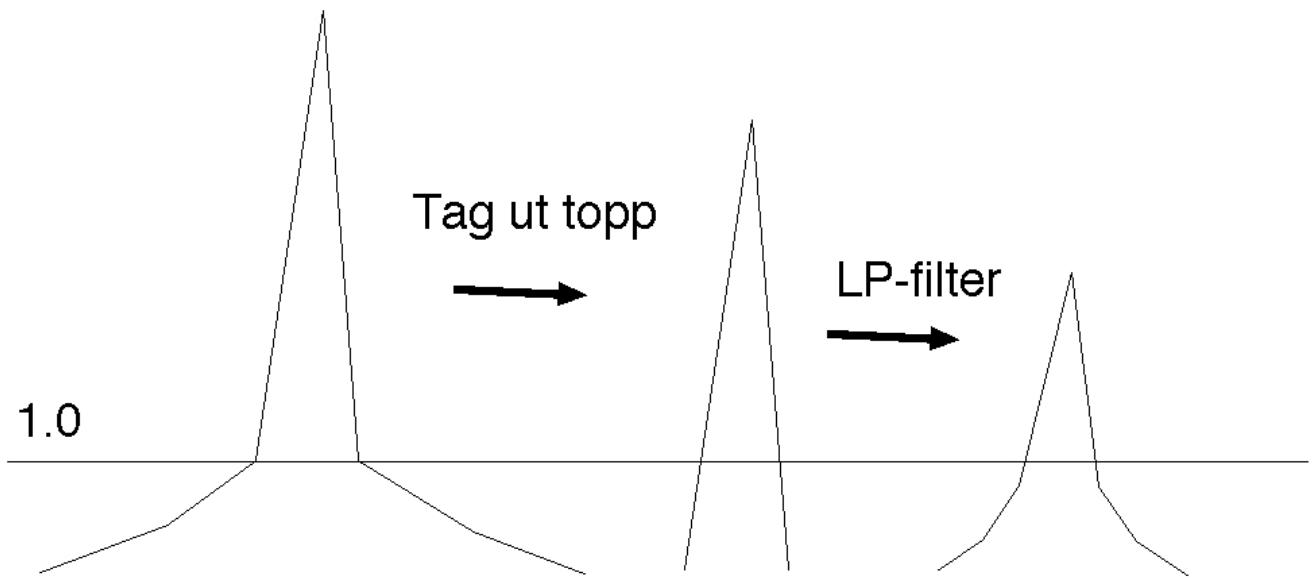
**Avslutande sammanviktning görs med en enkel shader eller blendning i fixed pipeline.**

Ingemar  
Ragnemalm  
[ingis@isy.liu.se](mailto:ingis@isy.liu.se)

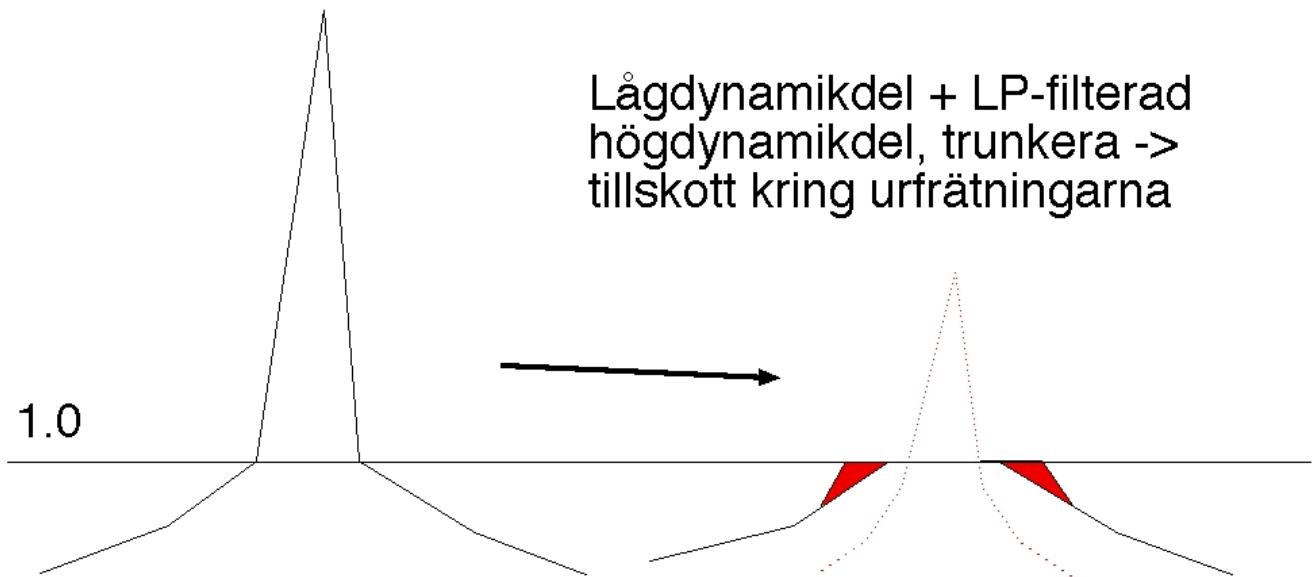
**Vanlig ”urfrätning” av ljusa områden**



# HDR-bloom



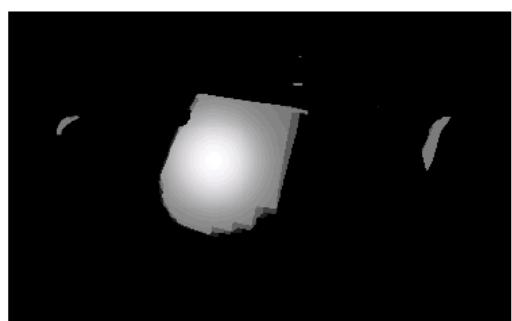
# HDR-bloom



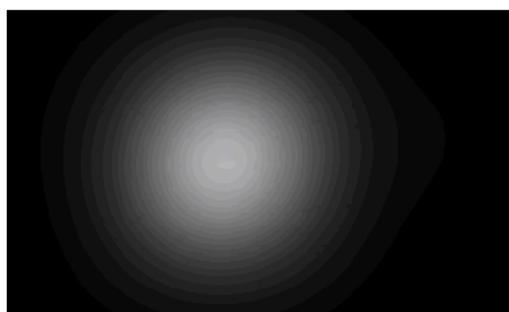
# Exemplet från boken



Normal rendering



Overflow



Overflow filterat (hårt!)



Resultat

## Observera i liveexemplet

**Texturen på tekannan blir inte suddig! Bara de ljusa områdena suddas ut!**

**Vissa artefakter syns p.g.a. dåliga filter**

# Exempel med kod

Kod för

- initiering
- shaders
- ping-pong

## Initiering

## Flera flyttalstexturer

```
// initialize texture 1
 glGenTextures(1, &tex1); // texture id
 glBindTexture(GL_TEXTURE_2D, tex1);
 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA16F_ARB, width, height, 0, GL_RGBA,
 GL_FLOAT, 0L); // NULL = Empty texture
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

# Initiering

## ...som får en matchande FBO

```
void initFBO(GLuint *fb0, GLuint *rb0, GLuint tex0)
{
    // create objects
    glGenFramebuffersEXT(1, fb0); // frame buffer id
    glGenRenderbuffersEXT(1, rb0); // render buffer id
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, *fb0);
    glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
        GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, tex0, 0);

    // Renderbuffer (KAN SKIPPAS I MÅNGA FALL)
    glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, *rb0);
    glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT,
        GL_DEPTH_COMPONENT24, width, height);
    // attach renderbuffer to framebuffer depth buffer
    glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
        GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, *rb0);
    printf("Framebuffer %d\n", *fb0);
}
```

## En del kod bör paketeras:

```
addShader = setupShader(&vertSource, &addFragSource);
```

```
GLhandleARB setupShader(const GLcharARB **vertSrc, const GLcharARB **fragSrc)
{
    GLhandleARB programObject; // the program used to update
    GLhandleARB fragmentShader, vertexShader;

    programObject = glCreateProgramObjectARB();

    fragmentShader = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);
    glShaderSourceARB(fragmentShader, 1, fragSrc, 0L);
    glCompileShaderARB(fragmentShader);
    glAttachObjectARB(programObject, fragmentShader);

    vertexShader = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
    glShaderSourceARB(vertexShader, 1, vertSrc, 0L);
    glCompileShaderARB(vertexShader);
    glAttachObjectARB(programObject, vertexShader);

    // Link the shader into a complete GLSL program.
    glLinkProgramARB(programObject);
    GLint progLinkSuccess;
    glGetObjectParameterivARB(programObject, GL_OBJECT_LINK_STATUS_ARB,
        &progLinkSuccess);
    if (!progLinkSuccess)
    {
        fprintf(stderr, "Shader could not be linked\n");
        exit(1);
        // Borde skicka ut hela infolog-felmeddelandet.
        // glGetShaderInfoLog eller glGetProgramInfoLog
    }

    return programObject;
}
```

# **Shaders:**

- **Enkel vertexshader**
  - Trunkera
  - Filtrera
- **Sammanslagning**

## **Enkel vertexshader**

```
varying vec3 norm;  
void main(void)  
{  
    gl_TexCoord[0] = gl_MultiTexCoord0;  
    gl_Position = ftransform();  
    norm = normalize(gl_NormalMatrix * gl_Normal);  
};
```

# Trunkering i shader

```
uniform sampler2D texUnit;  
uniform float texSize;  
void main(void)  
{  
    vec4 col = texture2D(texUnit, gl_TexCoord[0].xy);  
    gl_FragColor.r = max(col.r - 1.0, 0.0);  
    gl_FragColor.g = max(col.g - 1.0, 0.0);  
    gl_FragColor.b = max(col.b - 1.0, 0.0);  
}
```

## Filter

```
uniform sampler2D texUnit;  
uniform float texSize;  
void main(void)  
{  
    float offset = 2.0 / 256.0;  
    vec2 texCoord = gl_TexCoord[0].xy;  
    vec4 c = texture2D(texUnit, texCoord);  
    texCoord.x = texCoord.x + offset;  
    vec4 l = texture2D(texUnit, texCoord);  
    texCoord.x = texCoord.x - 2.0 * offset;  
    vec4 r = texture2D(texUnit, texCoord);  
    texCoord.x = texCoord.x - offset;  
    gl_FragColor = (c + c + l + r) * 0.25;  
}
```

1	2	1
---	---	---

# Sammanslagning

```
uniform sampler2D texUnit;
uniform sampler2D texUnit2;
void main(void)
{
    vec2 texCoord = gl_TexCoord[0].xy;
    vec4 a = texture2D(texUnit, texCoord);
    vec4 b = texture2D(texUnit2, texCoord);
    gl_FragColor = (a*0.3 + b*1.0);
}
```

## Körning av shader

Speciell funktion, lätt att köra många. Tar shader, texturer och ut-FBO som parametrar.

- Ställ in texturer och FBO
- Rensa GL\_PROJECTION och GL\_MODELVIEW
- Rita rektangel
- Ställ tillbaka

# Blooming med HDR

- Flera enkla shaders, appliceras i flera pass
- Flyttalsbuffrar för bästa resultat

# Bump mapping

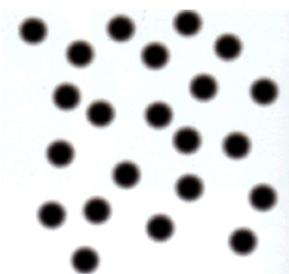
Principerna ingick i grundkursen.

- Mer detalj, implementation
- Koordinatsystem
- Normal mapping
- Utvidgning till mer avancerade metoder



## Bump mapping

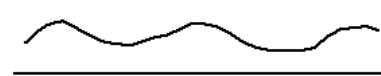
Simulates surface structure by manipulating the normal vector



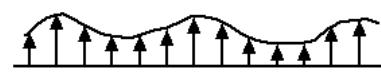
## Bump mapping - model



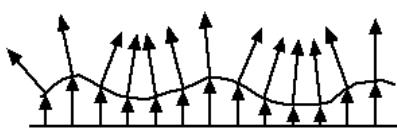
Surface with normal vectors



Bump map: scalar function of the texture coordinates



Modulate the surface by the bump function, along normal



Calculate new normals



Resulting normal vectors



## Bump mapping - practice

Input:

A point P, normal vector N

Texture coordinates  $s(P), t(P)$

Directions of texture coordinates  $V_s, V_t$

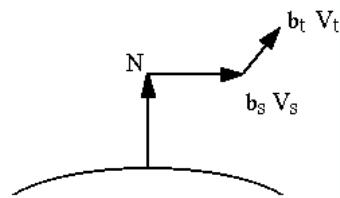
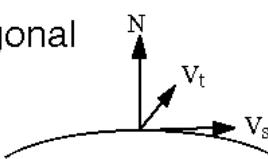
The bump function  $b(s,t)$

Calculate the partial derivative of the bump function,  $b_u$  and  $b_v$

$$N' = N + b_t * (V_s \times N) + b_s * (V_t \times N)$$

or, if  $V_s, V_t, N$  are orthogonal

$$N' = N + b_t * V_t + b_s * V_s$$



## Mer praktik

Var får vi  $V_s$  och  $V_t$  från? Vi har texturkoordinater  
men inget koordinatsystem!

Kryssa fram från normalvektorn? Men mot vad?

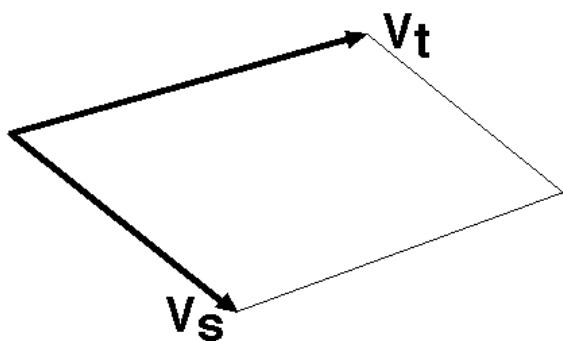
Beräkna från  $ds/dx$  och  $dt/dy$  i bilden? Då saknas  
z-variation!

# Mer praktik

Var får vi  $V_s$  och  $V_t$  från? Vi har texturkoordinater men inget koordinatsystem!

Kan skickas från värdprogram som parametrar till shader.

Enkelt för rektangel - bekvämt labbfall.



## Beräkning från vertexdata (enligt Angel)

I varje polygon kan man beräkna

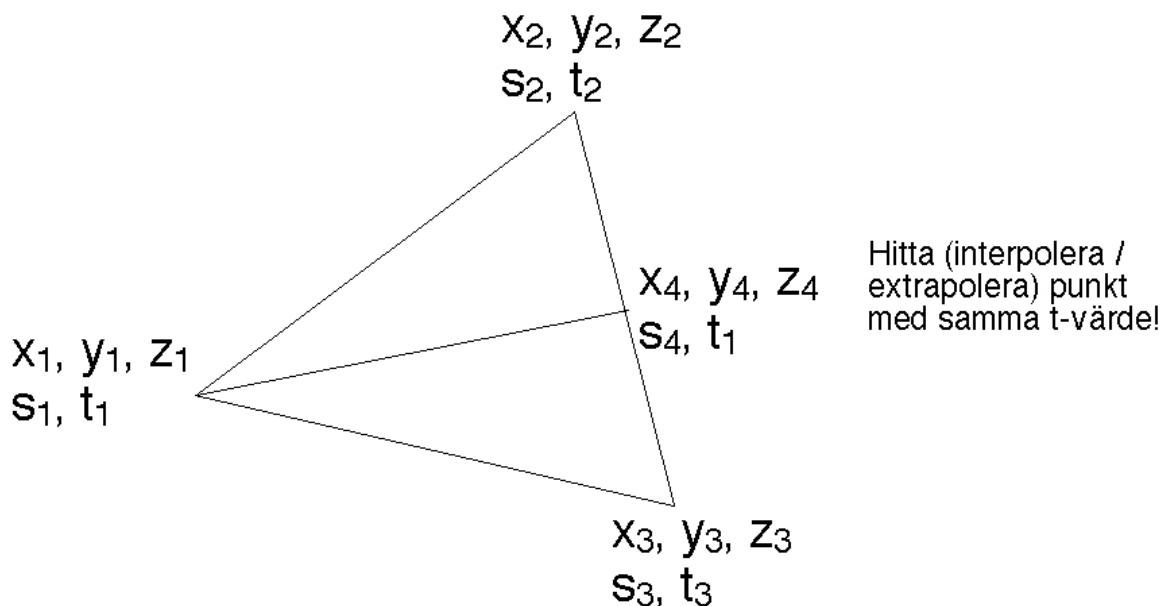
$$\begin{aligned} ds/dx, \quad ds/dy, \quad ds/dz \\ dt/dx, \quad dt/dy, \quad dt/dz \end{aligned}$$

och dessa ger

$$V_s = \begin{matrix} dx/ds \\ dy/ds \\ dz/ds \end{matrix} \quad V_t = \begin{matrix} dx/dt \\ dy/dt \\ dz/dt \end{matrix}$$

Kan beräknas enligt Dietrich (GPG 1)

# Beräkning av $dx/ds$ från vertexdata enligt Dietrich



Planets ekvation för  $(x, s, t)$  ger  $dx/ds$ . Motsv  $y, z$ .  
Hitta sedan punkt som ger linje med konstant  $s$ !

## Koordinatsystem

- Vy- och världskoordinater
  - Texturkoordinater
  - Tangentkoordinater

Ljuskälla fås ofta i vykoordinater  
Bumpmappen ges i texturkoordinater  
Normalvektor i modellkoordinater eller  
vykoordinater

**Vi måste kunna konvertera mellan  
koordinatsystemen! Ljussättning skall göras med  
vektorer i samma koordinatsystem.**

# Koordinatsystem

Modell till vy: gl\_NormalMatrix

$$P_s = \text{gl_NormalMatrix} * V_s$$

$$P_t = \text{gl_NormalMatrix} * V_t$$

Vy till textur:

$$M_{vt} = \begin{bmatrix} P_s \\ P_t \\ n \end{bmatrix} = \begin{bmatrix} P_{sx} & P_{sy} & P_{sz} \\ P_{tx} & P_{ty} & P_{tz} \\ n_x & n_y & n_z \end{bmatrix}$$

# Koordinatsystem

$P_s$  är *tangentvektorn* (kallas ofta  $t$  i andra texter)

$P_t$  kallas ibland inkorrekt *binormal*, men är egentligen *bitangent* ( $b$  i andra texter)

Texturrymd = bas med vektorer längs texturvariationer

Tangentrymd = ortonormal bas i texturrymd

Tangentrymd ofta bra approximation till texturrymd

## **Lite mer definitioner**

**bumpmap = bild med höjdvärdet**

**normal map = bild med förberäknade  
normalvektorer**

**(Förvirring mellan dessa två förekommer)**

## **Beräkning av modifierad normalvektor (vykoordinater)**

$$b_s = \frac{db}{ds}$$

$$b_t = \frac{db}{dt}$$

$$\mathbf{n}' = \mathbf{n} + b_s \cdot \mathbf{P_s} + b_t \cdot \mathbf{P_t} \quad ("in")$$

eller

$$\mathbf{n}' = \mathbf{n} - b_s \cdot \mathbf{P_s} - b_t \cdot \mathbf{P_t} \quad ("ut")$$

Förutsättertangentrymd. Texturrymd blir:

$$\mathbf{n}' = \mathbf{n} + b_s \cdot \mathbf{P_{txn}} + b_t \cdot \mathbf{P_{sxn}} \quad ("in")$$

# Beräkning av modifierad normalvektor (texturkoordinater)

$$b_s = db/ds$$

$$b_t = db/dt$$

$$\mathbf{n}' = \begin{bmatrix} b_s \\ b_t \\ 1 \end{bmatrix} + \text{normering}$$

Jättelått! MEN, ljus och vyriktning måste då transformeras till texturkoordinater!

$$\mathbf{L}_t = M_{vt} * \mathbf{L}$$

## Normalmappning

Förberäkna  $b_s$  och  $b_t$ , spara i bild!

$$-b_s = b[s, t] - b[s+1, t]$$

$$-b_t = b[s, t] - b[s, t+1]$$

1

Normera!

# Lagring i textur

”Scale and bias” här också:

$$\begin{aligned} R &= (x+1)/2 \\ G &= (y+1)/2 \\ B &= (z+1)/2 \end{aligned}$$

Hämtning ur textur:

$$\begin{aligned} x &= 2R - 1 && (\text{Varför?}) \\ y &= 2G - 1 \\ z &= 2B - 1 \end{aligned}$$

$$n_t = (x, y, z)$$

Bumpmappen kan läggas i alpha-komponenten!

## Exempel från boken

Jonas Lindmarks projekt



# Bättre än bump mapping

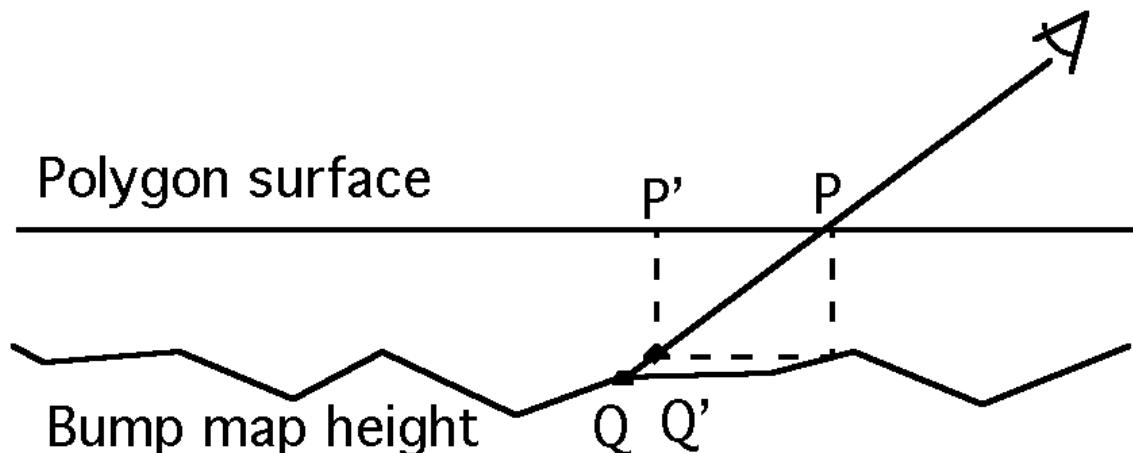
Bump mapping är “platt”, ser fel ut när man betraktar i lite vinkel.

Bättre metoder:

- Parallax Mapping
- Relief Mapping/Parallax Occlusion Mapping
- Per-pixel displacement mapping

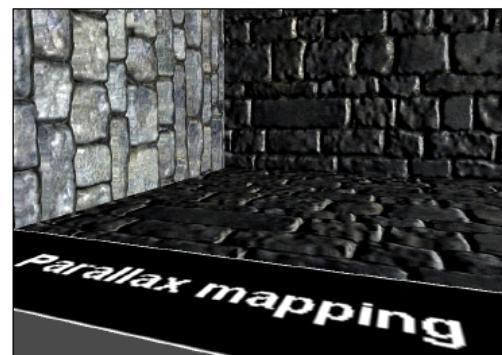
## Parallax Mapping

Enklaste metoden, gör ett grovt “språng” i texturrymden.



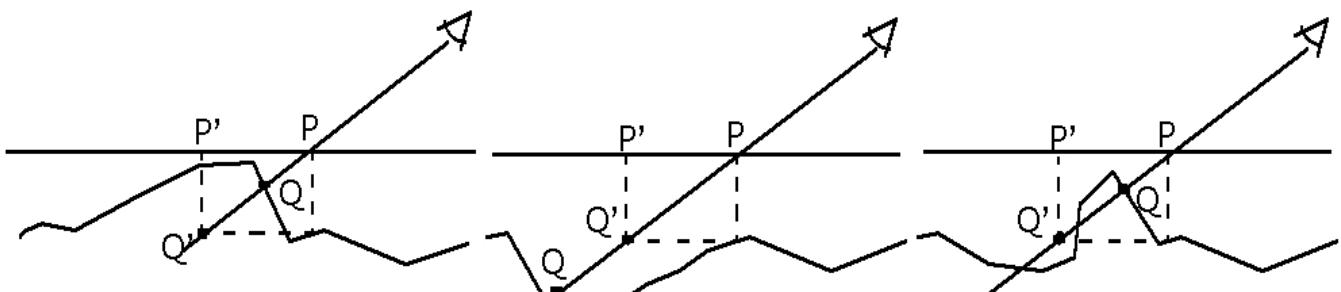
# Parallax Mapping

I många fall en god förbättring.



# Parallax Mapping

Antar långsam variation i bump mappen.



# Parallax Mapping

Konstiga resultat i branta vinklar eller högfrekvent bump map.



# Parallax Mapping

$$T_n = T_0 \pm b(s, t) \cdot V_{xy} / V_z$$

Felet växer vid små  $V_z$ !

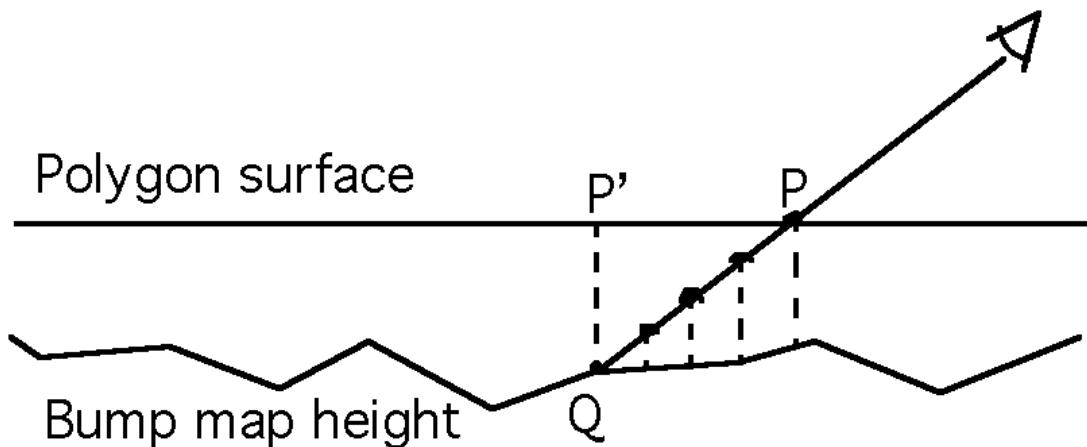
Variant: Offset limiting:

$$T_n = T_0 \pm b(s, t) \cdot V_{xy}$$

Tar helt enkelt bort  $V_z$ !

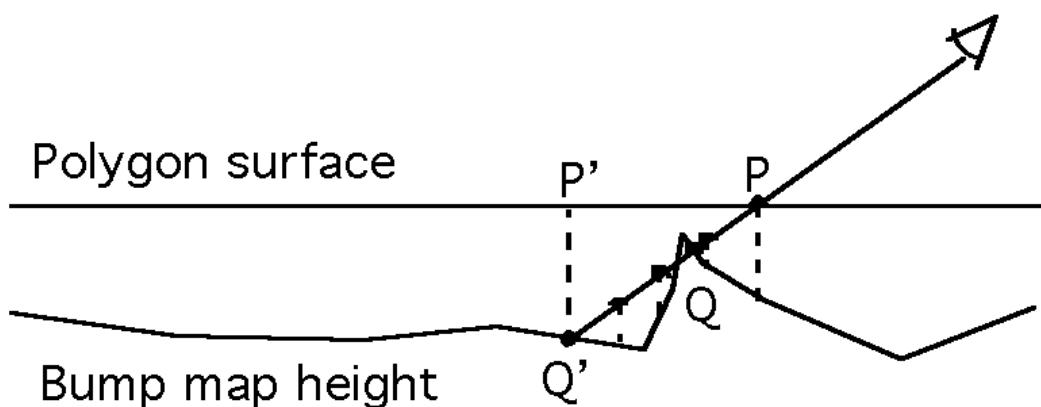
# Relief Mapping

Söker i flera steg



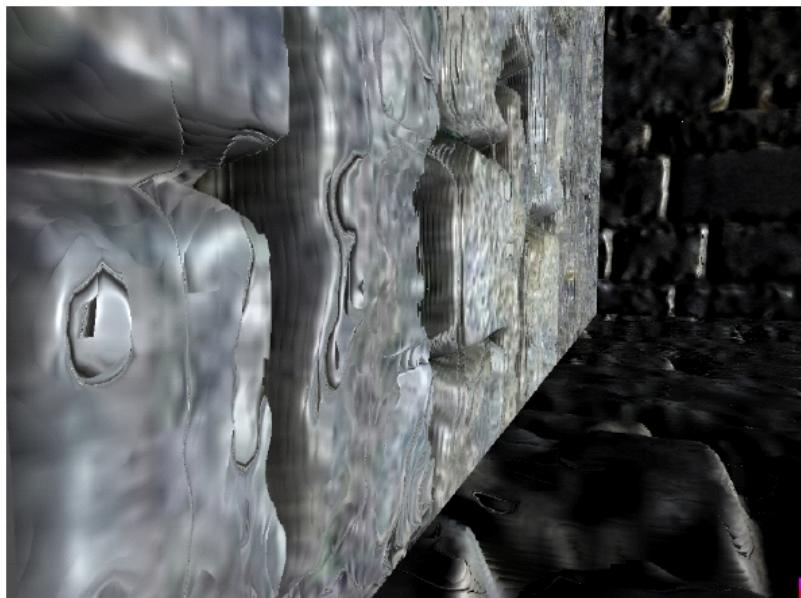
# Relief Mapping

Missar ibland om stegen är för stora



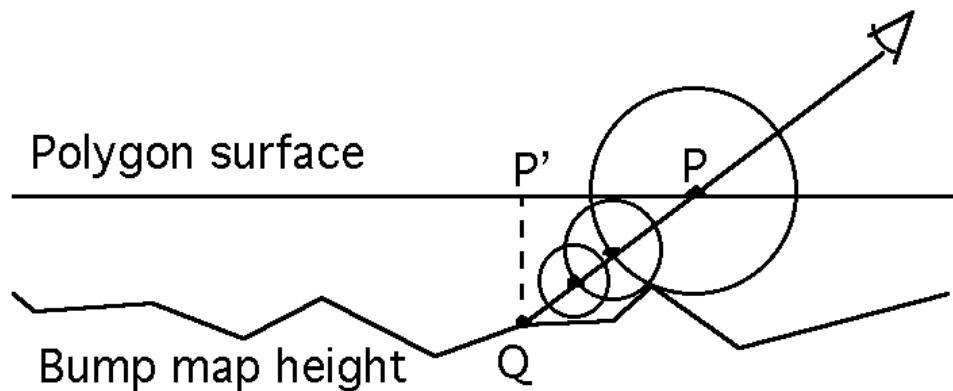
## **Variant: Parallax Occlusion Mapping**

**Liknande algoritm, uppvisar enligt Lindmark betydande fel.**



## **Per-pixel Displacement Mapping**

**Mycket exakt metod som ger fina  
resultat**



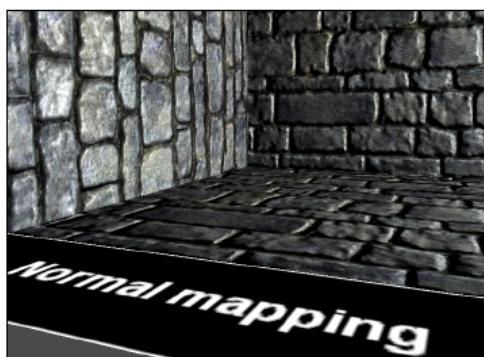
# Per-pixel Displacement Mapping

Använder Euklidisk avståndstransform

Algoritm för att effektivt beräkna  
avståndsvärden i samplad rymd

Uppfanns av PE Danielsson 1980

# Per-pixel Displacement Mapping



# Vad behöver ett spel då?

- Platt texturmappning: Räcker inte alls.
- Bump mapping: Räcker långt!
- Parallax mapping: Kostar inte mycket när man har bump mapping
- Per-pixel displacement mapping: Visst blir det fint, men *behövs* det verkligen?

Får jag en snabb och stabil PPDM gratis tar jag det nog, men annars är bump mapping och parallax mapping frestande, kostnadseffektiva.

## Inlämningsuppgifter

7. Varför trunkerar man en del av original-bilddata i HDR-bloom? Varför tar man inte hela bilden?
8. Du skall applicera ett 3x3-filter på en bild. Det kan separeras till ett 1x3 och ett 3x1. Borde det bli snabbare? Varför?
9. På vilket sätt får vi bättre parallax mapping med offset limiting?

OBS! Deadline *måndag*!