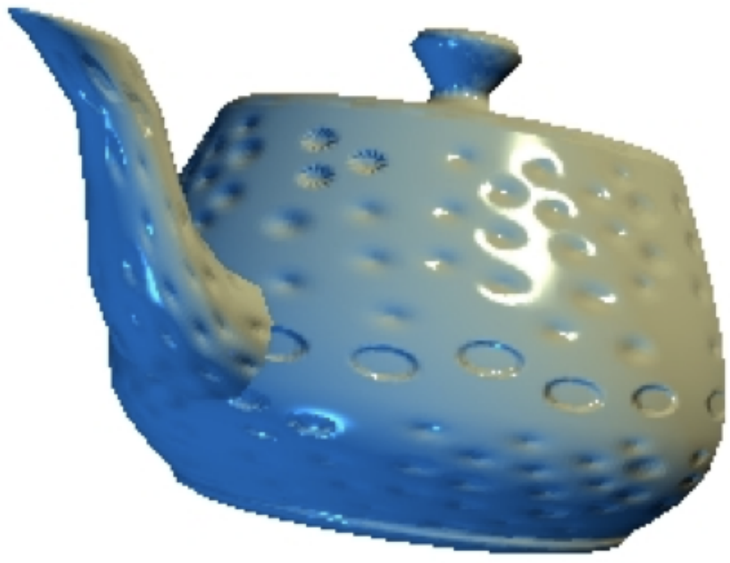
Bump mapping

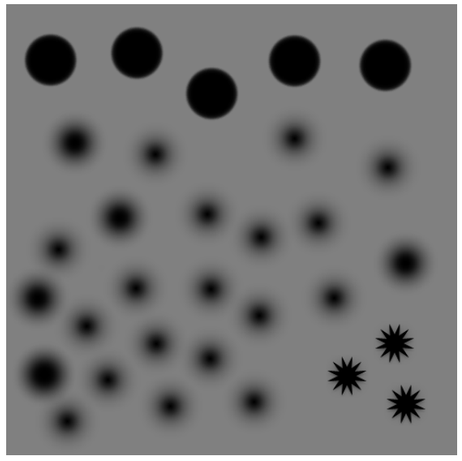
Principerna ingick i grundkursen.

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

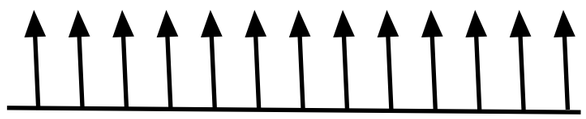
Bump mapping

Simulates surface structure by manipulating the normal vector





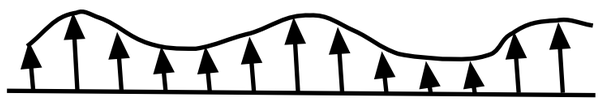
Bump mapping - model



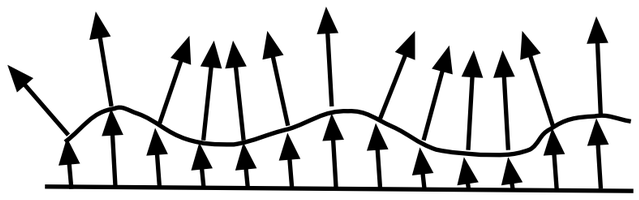
Surface with normal vectors



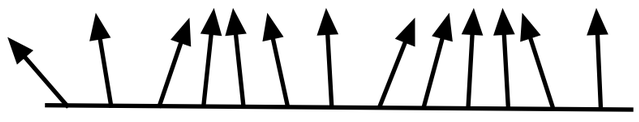
Bump map: scalar function of the texture coordinates



Modulate the surface by the bump function, along normal



Calculate new normals



Resulting normal vectors

Bump mapping - practice

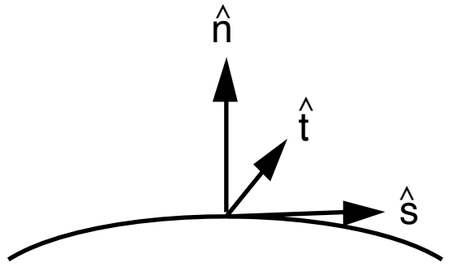
Input:
A point p, normal vector $\hat{n}$
Texture coordinates s(p), t(p)
Directions of texture coordinates $\hat{s}$, $\hat{t}$
The bump function b(s,t)

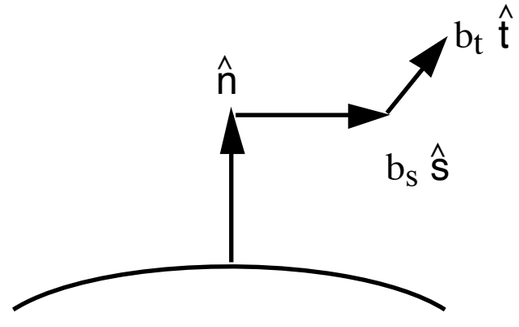
Calculate the partial derivative of the bump function, b_s and b_t

$\hat{n}' = \hat{n} + b_t * (\hat{s} \times \hat{n}) + b_s * (\hat{t} \times \hat{n})$

or, if $\hat{s}$, $\hat{t}$, $\hat{n}$ are orthogonal

$\hat{n}' = \hat{n} + b_t * \hat{t} + b_s * \hat{s}$





Koordinatsystemet $\hat{s}$ och $\hat{t}$

Var får vi $\hat{s}$ och $\hat{t}$ 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!

Fuskmetod

Kryssa fram från normalvektorn mot vad tusan som helst!

$$\hat{s} = \hat{x} \times \hat{n} / |\hat{x} \times \hat{n}|$$
$$\hat{t} = \hat{n} \times \hat{s}$$

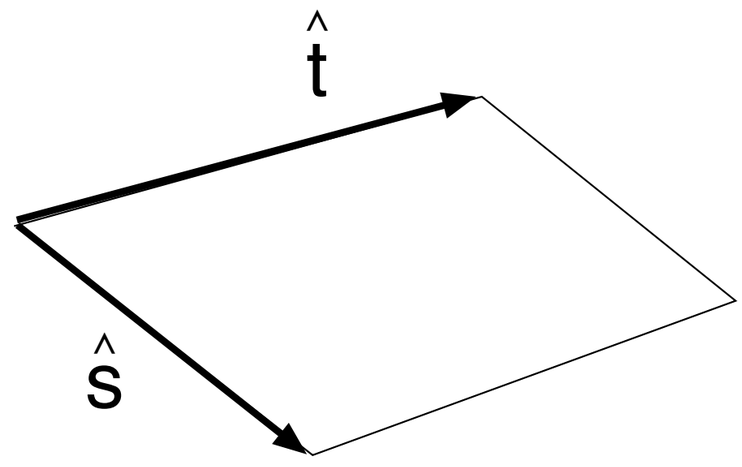
Funkar för vissa fall. (Brus i synnerhet.)

Bättre kan vi!

Enkel geometri

Men om vi nu vill ha mer korrekta $\hat{s}$ och $\hat{t}$, vad gör vi då?

Enkelt för rektangel - bekvämt labbfall.



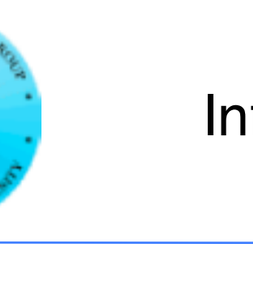
Lengyel's metod

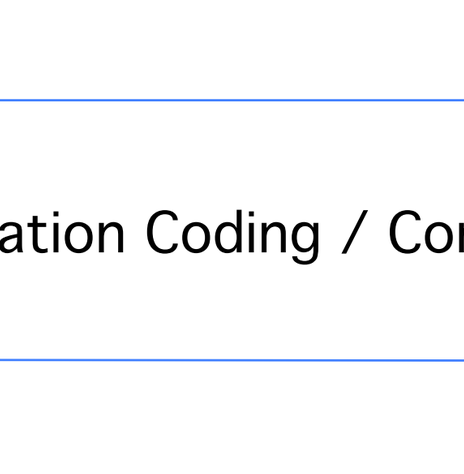
Härled genom steg längs $\hat{s}$ och $\hat{t}$ i xyz-rymden (modellkoordinater).

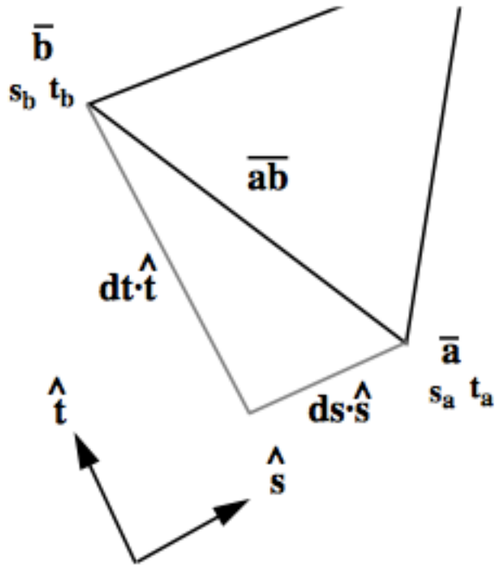
Rak och snygg metod genom matrisalgebra.

Uttryck två sträckor som funktion av $\hat{s}$ och $\hat{t}$, finn inversen!

Lengyel's metod







Givet en triangel med texturkoordinater,
finn basvektorer för texturkoordinater

Lengyel's metod

i programkod - ganska enkelt!

```
float ds1 = sb - sa; float ds2 = sc - sa;
float dt1 = tb - ta; float dt2 = tc - ta;
vec3 s, t;
float r = 1/(ds1 * dt2 - dt1 * ds2);
s = (ab * dt2 - ac * dt1) * r;
t = (ac * ds1 - ab * ds2) * r;
```

OBS! Vektoroperationer!

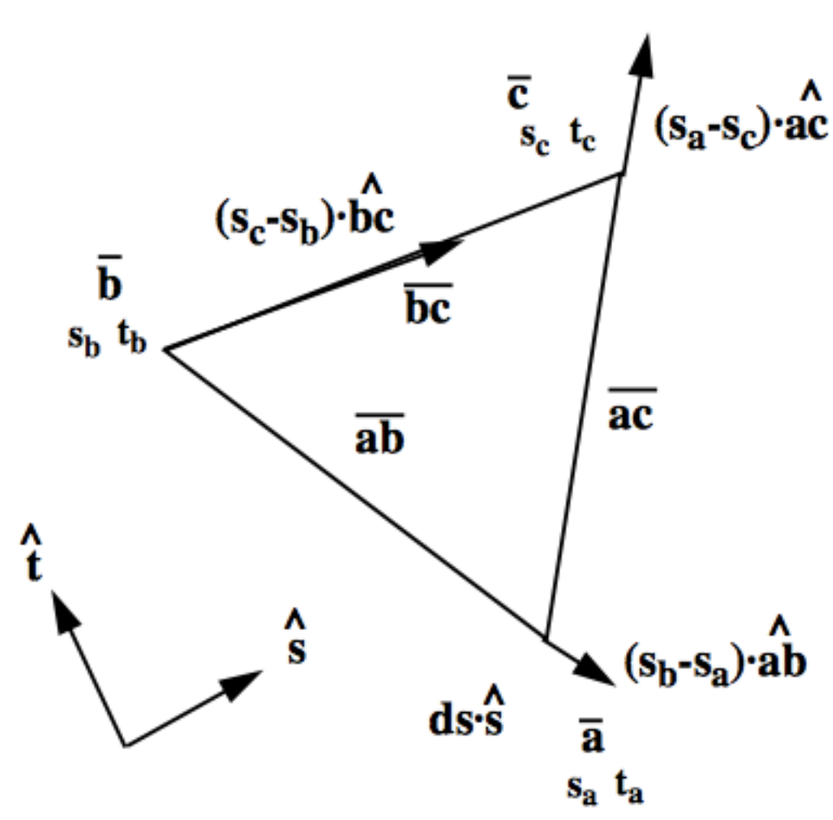
Approximativ metod

Låt varje sida av polygoner bidra till $\hat{s}$ och $\hat{t}$ beroende på deras s- och t-variation!

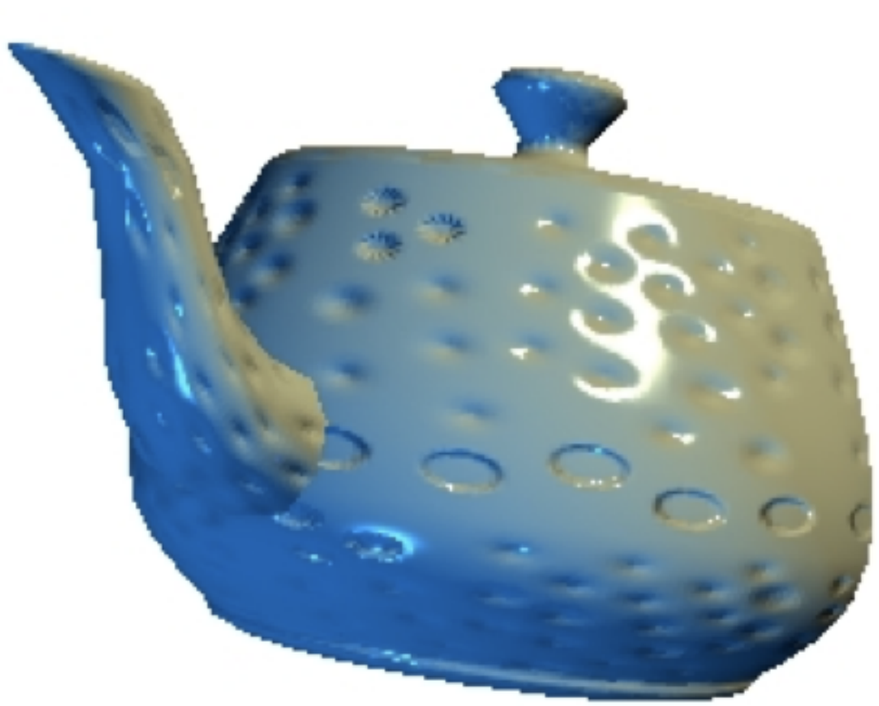
Varje sidas bidrag till $\hat{s}$ = sidans riktning (normerad gånger variationen i $\hat{s}$. Summera!

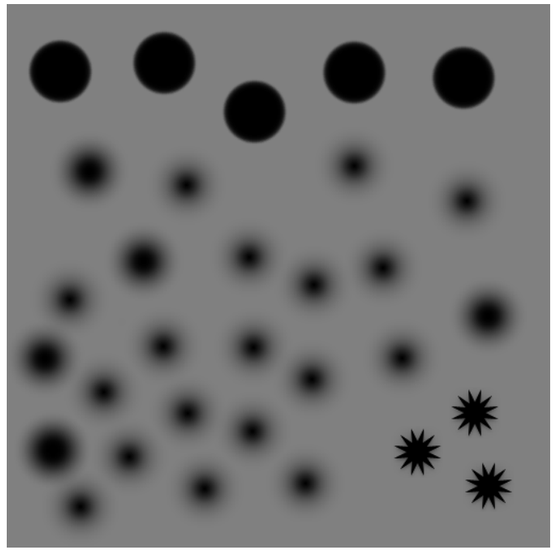
Approximativ metod

$$s_{ab} = \frac{\boldsymbol{ab}}{|\boldsymbol{ab}|}(s_b - s_a)$$



Båda metoderna ger bra resultat för komplicerade modeller!





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: normalmatrisen

$$p_s = \text{normalMatrix} * s$$
$$p_t = \text{normalMatrix} * t$$
$$n_v = \text{normalMatrix} * n$$

Vy till textur:

$$M_{vt} = \begin{bmatrix} p_s \\ p_t \\ n_v \end{bmatrix} = \begin{bmatrix} p_{sx} & p_{sy} & p_{sz} \\ p_{tx} & p_{ty} & p_{tz} \\ n_{vx} & n_{vy} & n_{vz} \end{bmatrix}$$

Koordinatsystem

p_s är tangentvektorn (kallas ofta t i andra texter)
p_t bitangent (inte binormal)

Texturrymd = bas med vektorer längs texturvariationer

Tangentrymd = ortonormal bas i texturrymd

Tangentrymd ofta bra approximation till texturrymd

Lite mer definitioner

bumpmap = bild med höjdvärden

normal map = bild med förberäknade normalvektorer

(Förvirring mellan dessa två förekommer)

Beräkning av modifierad normalvektor (vykoordinater)

$b_s = db/ds$
$b_t = db/dt$

$$n' = n + b_s \cdot P_s + b_t \cdot P_t \qquad (\text{”in”})$$

eller

$$n' = n - b_s \cdot P_s - b_t \cdot P_t \qquad (\text{”ut”})$$

Förutsätter tangentrymd. Texturrymd blir:

$$n' = n + b_s \cdot P_t \times n + b_t \cdot P_s \times n \qquad (\text{”in”})$$

Beräkning av modifierad normalvektor (texturkoordinater)

$$b_s = db/ds$$
$$b_t = db/dt$$

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

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

$$l_t = M_{vt} * l$$

Normalmappning

Förberäkna b_s och b_t, spara i bild!

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

Normera!

Lagring i textur

"Scale and bias" här också:

$$R = (b_s + 1)/2$$
$$G = (b_t + 1)/2$$

(Varför?)

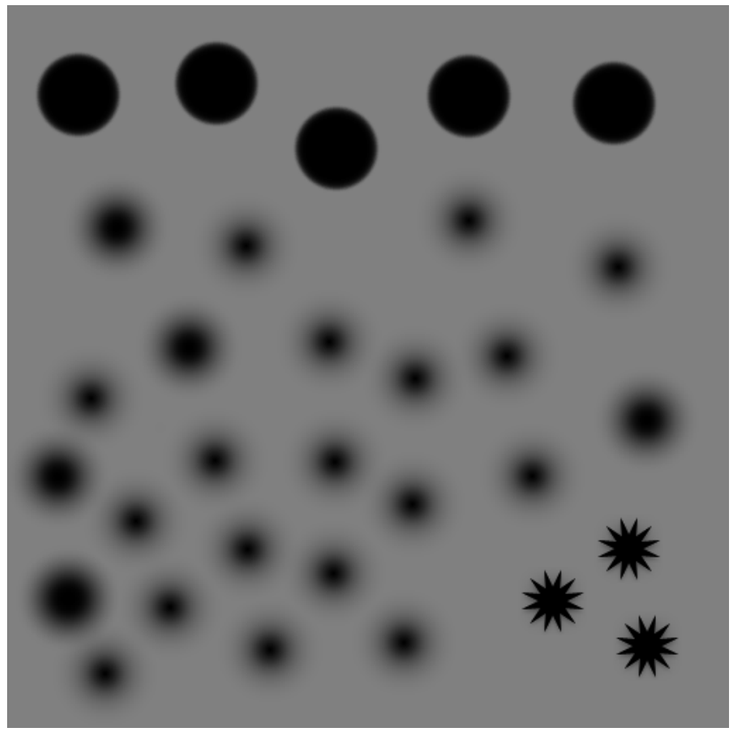
Hämtning ur textur:

$$b_s = 2R - 1$$
$$b_t = 2G - 1$$

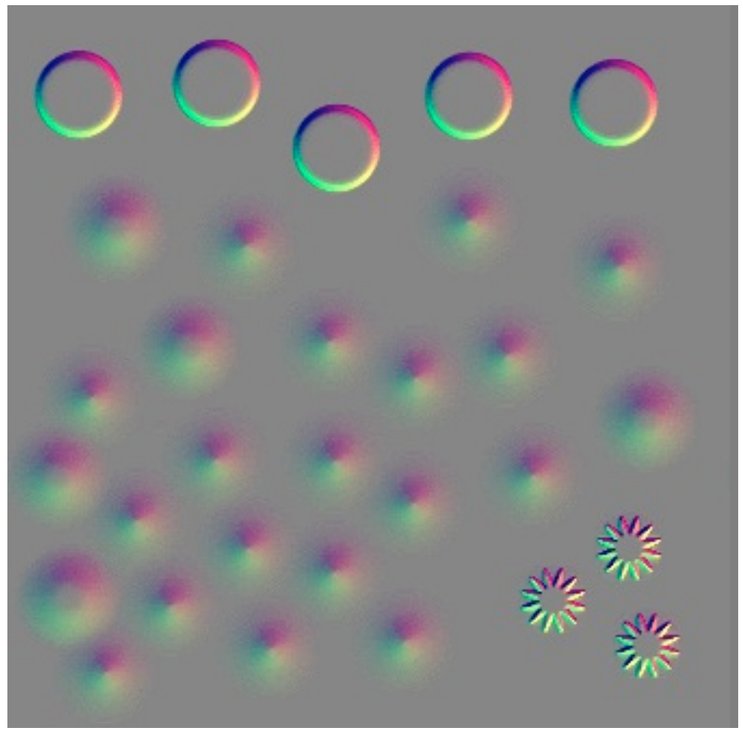
Exempel på normal map (”normalkarta”?)

Bild borttagen av
copyrightskäl

Bump map som användes i exemplet



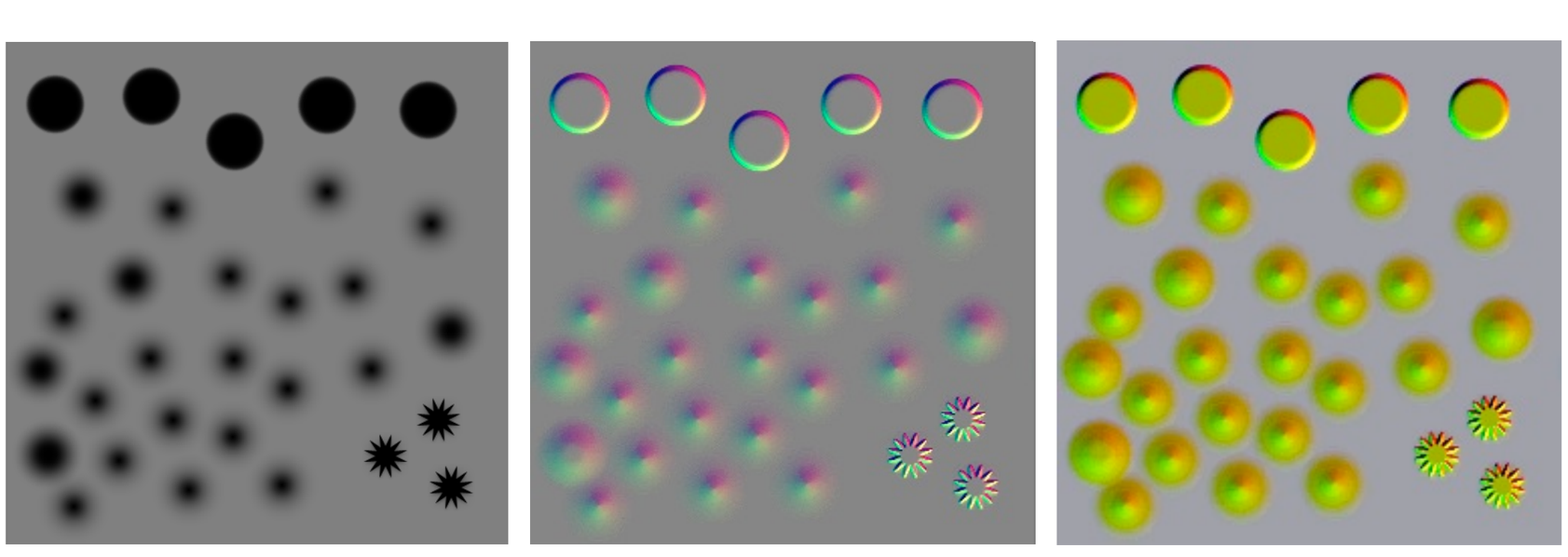
Bump map



Normal map

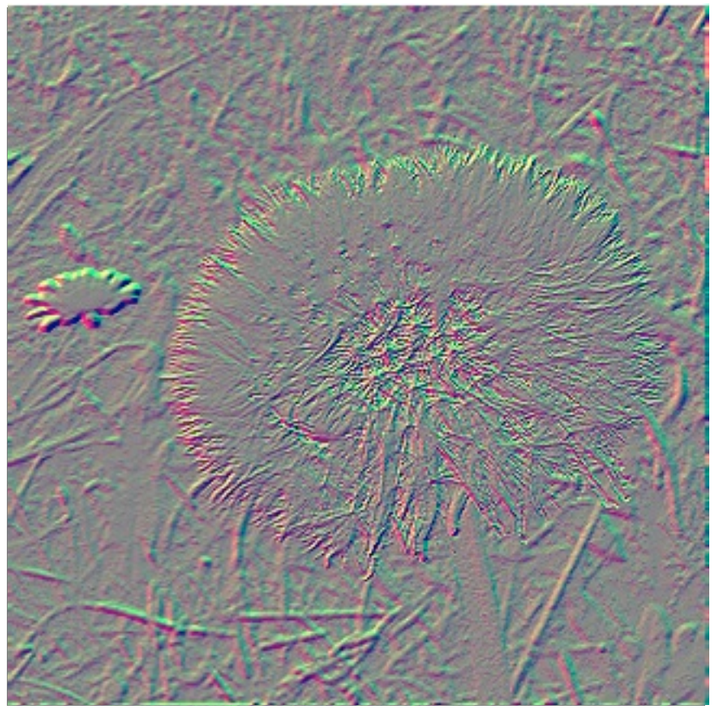
Bumpmappen kan läggas i blå- eller alpha-komponenten!

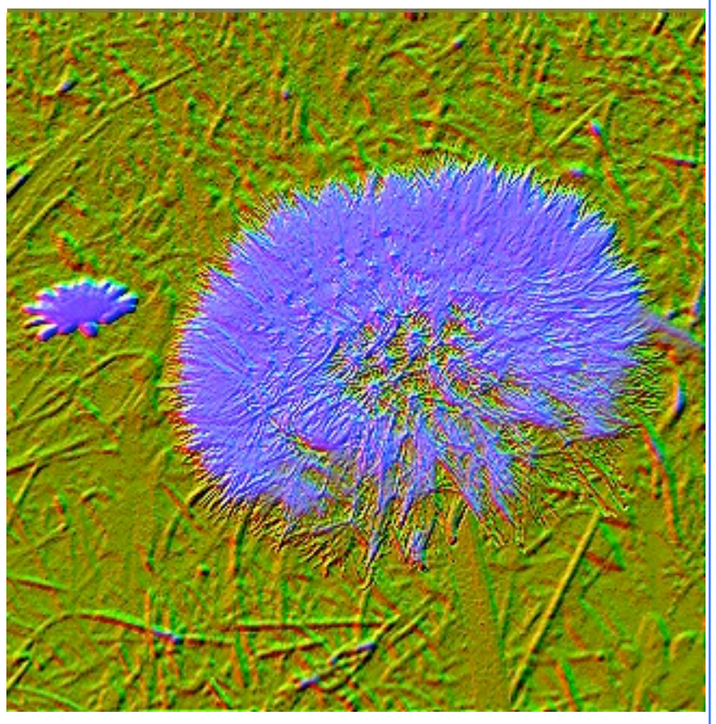
➪ Hybrid bump/normal-map



Bump map Normal map Hybrid



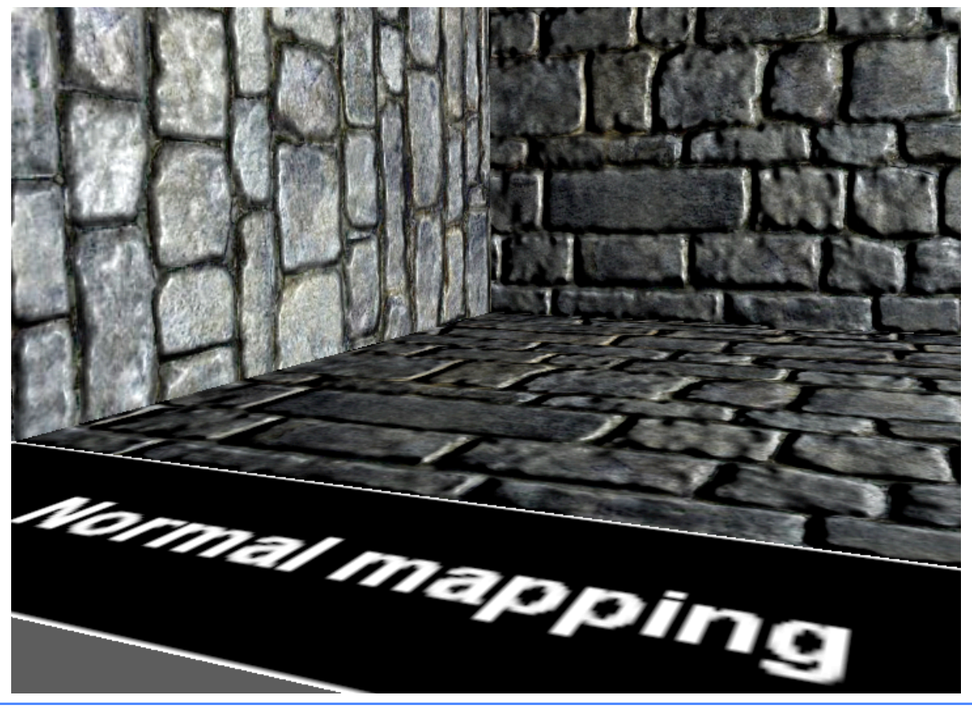




Exempel från boken

Jonas Lindmarks projekt





Bättre än bump mapping

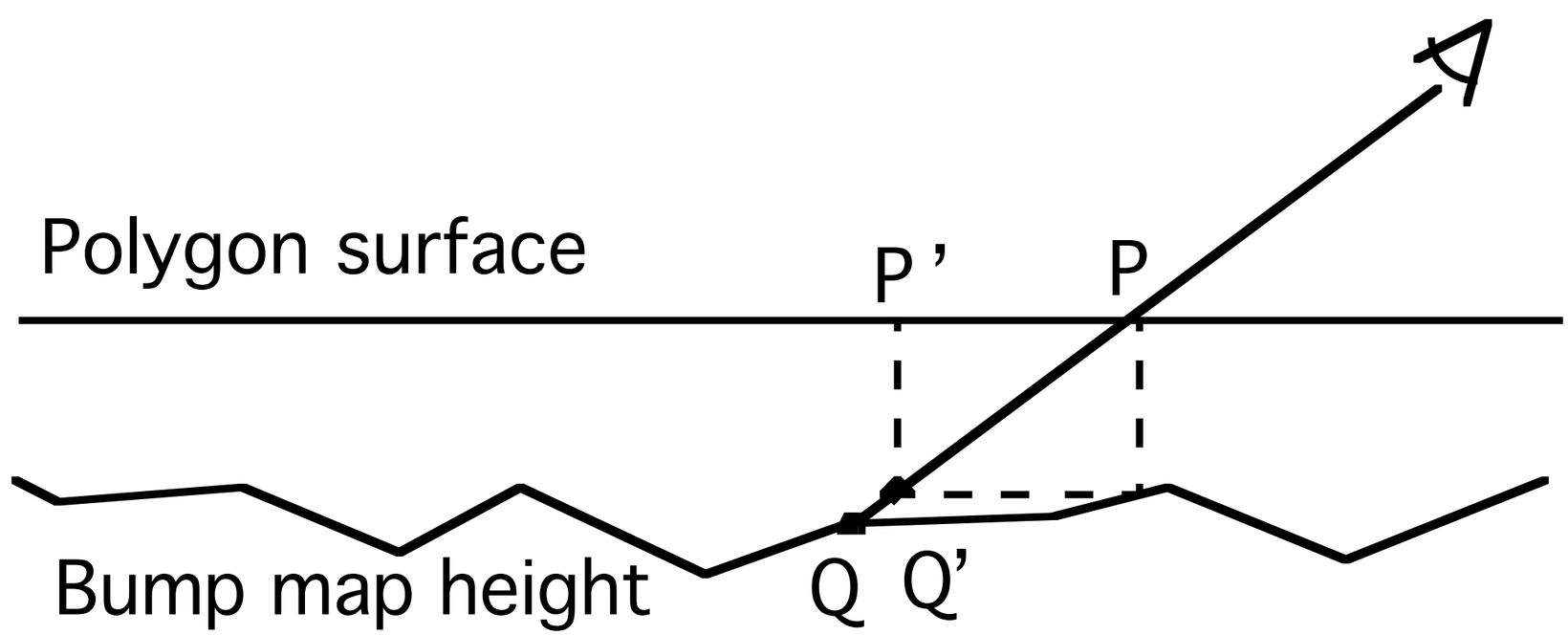
Bump mapping är “platt”, ser fel ut när man betraktar i lite vinkel.

Bättre metoder:

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

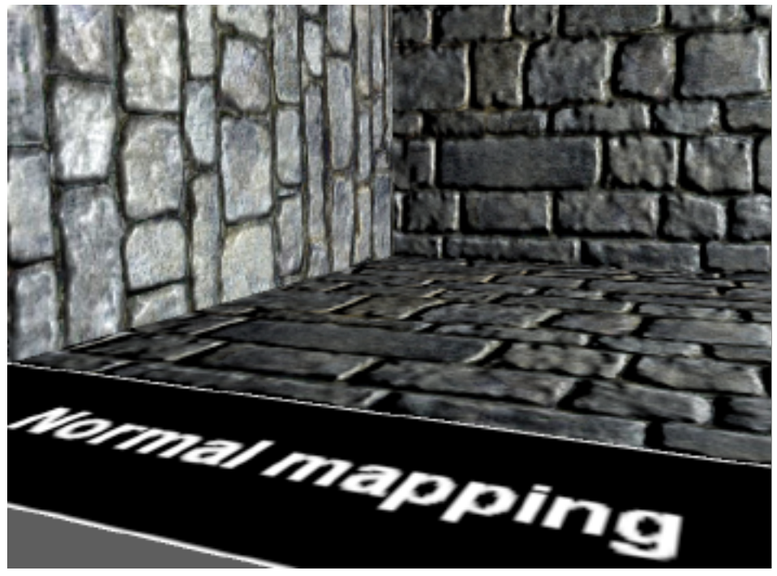
Parallax Mapping

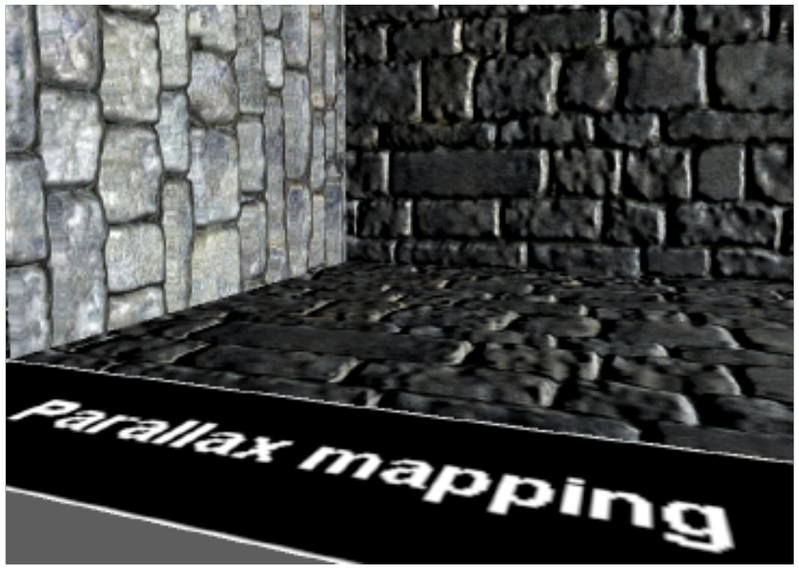
Enklaste metoden, gör ett grovt “språng” i texturrymden.



Parallax Mapping

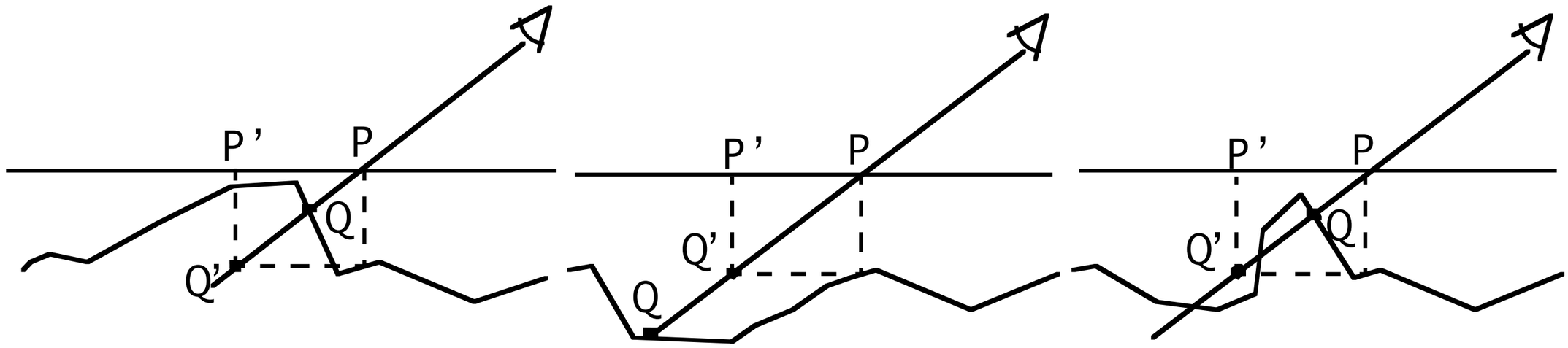
I många fall en god förbättring.





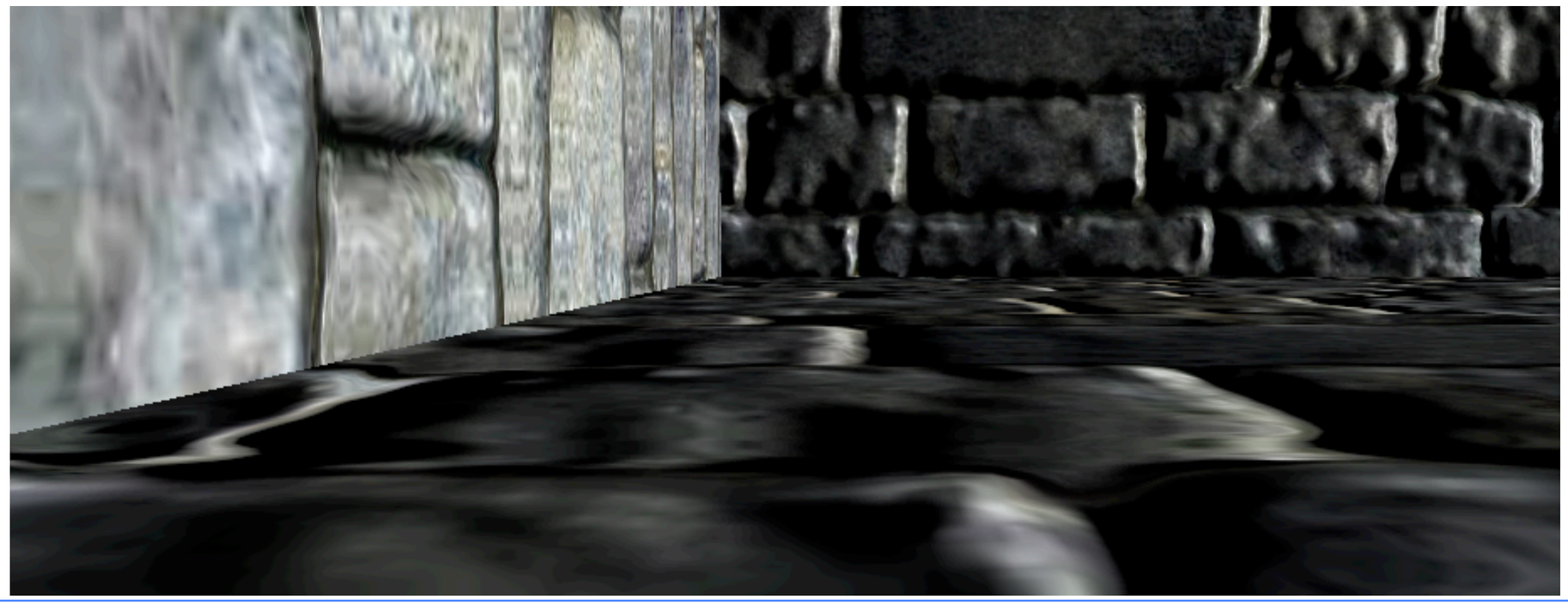
Parallax Mappings svaghet

Antar långsam variation i bump mappen.



Hur det yttrar sig

Konstiga resultat i branta vinklar eller högfrekvent bump map.



Minska problemet

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

Felet växer vid små Vz!

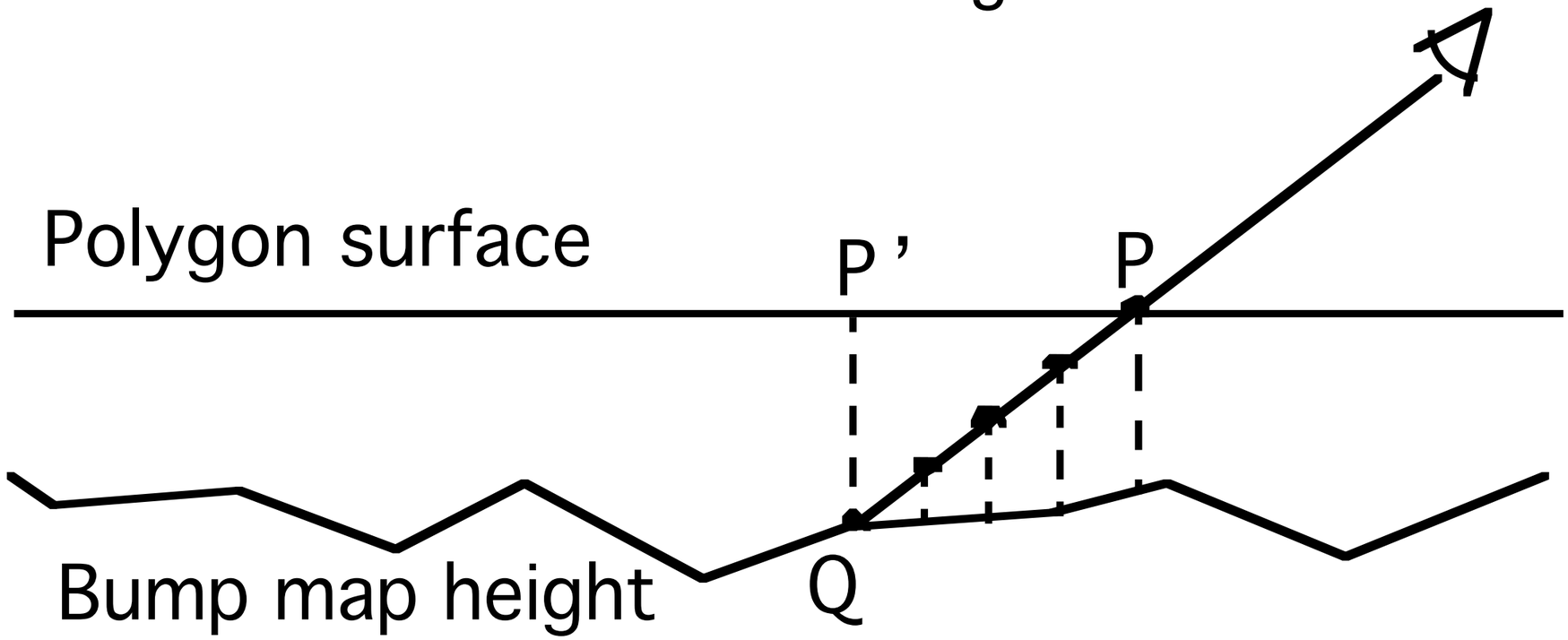
Variant: Offset limiting:

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

Tar helt enkelt bort V_z!

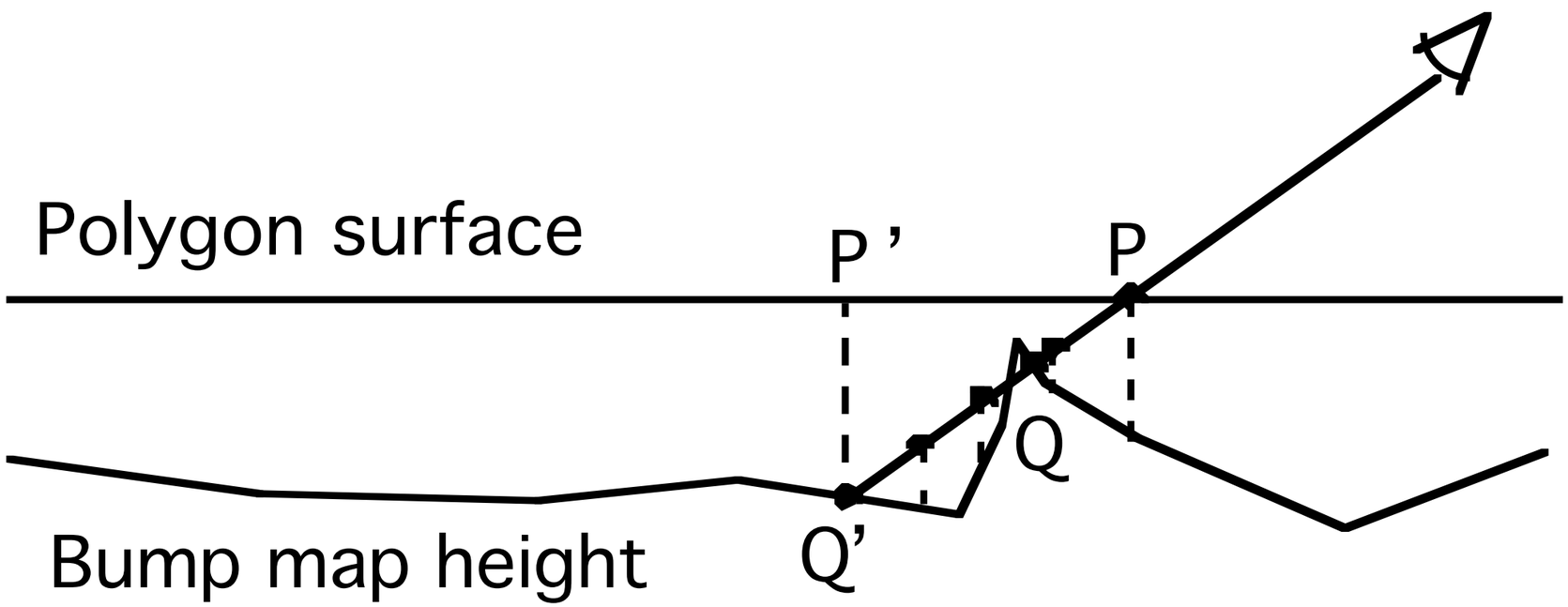
Relief Mapping

Söker i flera steg



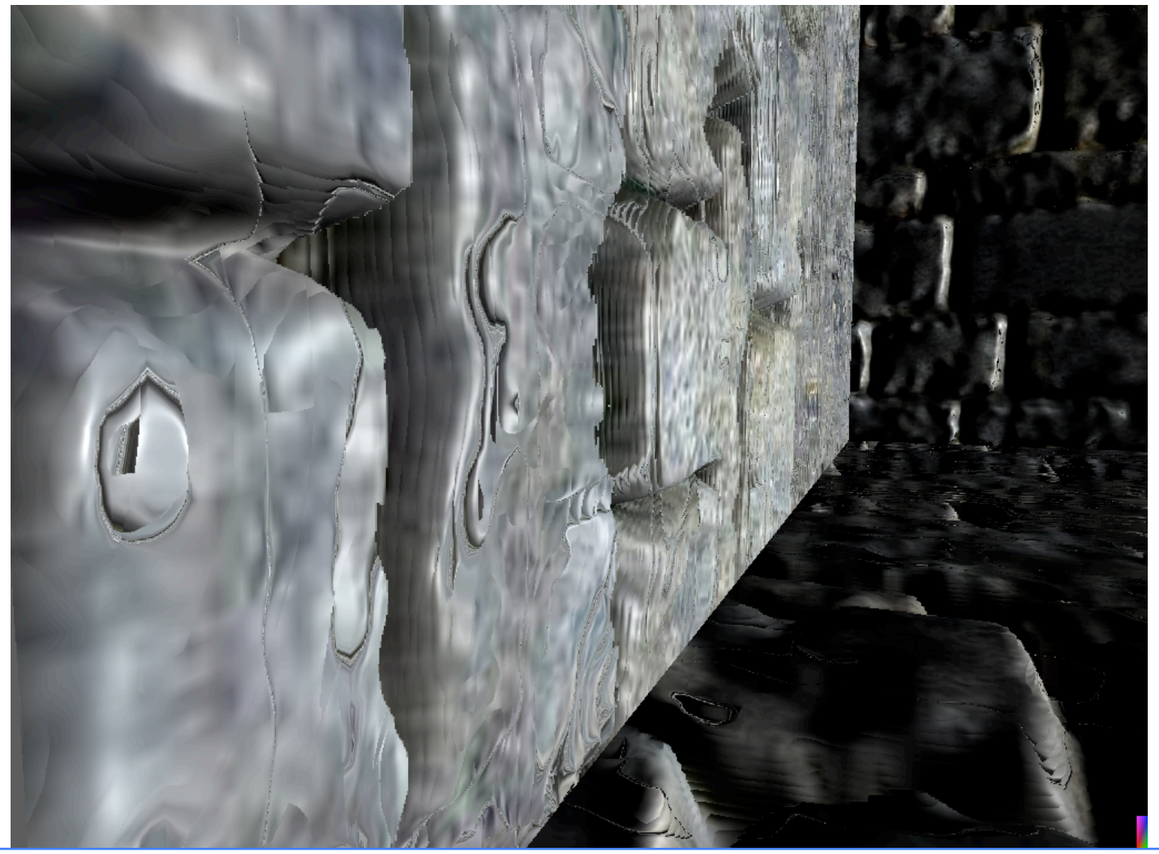
Relief Mappings svaghet

Missar ibland om stegen är för stora



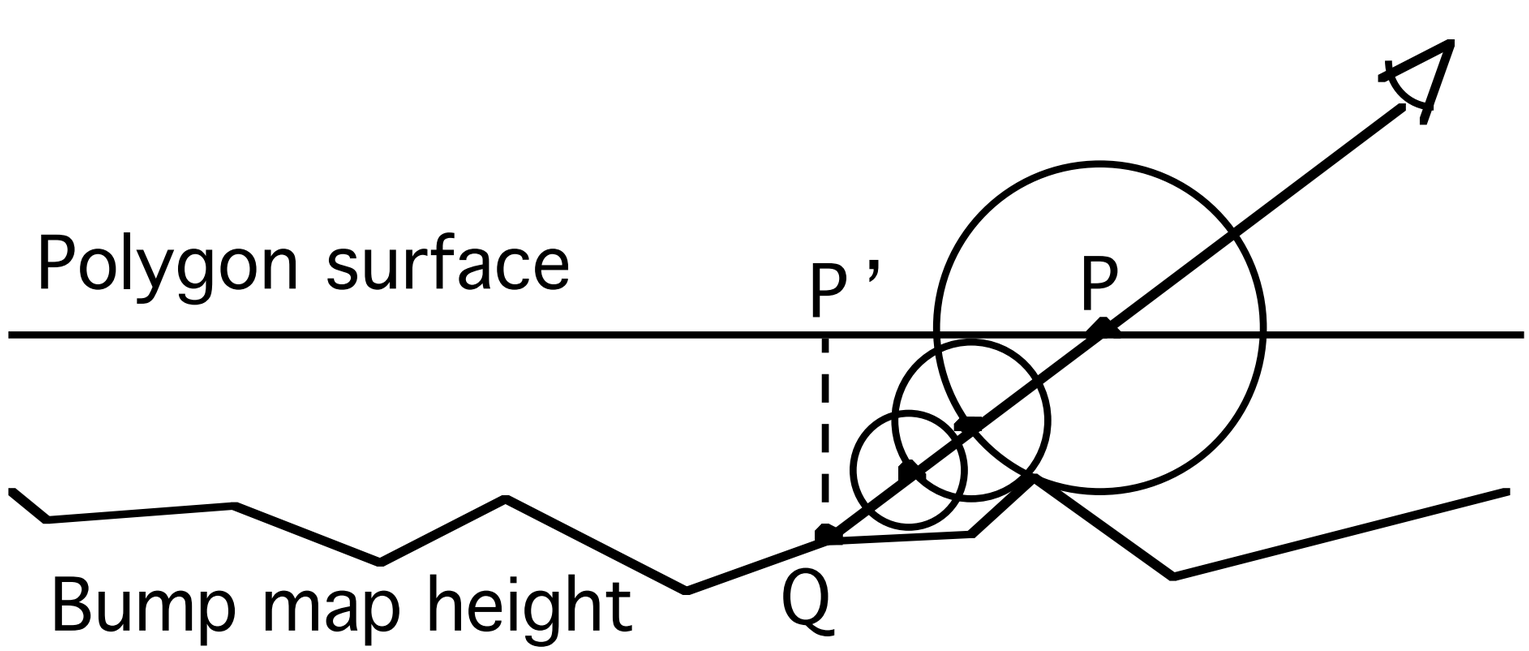
Variant: Parallax Occlusion Mapping

Liknande algoritm, uppvisar enligt Lindmark betydande fel.



Per-pixel Displacement Mapping

Mycket exakt metod som ger fina resultat



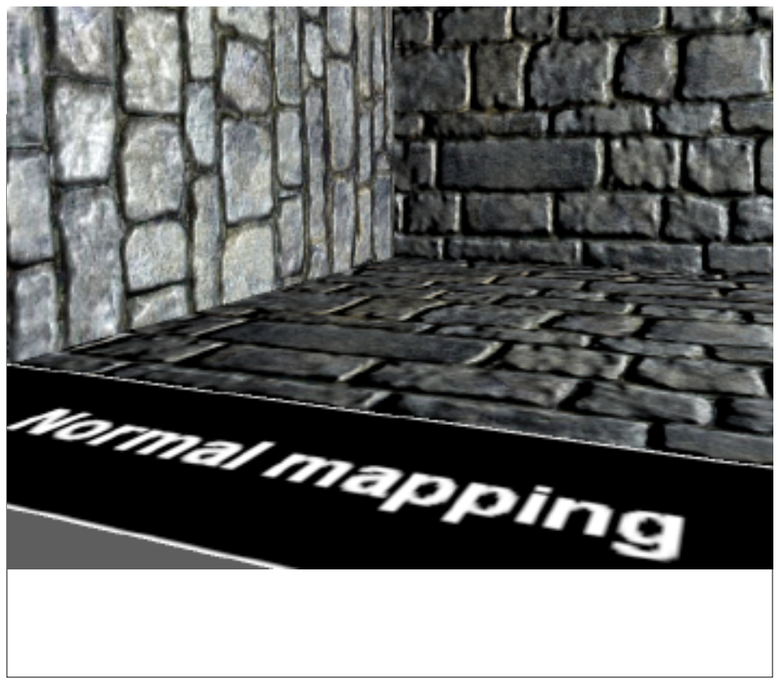
Hur utförs mätningen?

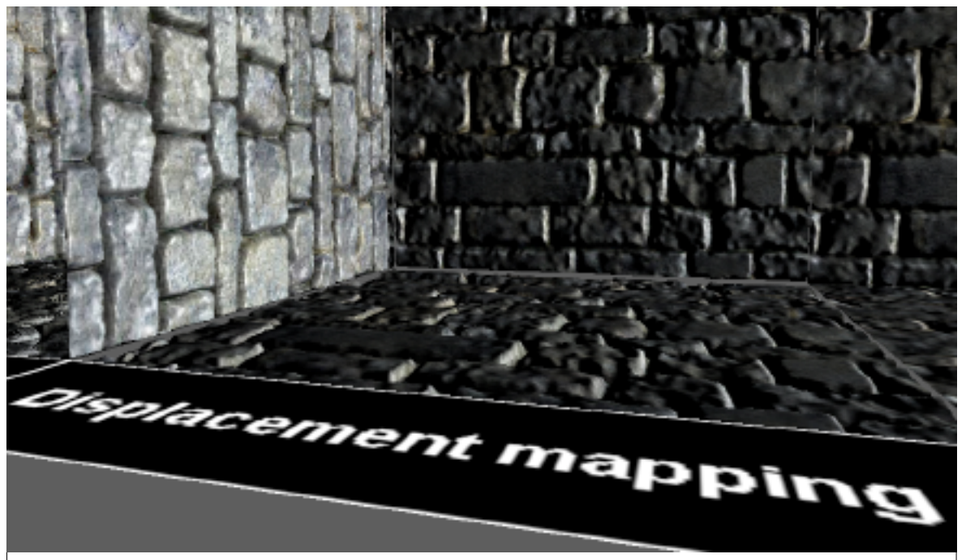
Använder Euklidisk avståndstransform

Algoritm för att effektivt beräkna avståndsvärden i samplad rymd

Uppfanns av PE Danielsson 1980

Per-pixel Displacement Mapping

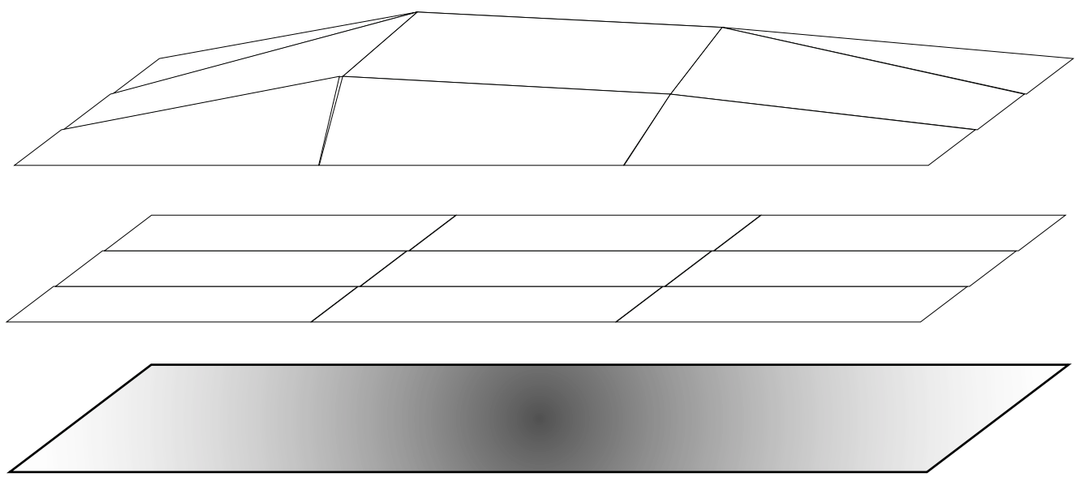




OBS! Förväxla inte per-pixel displacement mapping med per-vertex displacement mapping (ofta kombinerad med tesselering)

Per-vertex displacement mapping ändrar geometrin, inte bara shading och textur!

Intressant när vi använder tesselation shaders (kommer senare)



Vad behöver ett spel då?

- Platt texturmappning:Platt, lämnar allt till geometrin
- 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.

Sammanfattning bumpmappning

- Generera texturkoordinatsystem från texturkoordinater
- Beräkna bump mapping i vy- eller texturkoordinater
  - Normalmappning: förberäknad bumpmappning
- Utvidgningar och förbättringar av bumpmappning