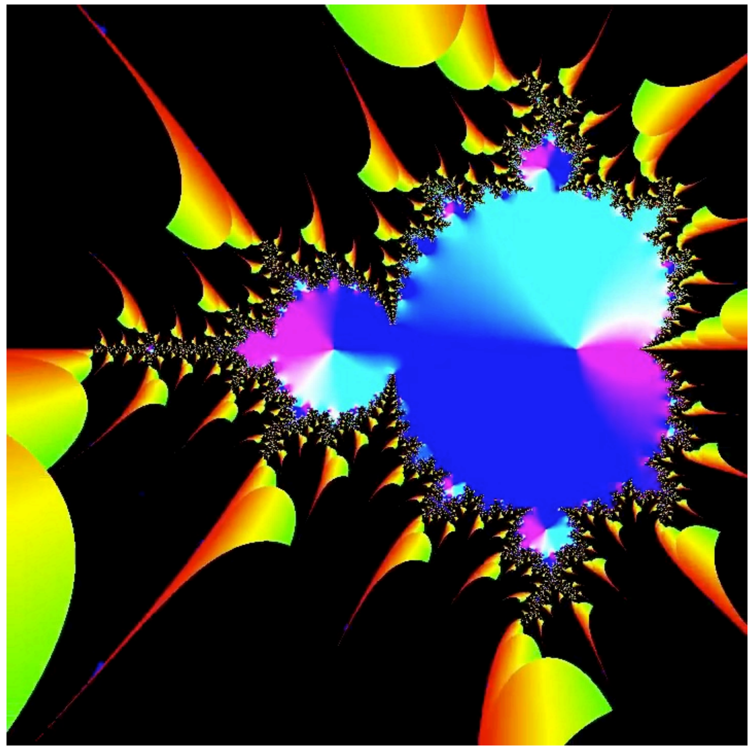
TSBK 07

Computer Graphics

Ingemar Ragnemalm, ISY

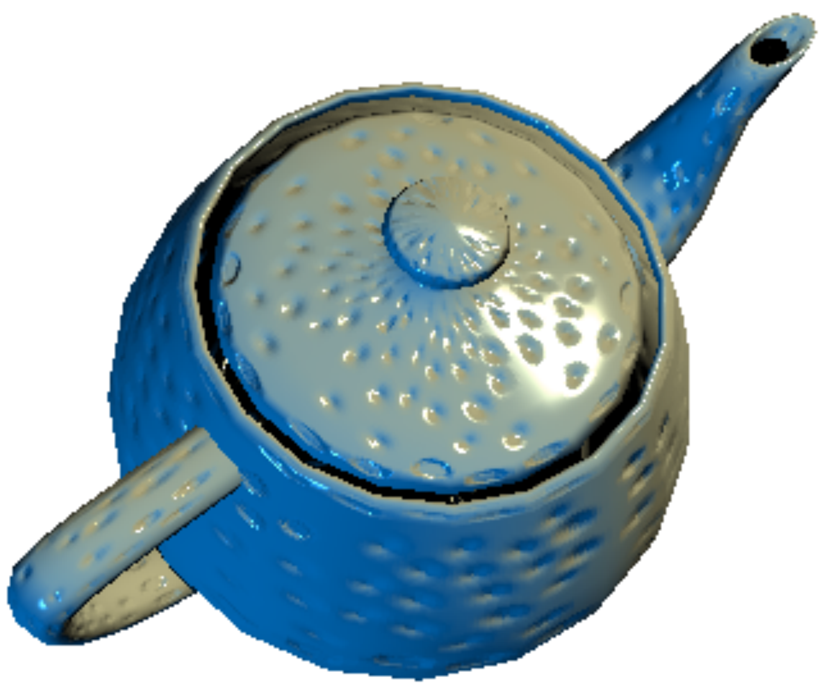


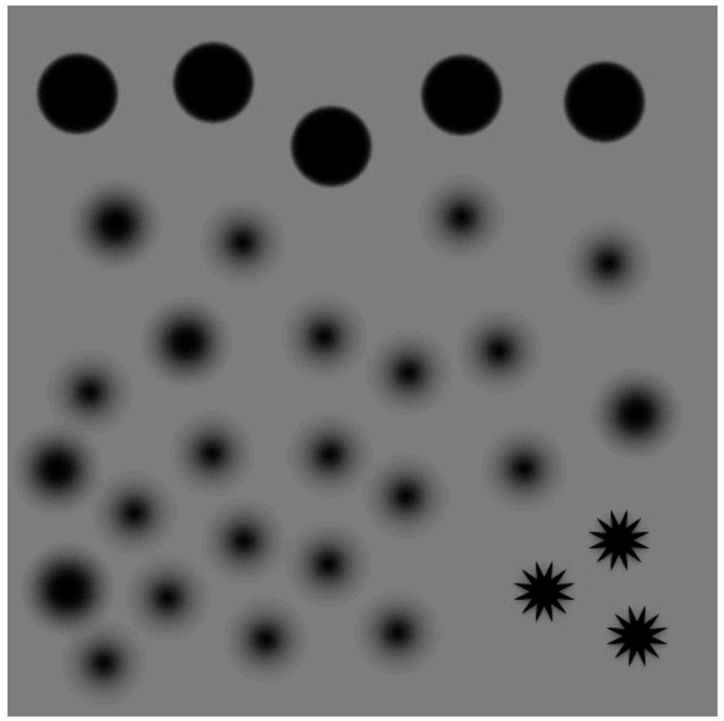
Lecture 7

More bump mapping
Light mapping
Normal matrix
Painter's algorithm
Transparency

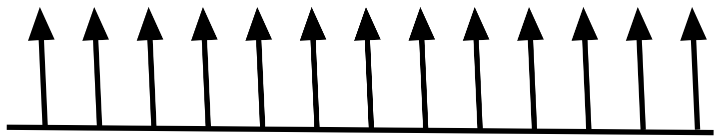
Bump mapping

Simulates surface structure by manipulating the normal vector

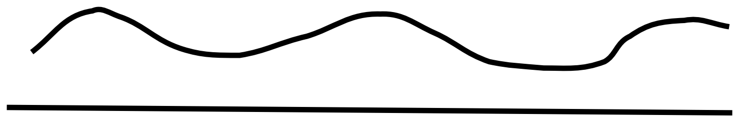




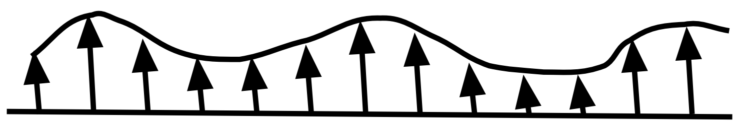
Bump mapping - model



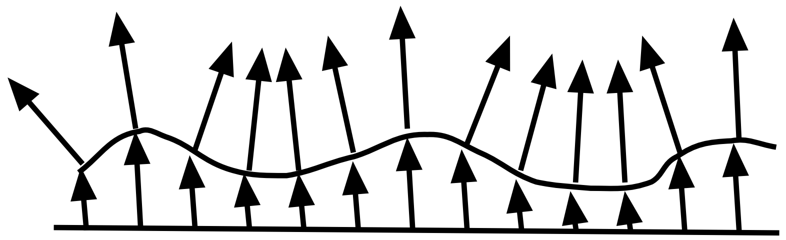
Surface with normal vectors



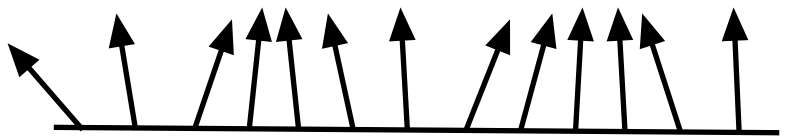
Bump map: scalar function of the texture coordinates



Modulate the surface by the bump function, along normal



Calculate new normals



Resulting normal vectors

Bump mapping - the coordinate systems

Input:
A point **p**, normal vector **n**
Texture coordinates s(**p**), t(**p**)
Directions of texture coordinates **s**, **t**
The bump function b(s,t)

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

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

or, if **s**, **t**, **n** are orthogonal

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

Texture coordinate system

How do we find the s and t vectors? We have the texture coordinates but no coordinate system!

Cross product with normal vector? With what?

Faking it

Cross product with absolutely anything!

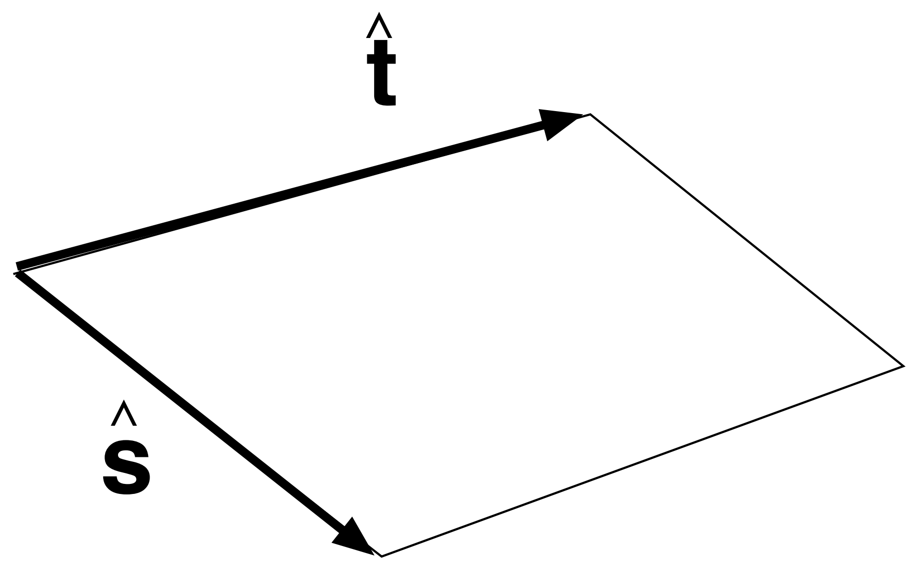
$$s = x \times n / |x \times n|$$
$$t = n \times s$$

Works for some cases. (Noise bump maps in particular.

But we can do better!

Trivial geometry

Very easy for a cube. Comfortable test case.



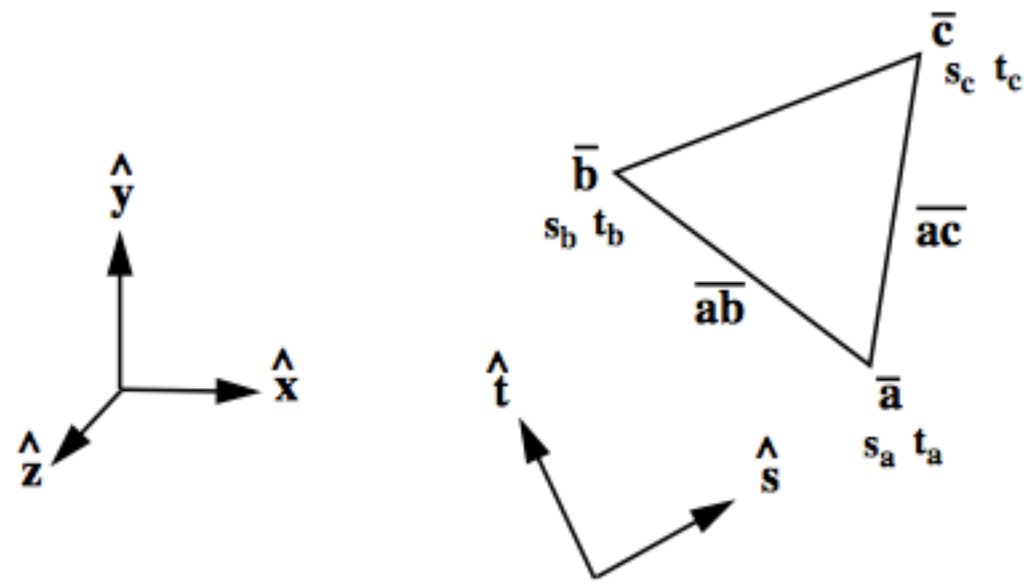
Lengyel's method

Derive through steps by s and t in xyz space

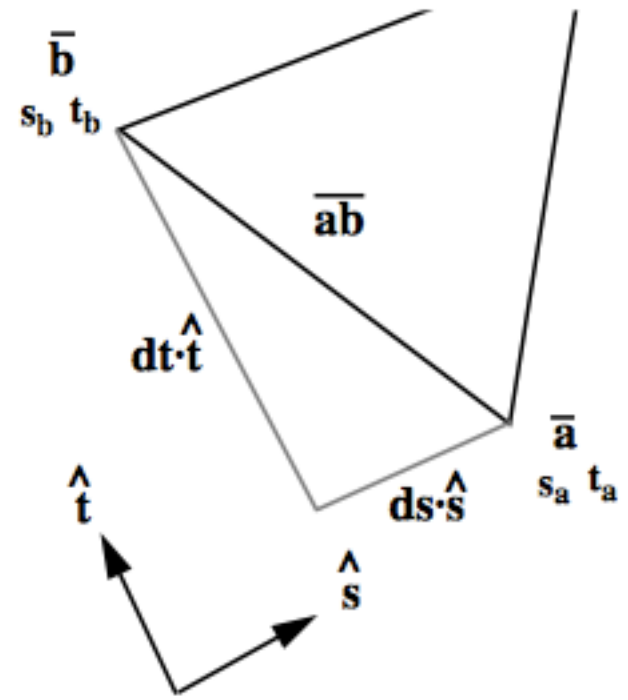
Straight and clean method using matrix algebra

Express two line segments as function of s and t, find the inverse!

Lengyel's method



Given a triangle with texture coordinates, find basis vectors for texture coordinates!



Take edge ab, split to components along s and t. Express as matrix. Find s and t by matrix inverse!

Lengyel's method

in program code - fairly simple!

```
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;
```

Note! Vector operations!

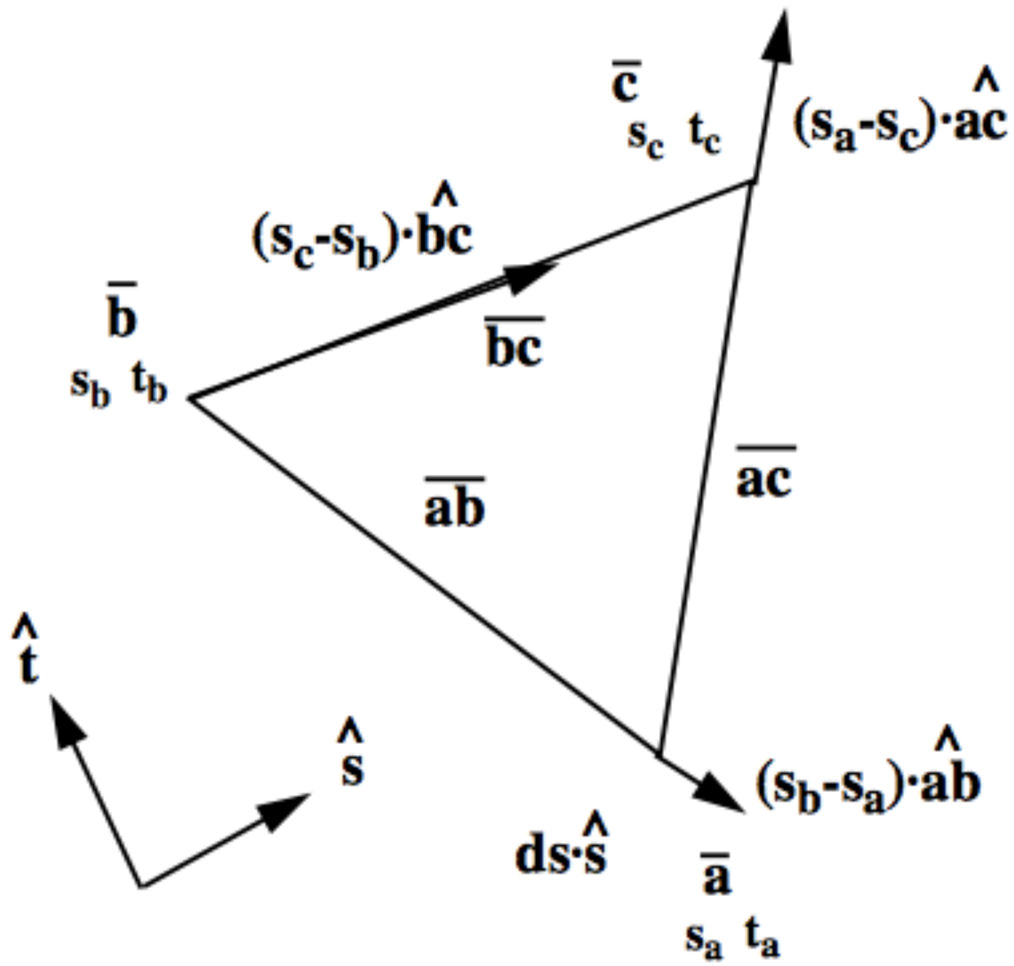
Approximative method

Let each edge of a polygon contribute to s and t depending on their variation in s and t!

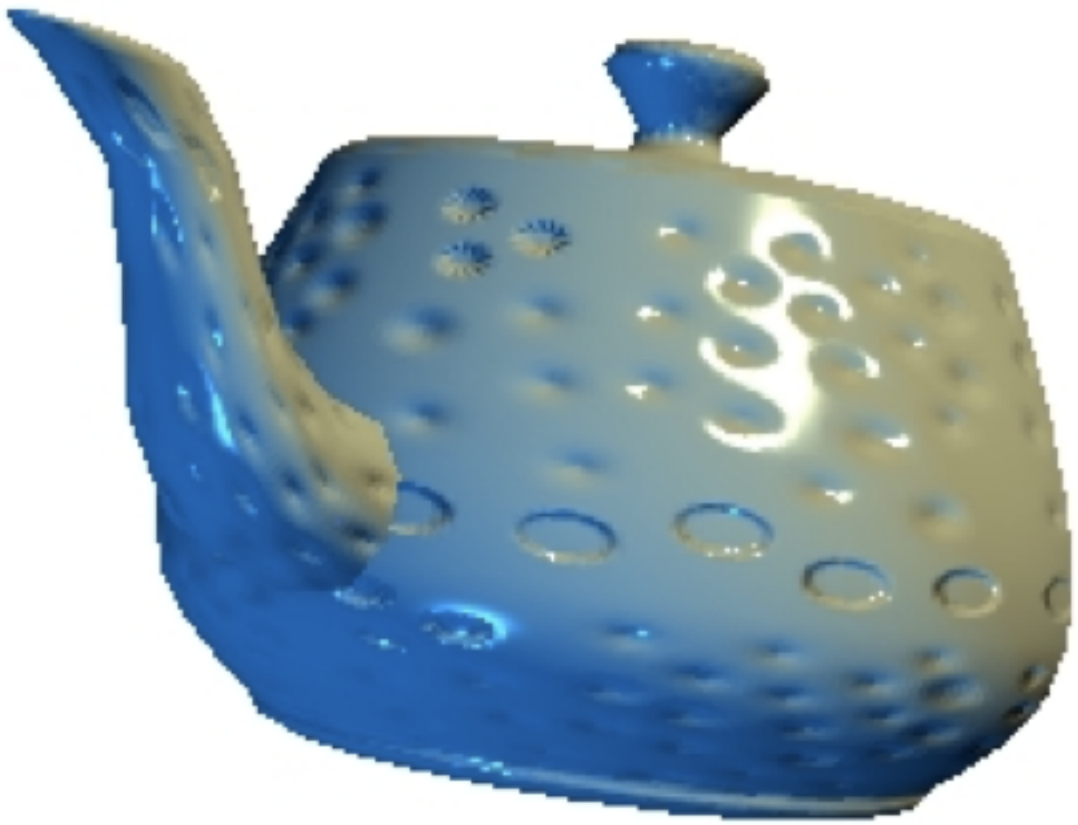
Contribution to s from each edge = the edge direction normalized times the variation in s.

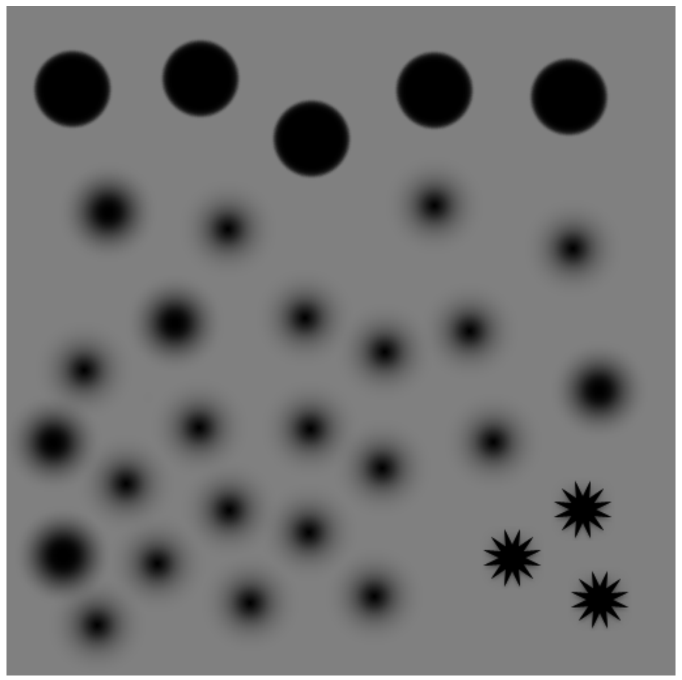
Approximative method

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



Both methods give good results for complicated models!





Coordinate systems

- View and world coordinates
- Texture coordinates
- Tangent coordinates

Light source often given in view coordinates

Bump map given in texture coordinates

Normal vector in model or view coordinates

We must convert between these coordinate systems! Light is calculated with vectors in the same coordinate system!

Coordinate systems

Model to view: normalMatrix

$$s_v = \text{normalMatrix} * s$$
$$t_v = \text{normalMatrix} * t$$
$$n_v = \text{normalMatrix} * n$$

View to texture:

$$M_{vt} = \begin{bmatrix} s_v \\ t_v \\ n_v \end{bmatrix} = \begin{bmatrix} s_{vx} & s_{vy} & s_{vz} \\ t_{vx} & t_{vy} & t_{vz} \\ n_{vx} & n_{vy} & n_{vz} \end{bmatrix}$$

Coordinate system

s_V is the *tangent vector* (often called t i other texts)
t_V *bitangent (<u>not</u> binormal*)

Texture space = basis with vectors along texture variations

Tangent space = orthonormal basis in texture space

Tangent space often good approximation to texture space

More definitions

bump map = picture with height values

normal map = picture with pre-calculated normal vectors

(These are sometimes confused)

Calc of modified normal vector (view coordinates)

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

$\mathbf{n'} = \mathbf{n_v} + b_s \cdot \mathbf{s_v} + b_t \cdot \mathbf{t_v}$ **("in")**

or

$\mathbf{n'} = \mathbf{n_v} - b_s \cdot \mathbf{s_v} - b_t \cdot \mathbf{t_v}$ **("ut")**

Calc of modified normal vector (texture coordinates)

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

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

Really easy! BUT, the light and view directions must be transformed to texture coordinates!

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

Normal mapping

Precalculate b_s och b_t, save as picture!

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

Normalize!

Storage in texture

”Scale and bias”:

R = (ds+1)/2
G = (dt+1)/2

(Why?)

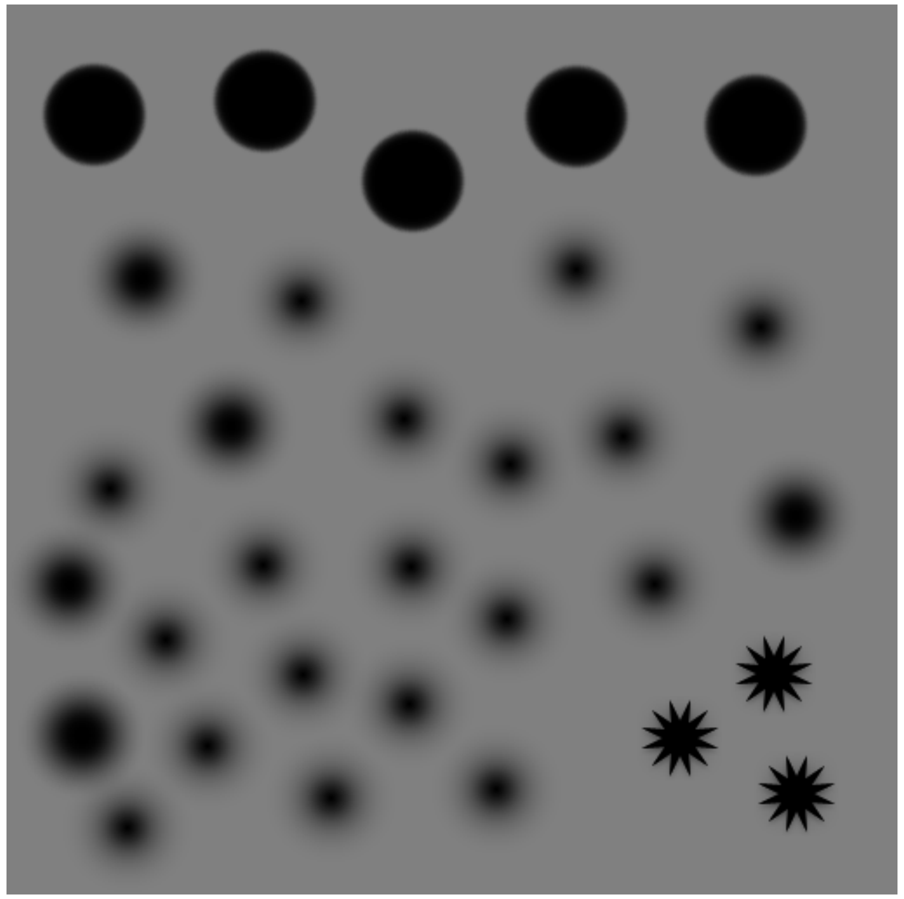
Fetch from texture:

ds = 2R - 1
dt = 2G - 1

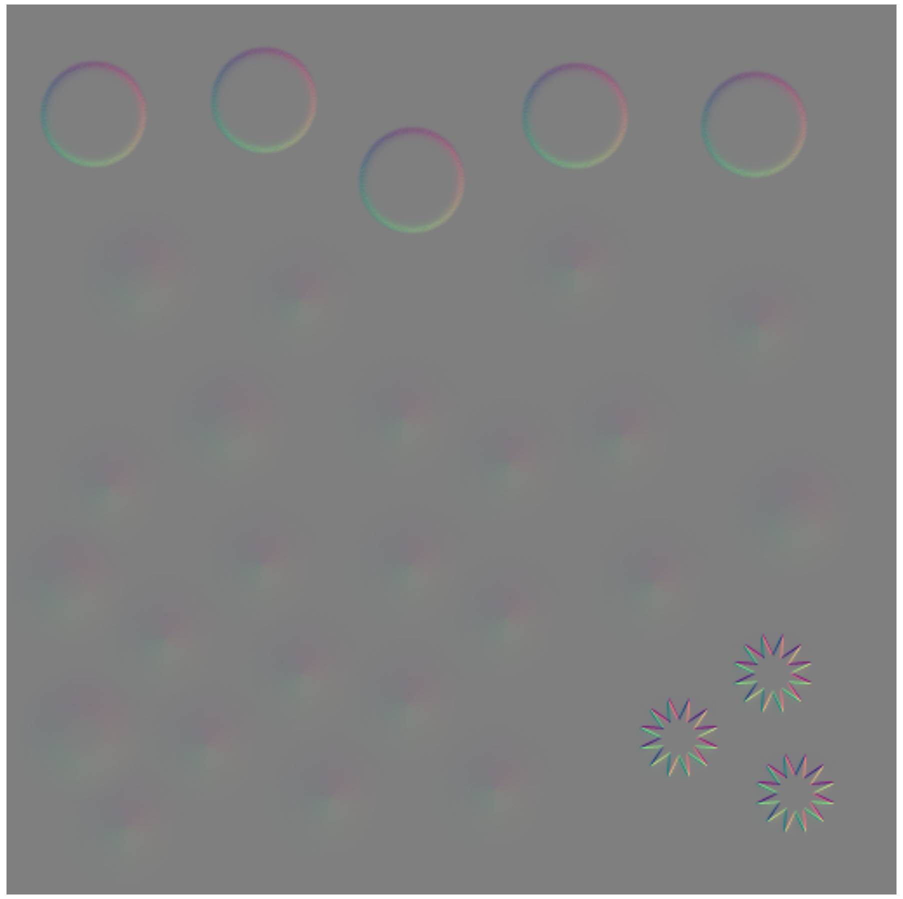
n_t = (x, y, z)

Example of normal map

Bump map in my example



Bump map



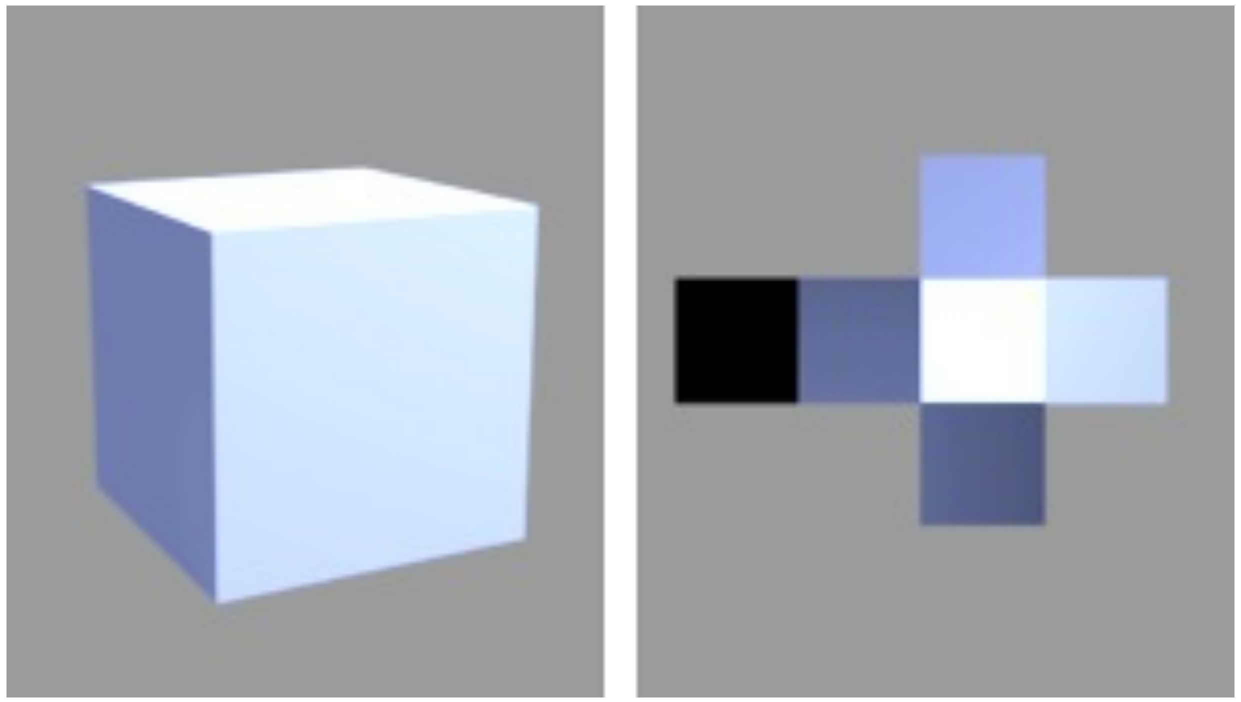
Normal map

Light mapping

Applying pre-calculated lighting to a model

Saves real-time processing time for models with static lighting

Allows high-quality lighting with high performance



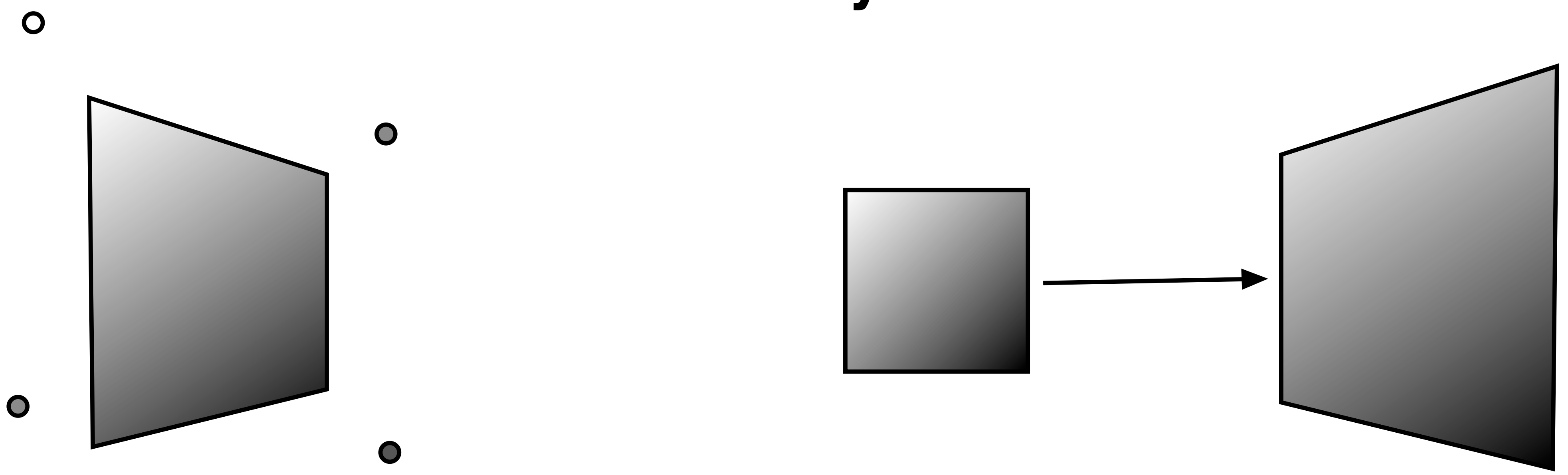
(Image from Wikipedia)

Light mapping

Two approaches:

- **Vertex-level light mapping**
- **Light map textures**

Both methods are high-performance and require little memory!



Vertex-level light mapping

Calculate lighting per vertex
Apply with glColor
Render with Gouraud shading

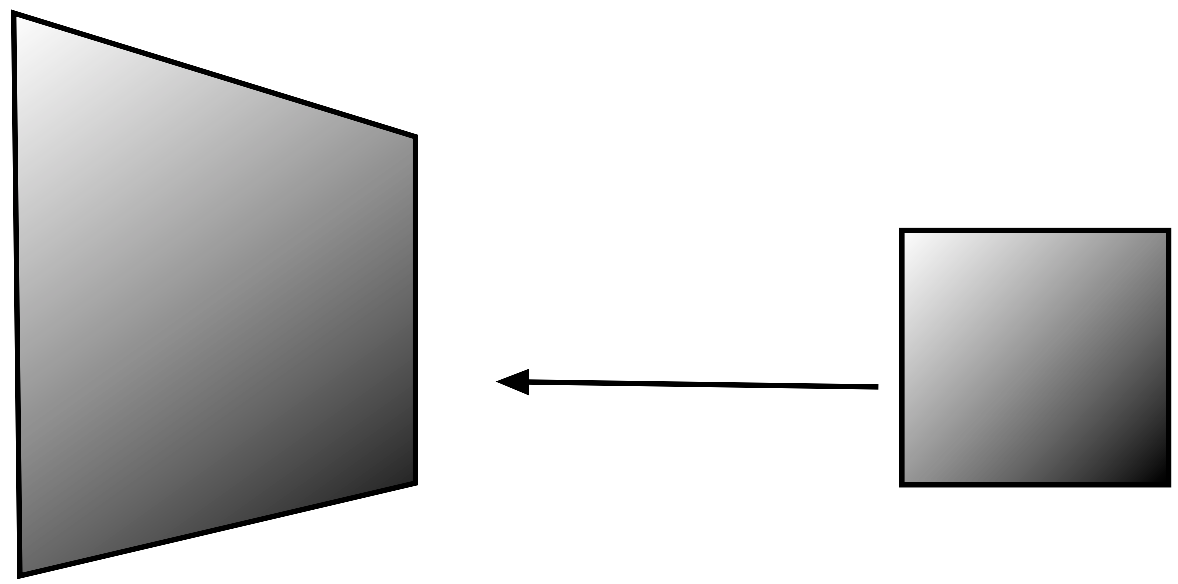
Trivial to use with textures. Very fast and low memory demand. Limited quality,

Light map textures

Pre-calculate image any way you like (e.g. radiosity)

Allows arbitrary precision with low processing demand

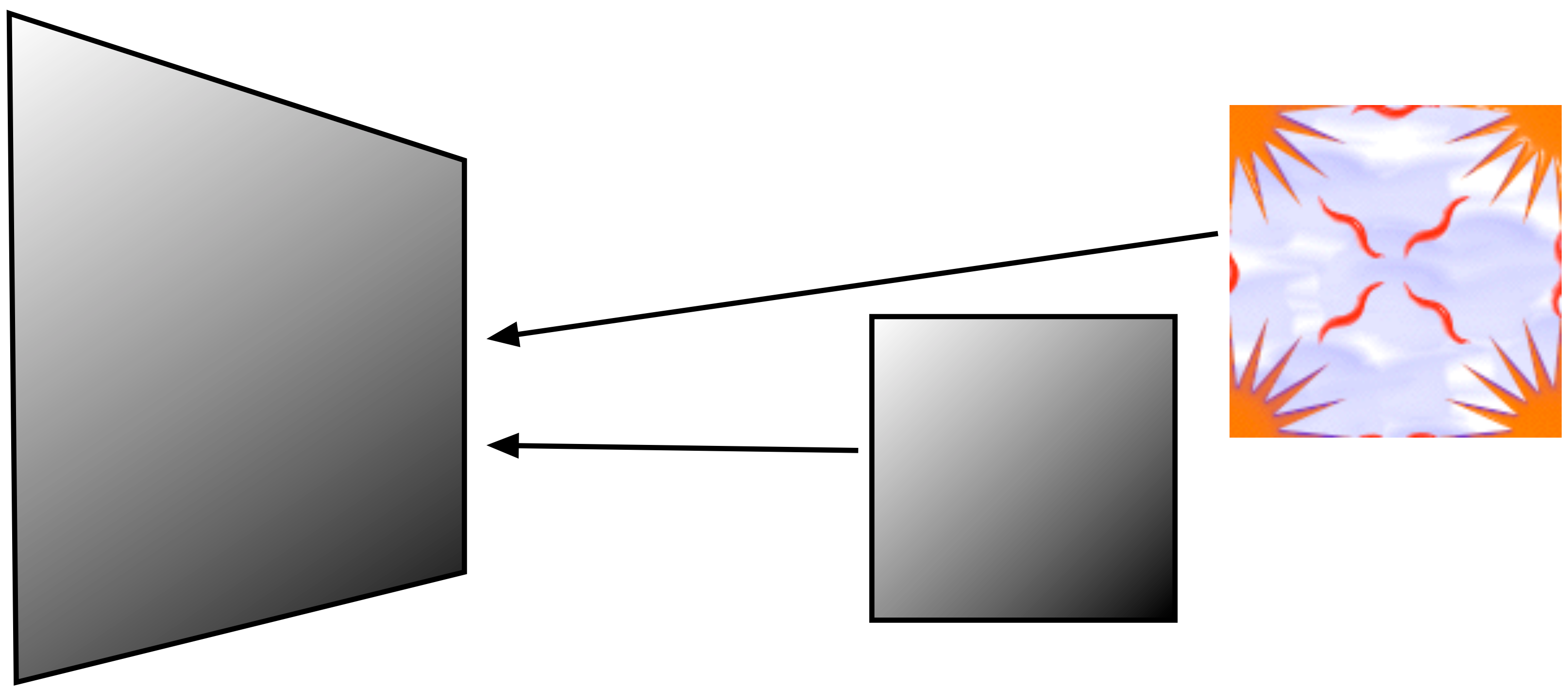
Images usually very small, but can be made large when needed



Light map textures

Texture maps and light maps can be applied to the same surface by multitexturing.

Texture values (texels) are multiplied by light map values.



Light map textures

Generation:

1. Hand-painted
2. Ray-tracing
3. Radiosity
4. Provided by 3D tools

Generate a palette of light maps, reuse within some tolerance