Lecture 7:

Texturing 1

Contents

Purely heuristic model

- Initially without units (values $\in [0; 1]$)

$$L_r = L_{r,a} + L_{r,d} + L_{r,s}\left(+L_{r,m} + L_{r,t}\right)$$

$L_{r,a}$: Ambient term

- Approximate indirect illumination

$L_{r,d}$: Diffuse term (Lambert)

- Uniform reflection

$L_{r,s}$: Specular term

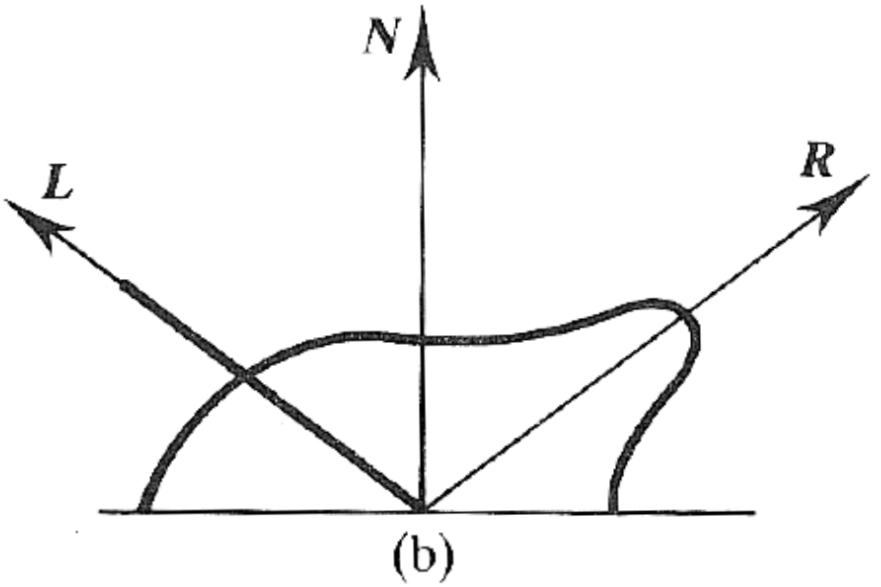
- Mirror-reflection on a rough surface

$+L_{r,m}$: Perfect reflection

- Only possible with Ray-Tracing

$+L_{r,t}$: Perfect transmission

- Only possible with Ray-Tracing



Extended light sources: l point light sources

$$L_r = k_a L_{i,a} + k_d \sum_l L_i (I_l \cdot N) + k_s \sum_l L_i (R(I_l) \cdot V)^{k_e} \quad \text{(Phong)}$$

$$L_r = k_a L_{i,a} + k_d \sum_l L_i (I_l \cdot N) + k_s \sum_l L_i (H_l \cdot N)^{k_e} \quad \text{(Blinn)}$$

Color of specular reflection equal to light source

Heuristic model

- Contradicts physics
- Purely local illumination
  - Only direct light from the light sources
  - No further reflection on other surfaces
  - Constant ambient term

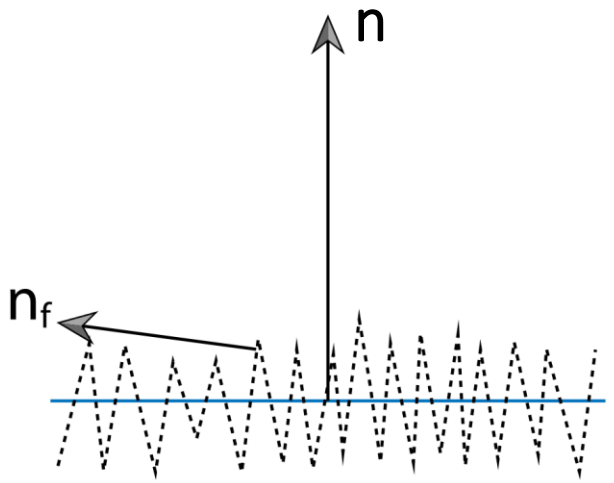
Often: light sources & viewer assumed to be far away

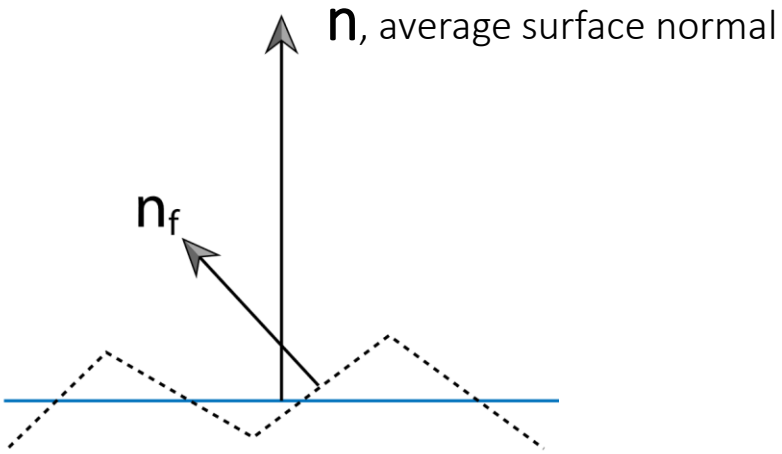
Idea:

- Rough surfaces can be modelled as a collection of small *microfacets*
- Where the distribution of faces is described statistically

Microfacet surface models are often described by a function that gives the distribution of microfacet normals with respect to the surface normal

- The greater the variation of microfacet normals, the rougher the surface is
- Smooth surfaces have relatively little variation of microfacet normals



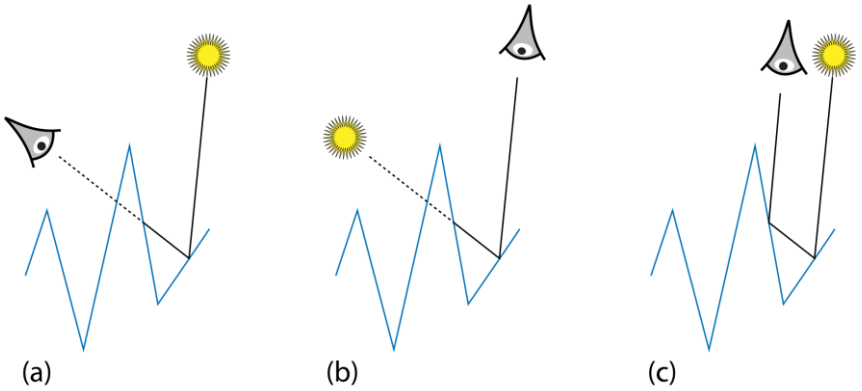


Important geometric effects to consider with microfacet reflection models:

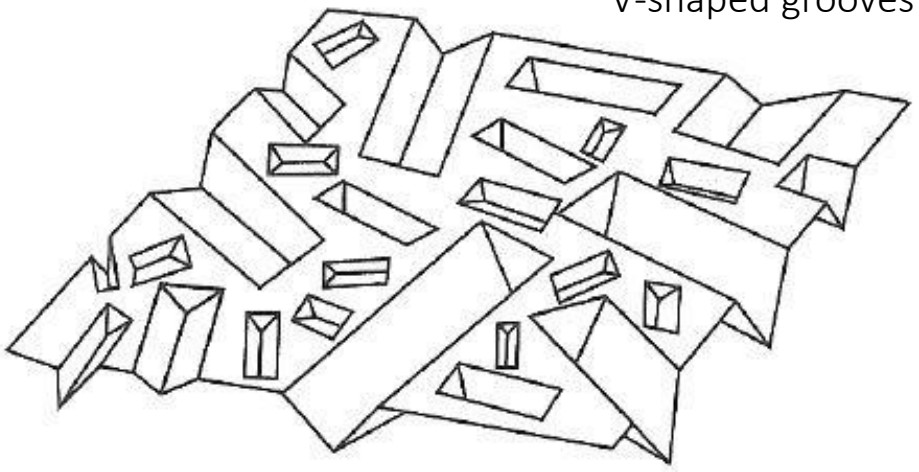
- **Masking**: the microfacet of interest isn't visible to the viewer due to occlusion by another microfacet
- **Shadowing**: analogously, light doesn't reach the microfacet
- **Interreflection**: light bounces among the microfacets before reaching the viewer

Microfacets assumed as perfectly smooth reflectors BRDF

- Torrence-Sparrow model (1967): Perfect Mirror model
- Oren-Nayar model (1994): Lambertian reflectors
- Ward anisotropic distribution (1992):

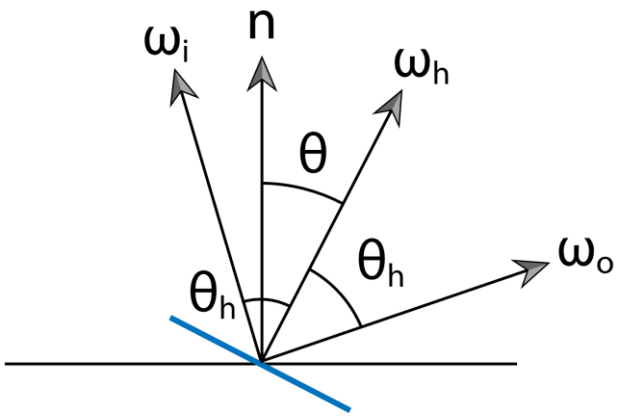


V-shaped grooves



Setting for the Derivation of the Torrance–Sparrow Model:

- For directions ω_i and ω_o, only microfacets with normal ω_h reflect light.



The Torrance–Sparrow Model:

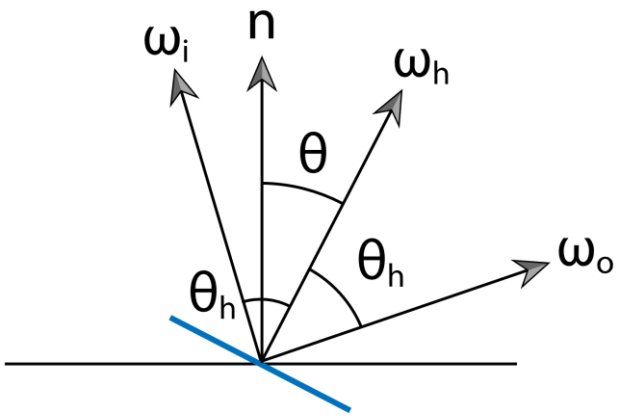
$$f_r = \frac{D(\omega_h) \cdot G(\omega_o, \omega_i) \cdot F_r(\omega_0)}{4 \cos \theta_o \cdot \cos \theta_i}$$

- D: *statistical microfacet distribution function*, that gives the probability that a microfacet has orientation ω_h
- G: *geometric attenuation* term, which describes the fraction of microfacets that are masked or shadowed
- F_r: *Fresnel term,* computed by Fresnel equation and relates incident light to reflected light for each planar microfacet

Isotropic microfacet distributions:

$$D(\omega_h) \Longrightarrow D(\theta)$$

$$\cos\theta = n \cdot \omega_h$$



Blinn

$$D(\theta) = \frac{e+2}{2\pi}(\cos\theta)^e$$

Torrance-Sparrow

$$D(\theta) = e^{-\left(\frac{\sqrt{2}\alpha}{\theta_h}\right)^2}$$

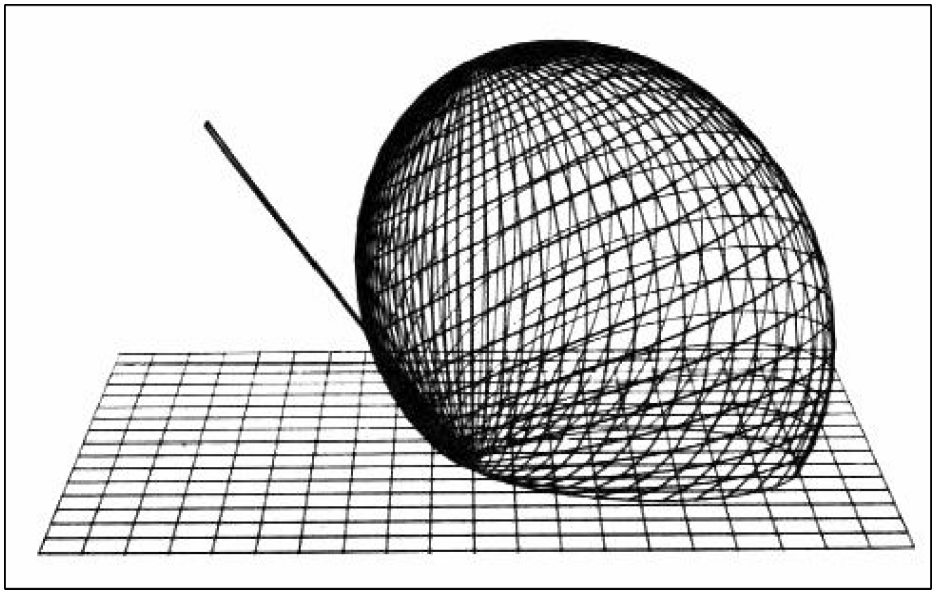
Beckmann

$$D(\theta) = \frac{1}{4\sigma^2 cos^4\theta} e^{-\left(\frac{tan\theta}{\sigma}\right)^2}$$

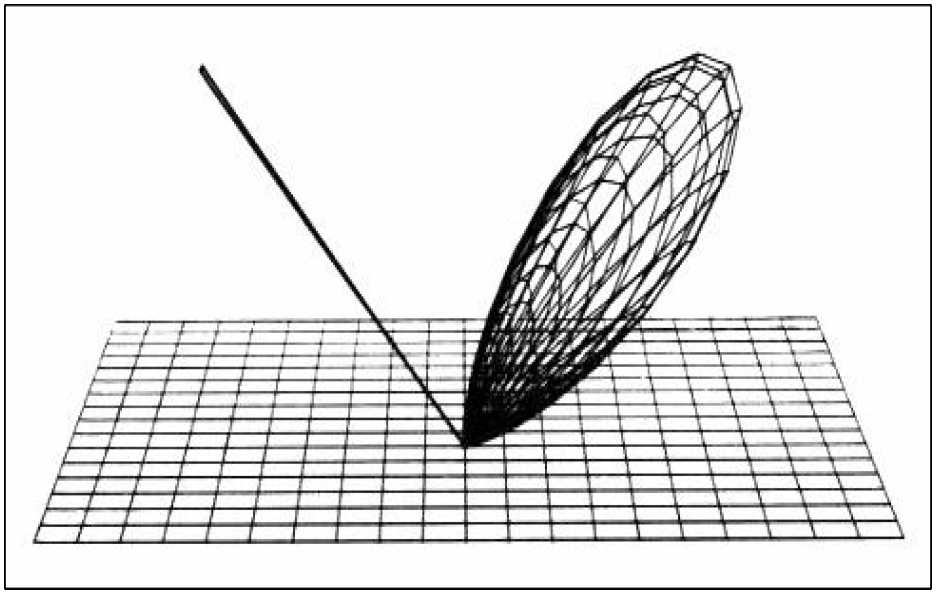
- σ : root mean square

The effect of varying the exponent for the Blinn microfacet distribution model:

- The larger the exponent, the more likely it is that a microfacet will be oriented close to the surface normal, as would be the case for a smooth surface.



Distribution from exponent *e=4*

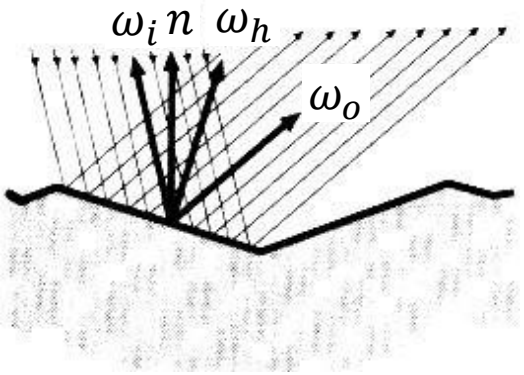


Distribution from exponent *e=20*

V-shaped grooves

- Fully illuminated and visible

$$G(\omega_o, \omega_i) = 1$$

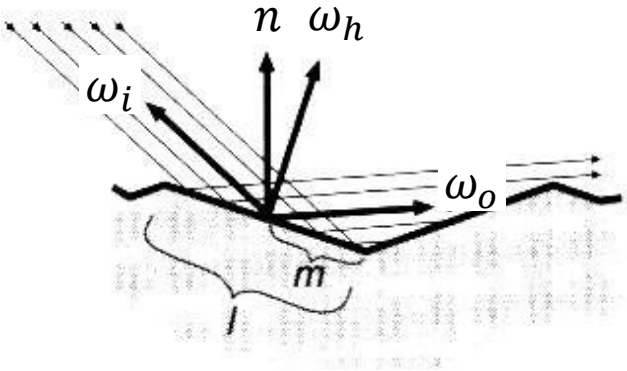


- Partial masking of reflected light

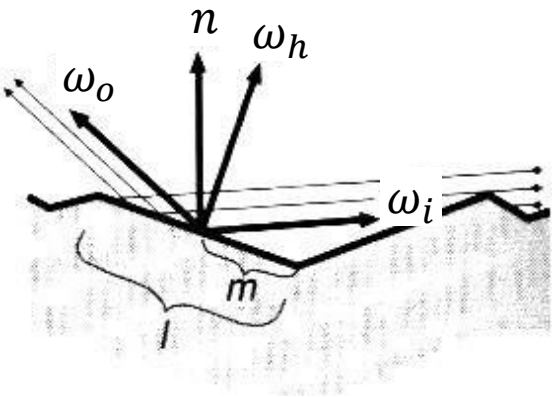
$$G(\omega_o, \omega_i) = \frac{2(n \cdot \omega_h)(n \cdot \omega_o)}{(\omega_o \cdot \omega_h)}$$

- Partial shadowing of incident light

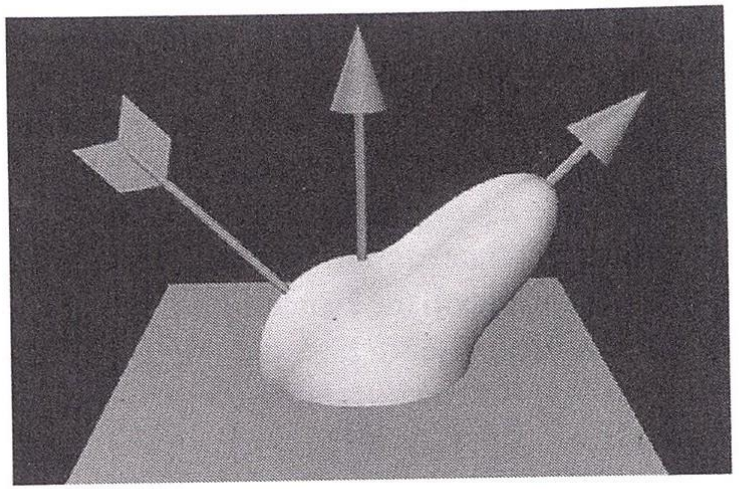
$$G(\omega_o, \omega_i) = \frac{2(n \cdot \omega_h)(n \cdot \omega_i)}{(\omega_o \cdot \omega_h)}$$



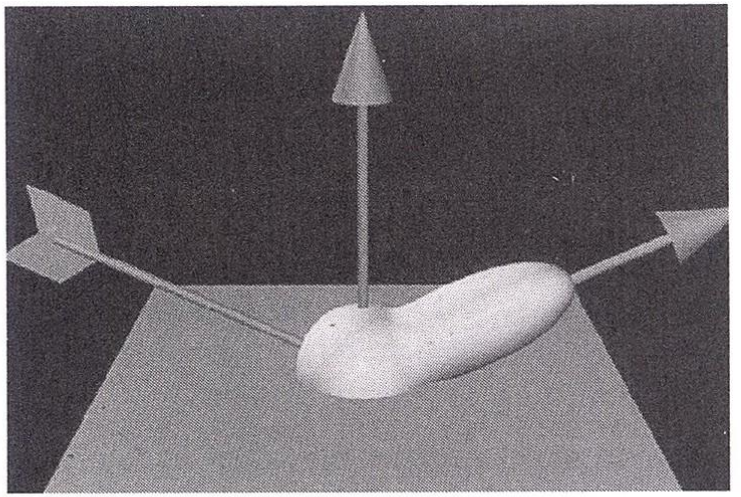
$$G(\omega_o, \omega_i) = \min\left\{1, \frac{2(n \cdot \omega_h)(n \cdot \omega_o)}{(\omega_o \cdot \omega_h)}, \frac{2(n \cdot \omega_h)(n \cdot \omega_i)}{(\omega_o \cdot \omega_h)}\right\}$$



Phong:

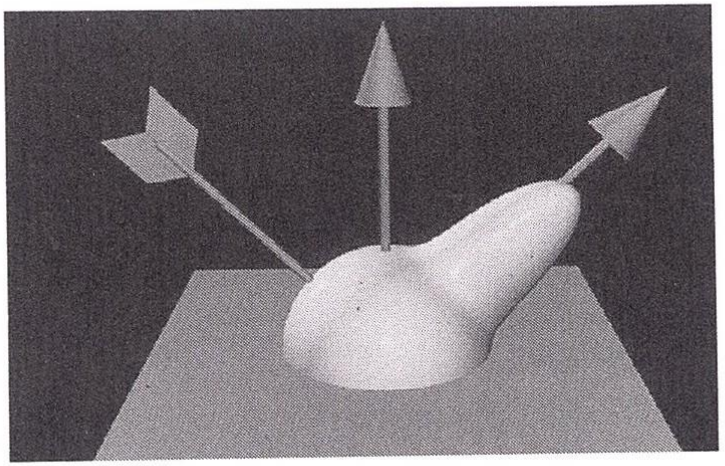


(a)

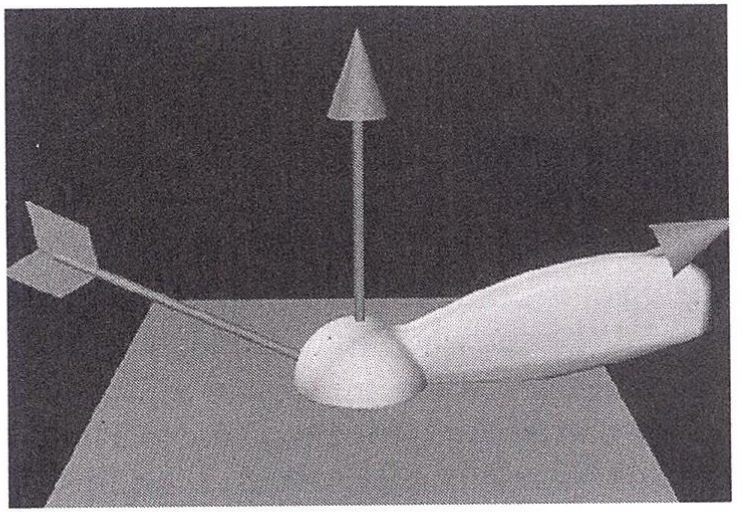


(b)

Torrance–Sparrow:



(c)



(d)

Application of an illumination model to compute intensity for every pixel has been time consuming.

Intensity of adjacent pixels is usually very similar (the so called shading coherence), which allows for less frequent shading evaluations.

Each polygon can be rendered with a single intensity or intensity can be obtained at each point of the surface using an interpolation scheme:

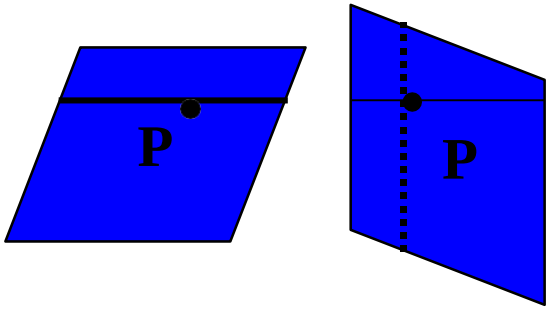
- **Flat shading**, single intensity is calculated for each polygon
- **Gouraud shading** (per vertex shading), intensity calculated at vertices is interpolated across the surface
- **Phong shading** (per pixel shading), normal vectors are calculated at vertices; then normal vectors are interpolated across the surface and an illumination model using these normal vectors is applied for every point of the surface

With modern hardware this is no big issue any more

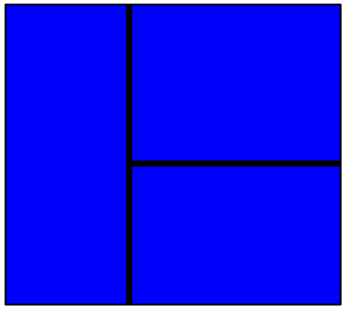
- Often even the normal is calculated per pixel
  - Bump or displacement maps

Problems

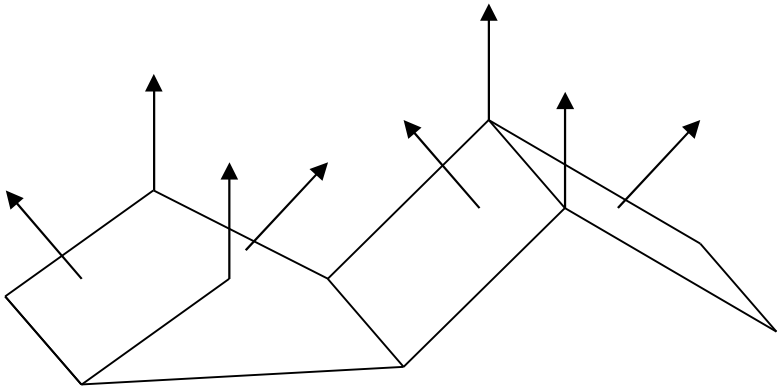
- Polygonal silhouette may not match the smooth shading
- Perspective distortion
  - Interpolation may be performed after perspective transformation in the 2-D screen coordinate system, rather than world coordinate system.
- Orientation dependence
  - This problem does not concern triangles for which linear interpolation is rotation-invariant.
- Shading discontinuities at shared vertices (T-edges).
- Unrepresentative normal vectors.



Shading at **P** is interpolated along different scan-lines when polygon rotates.



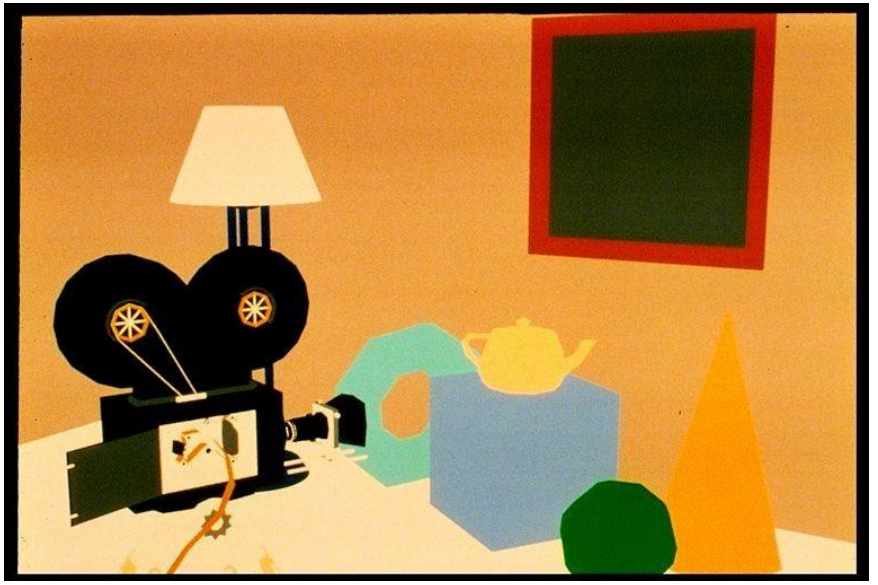
T-edges



Vertex normals are all parallel

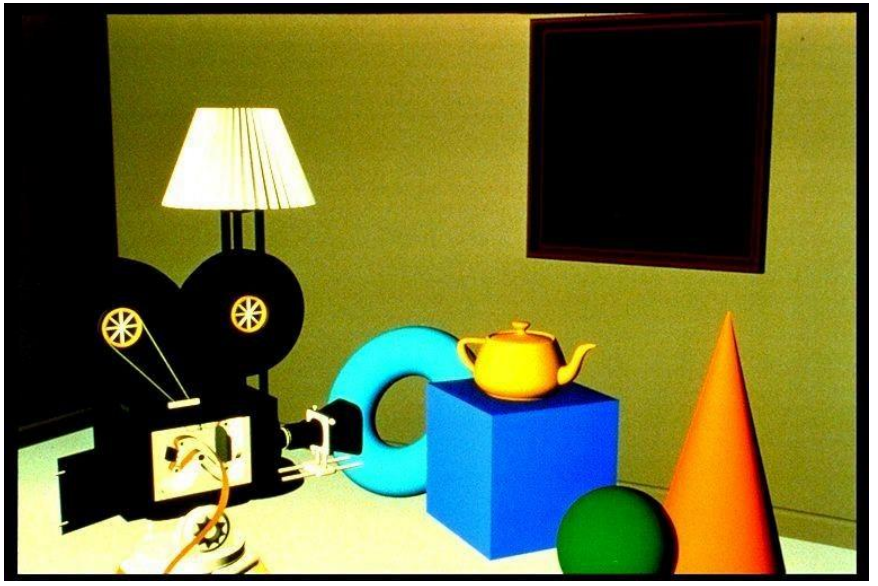
No illumination

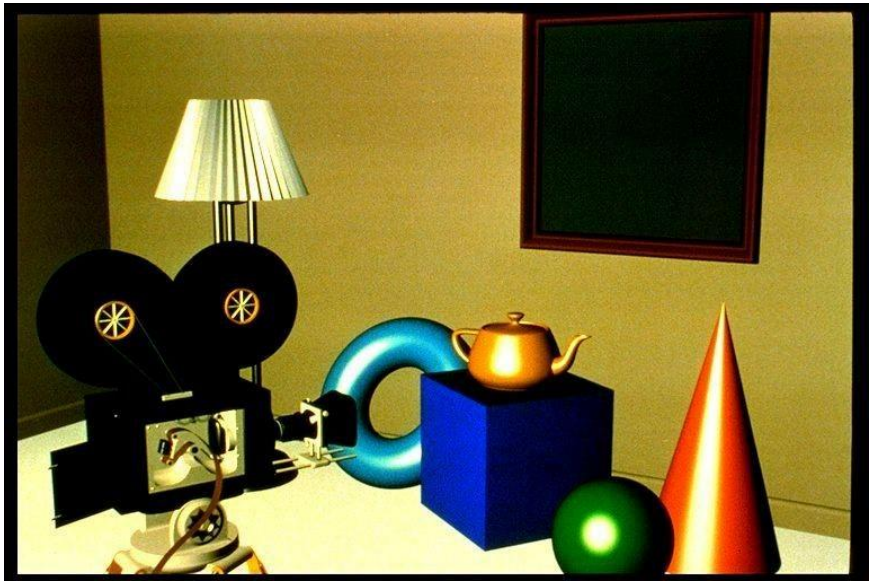
Constant colors



Parallel light

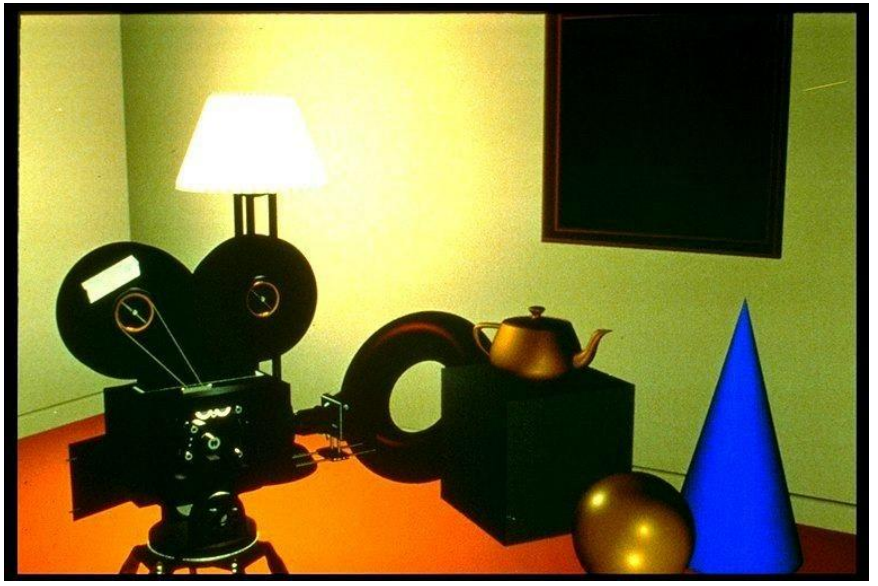
Diffuse reflection





Parallel light

Specular reflection



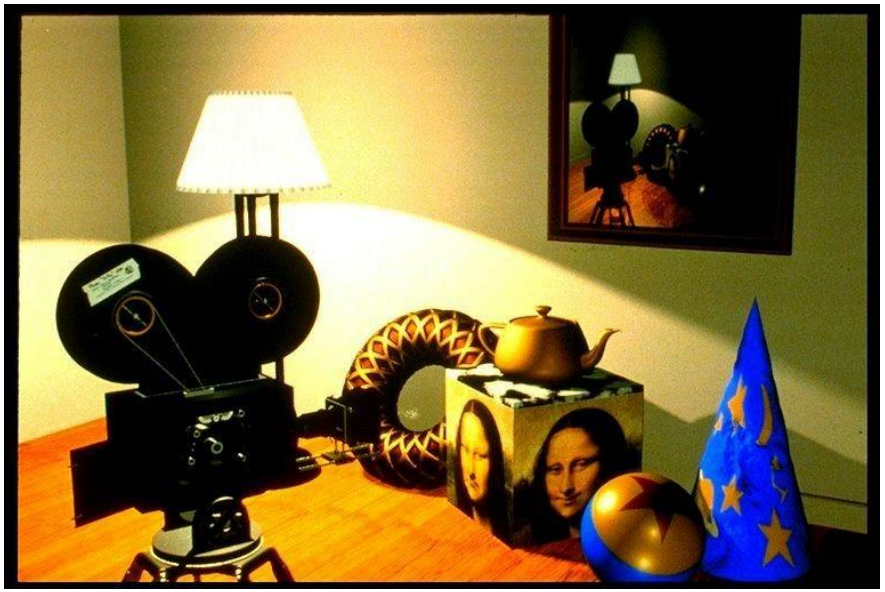
Multiple local light sources

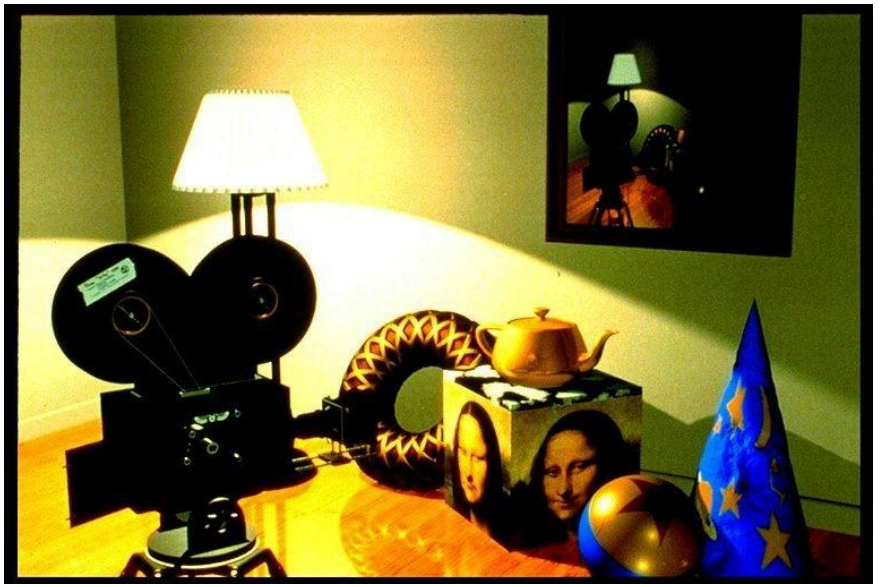
Different BRDFs

Object properties constant over surface

Locally varying object characteristics

- 2D Image Textures
- Shadows
- Bump-Mapping
- Reflection textures





Modulation of object surface properties

Reflectance

- Color (RGB), diffuse reflection coefficient k_d
- Specular reflection coefficient k_s

Opacity (α)

Normal vector

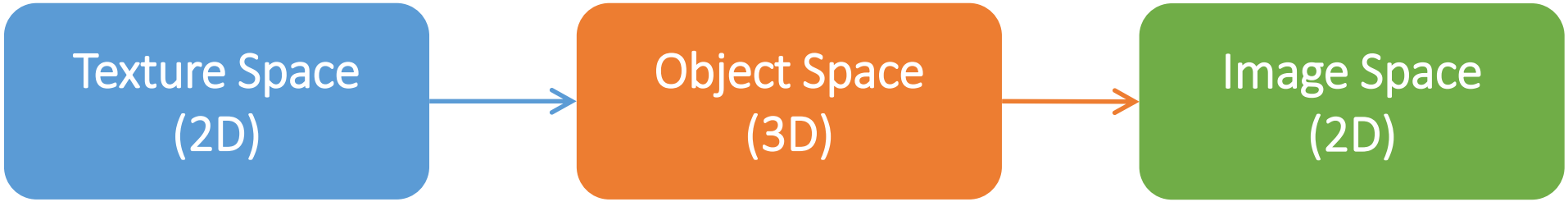
- $N(P) = N(P + tN)$ or $N = N + dN$
- „Bump mapping“ or „Normal mapping“

Geometry

- $P = P + dP$
- „Displacement mapping“

Distant illumination

- “Environment mapping“, “Reflection mapping“



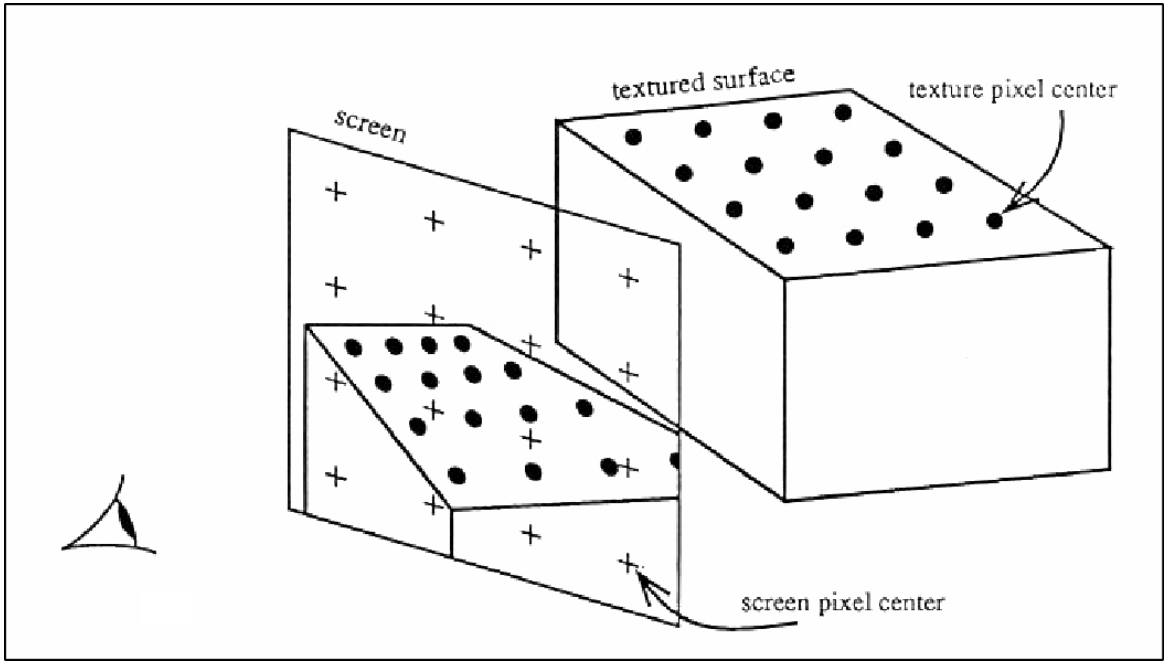
Texture-Surface Transformation

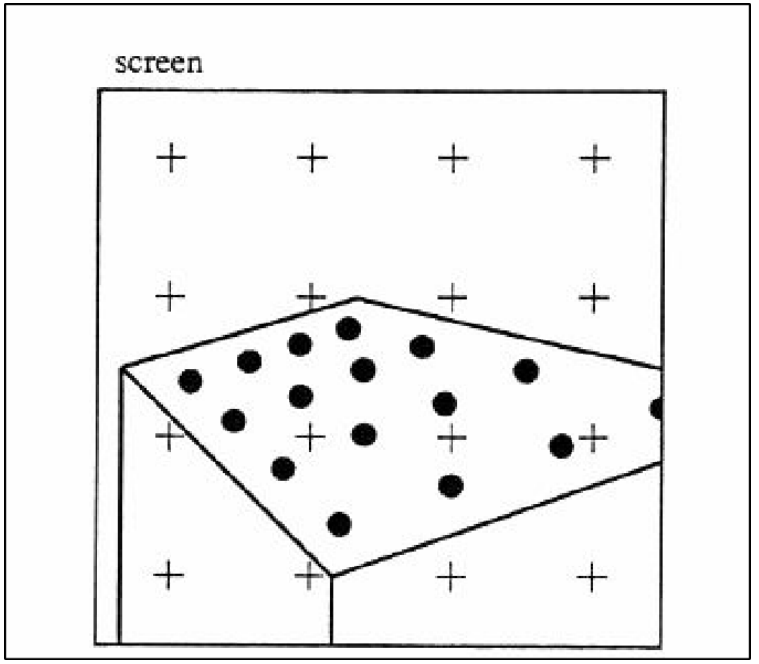
Viewing / Projection Transformation

The texture is mapped onto a surface in 3-D object space, which is then mapped to the screen by the viewing projection. These two mappings are composed to find the overall 2D texture space to 2D image space mapping, and the intermediate 3D space is often forgotten. This simplification suggests texture mapping's close ties with image warping and geometric distortion.

- Texture space (u, v)
- Object space (x_0, y_0, z_0)
- Screen space (x, y)

2D Texturing





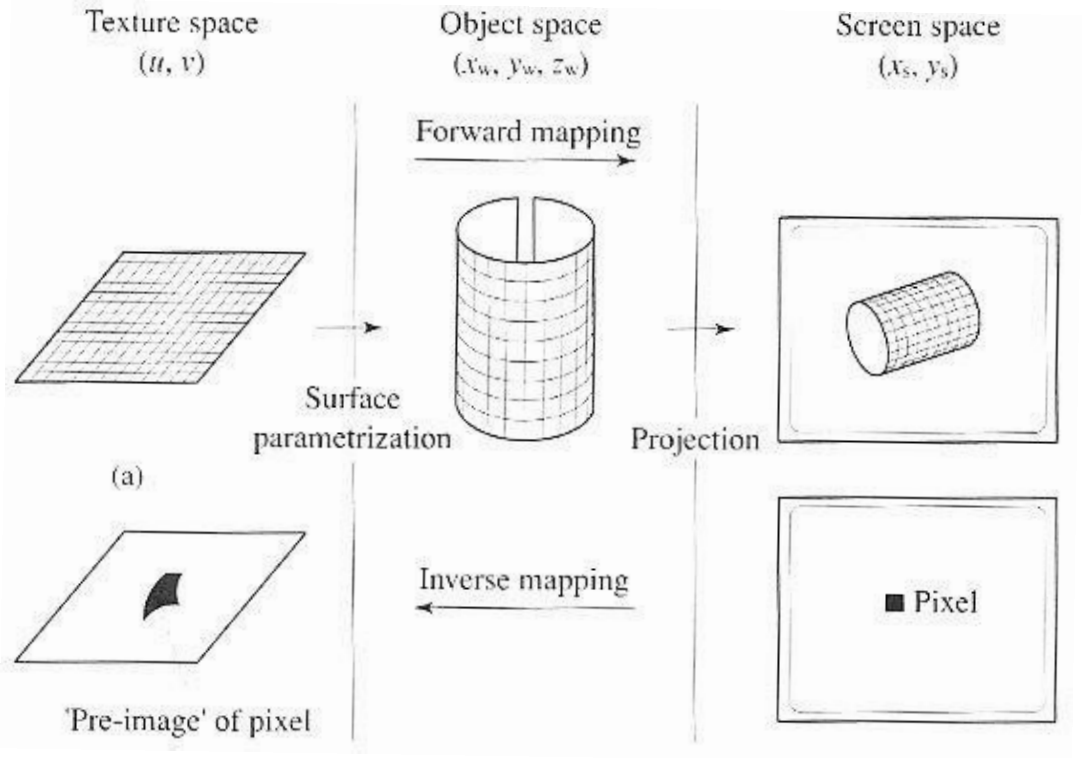
- 2D texture mapped onto object
- Object projected onto 2D screen
- 2D to 2D: warping operation
- Uniform sampling?
- Hole – filling / blending?

Forward mapping

- Object surface parameterization
- Projective transformation

Inverse mapping

- Find corresponding pre-image/footprint of each pixel in texture
- Integrate over pre-image



Maps each texel to its position in the image

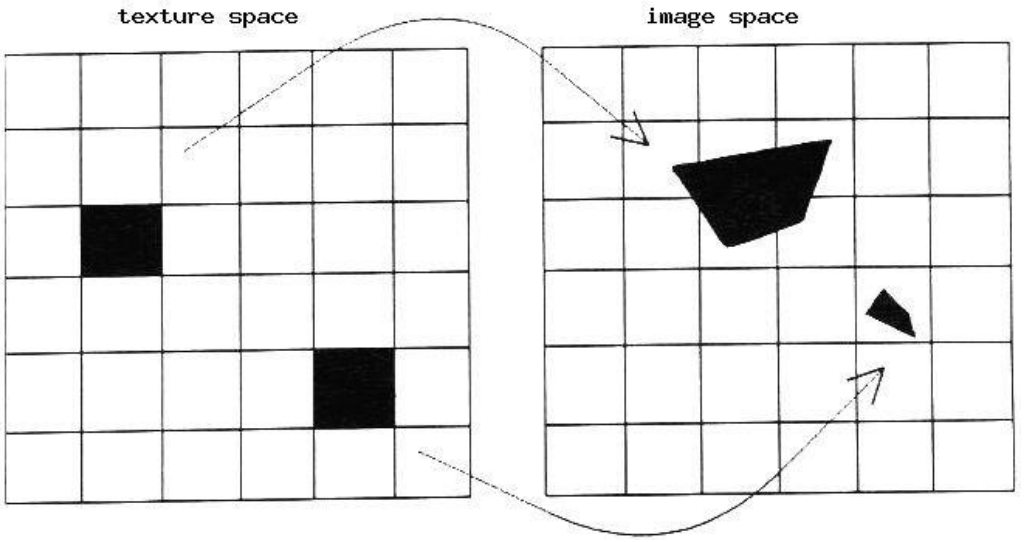
Uniform sampling of texture space does not guarantee

uniform sampling in screen spacespace

Possibly used if

- The texture-to-screen mapping is difficult to invert
- The texture image does not fit into memory

```
Texture scanning:
for v
  for u
    compute x(u,v) and y(u,v)
    copy TEX[u,v] to SCR[x,y]
(or in general
    rasterize image of TEX[u,v])
```



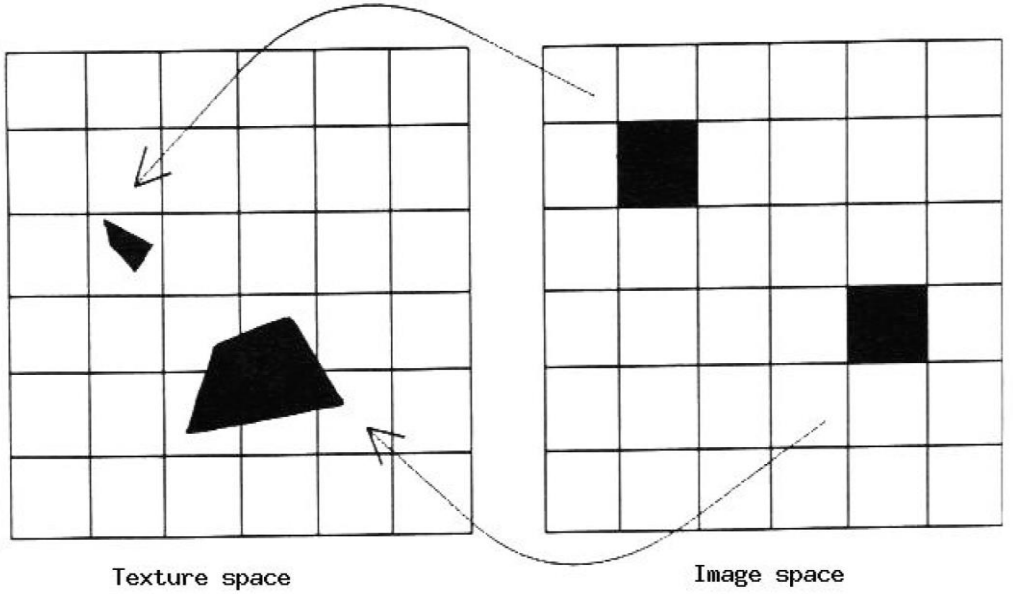
Requires inverting the mapping transformation

Preferable when the mapping is readily invertible and the texture image fits into memory

The most common mapping method

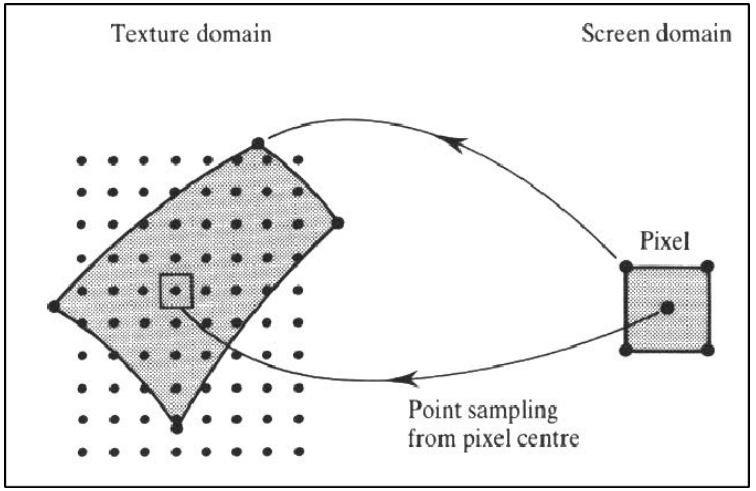
- For each pixel in screen space, the pre-image of the pixel in texture space is found and its area is integrated over

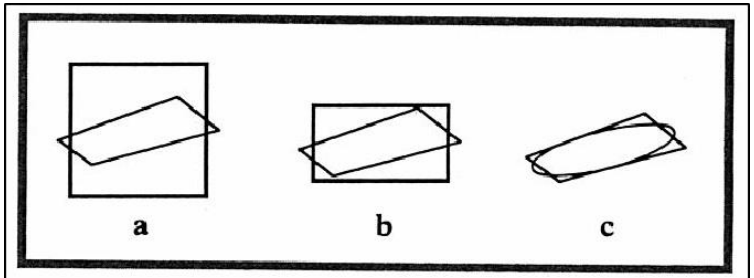
```
Texture scanning:
for y
  for x
    compute u(x,y) and v(x,y)
    copy TEX[u,v] to SCR[x,y]
(or in general
    Integrate over image of
    SCR[x,y])
```



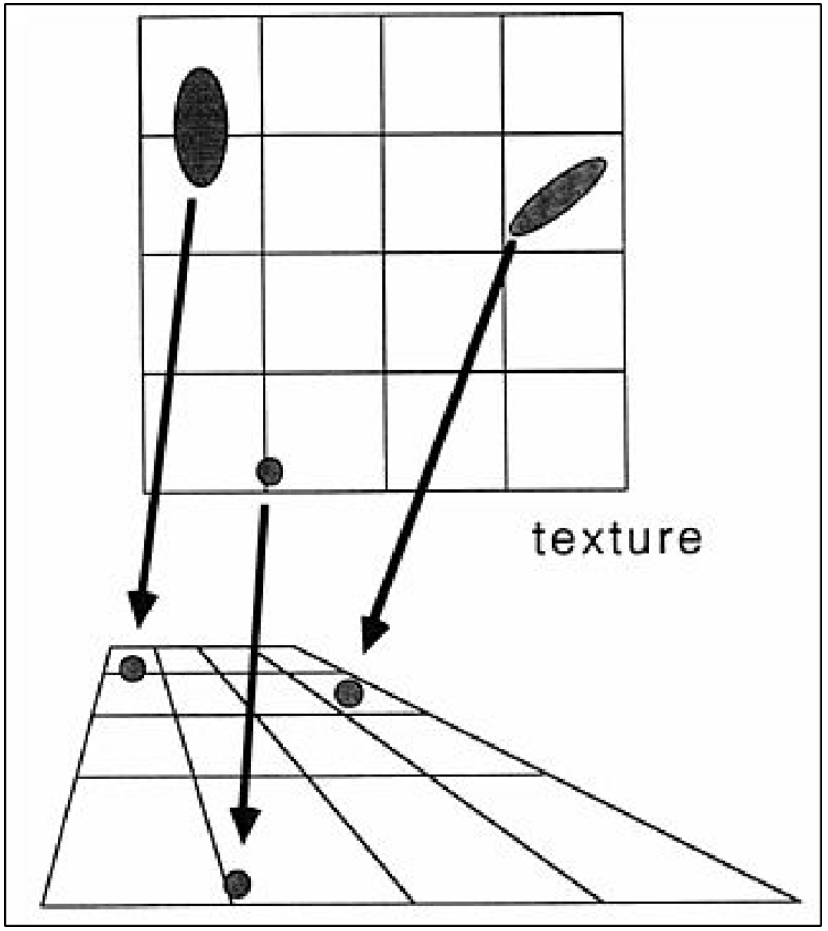
Pixel Pre-Image in Texture Space

A square screen pixel that intersects a curved surface has a curvilinear quadrilateral pre-image in texture space. Most methods approximate the true mapping by a quadrilateral or parallelogram. Or they take multiple samples within a pixel. If pixels are regarded as circles, their pre-images are ellipses





Approximating a quadrilateral texture area with (a) a square, (b) a rectangle, and (c) an ellipse. Too small an area causes aliasing; too large an area causes blurring.

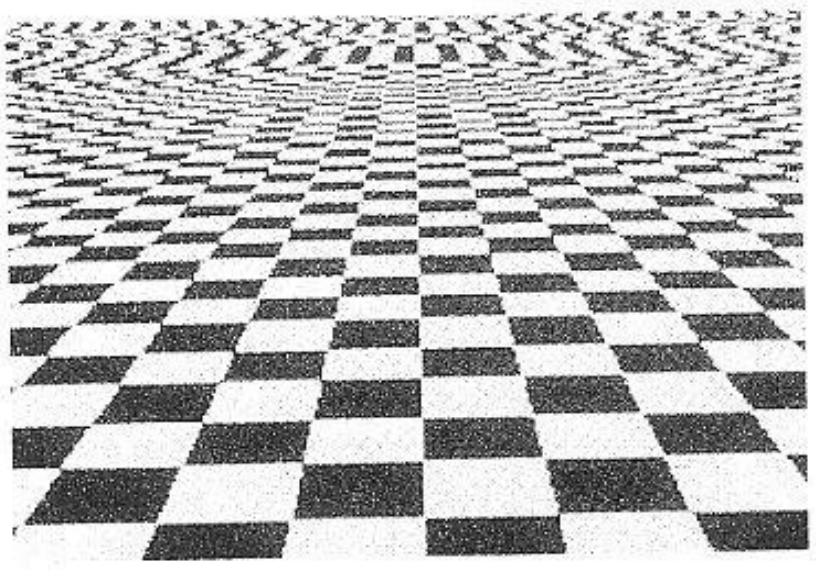


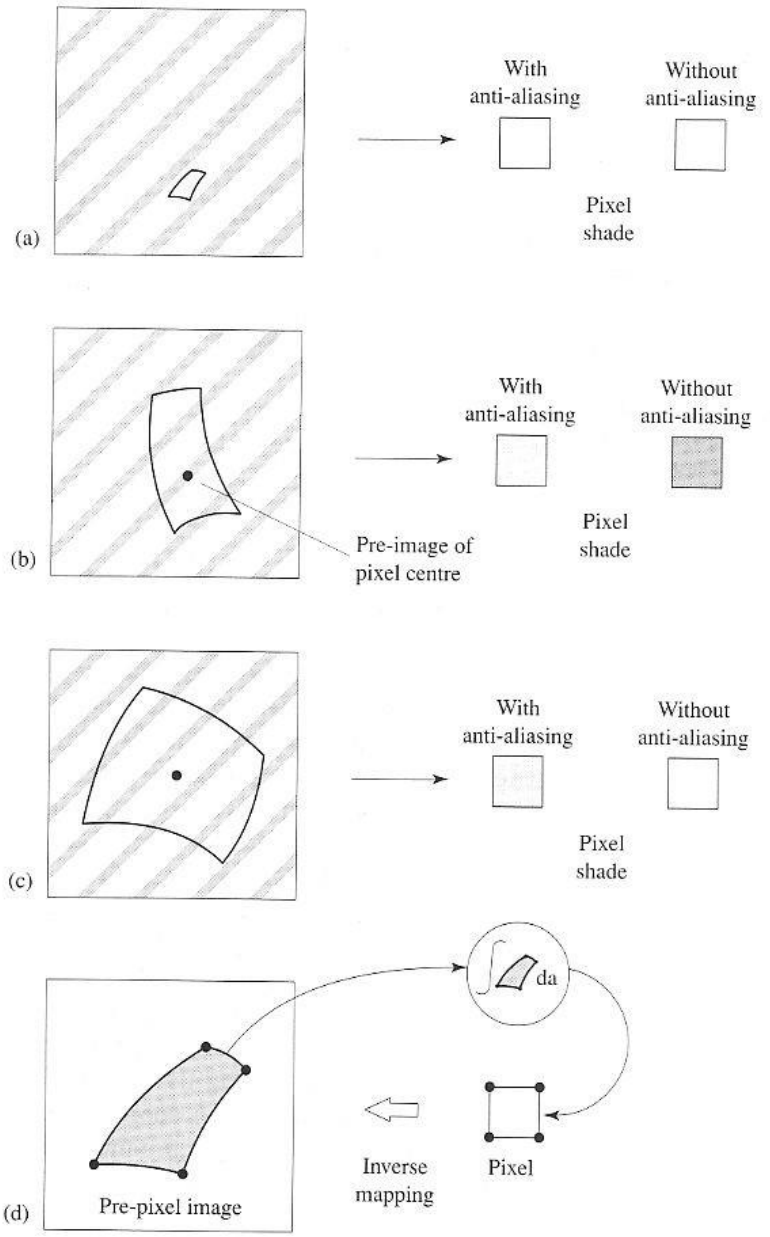
Integration of Pre-image

- Integration over pixel footprint in texture space

Aliasing

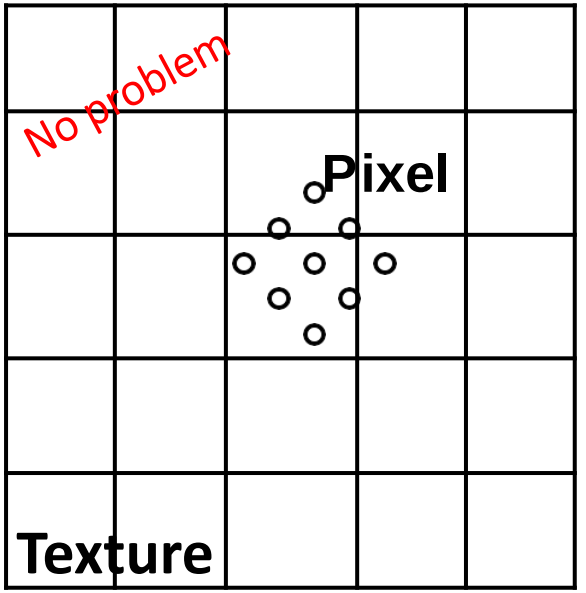
- Texture insufficiently sampled
- Incorrect pixel values
- “Randomly” changing pixels when moving





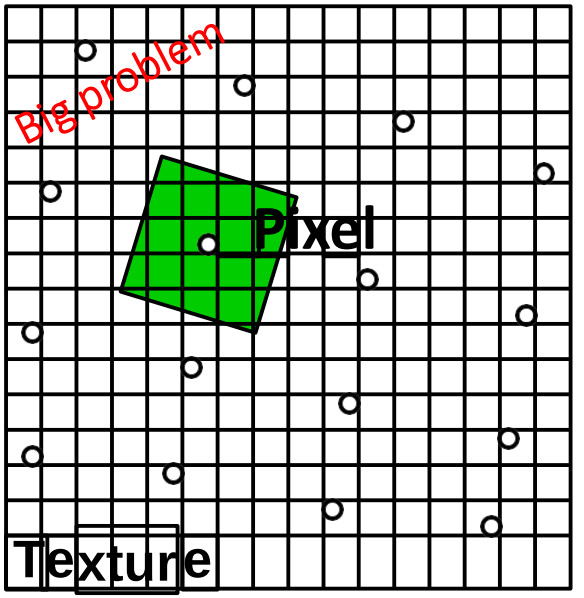
Magnification

- Map few texels onto many pixels
- Nearest:
  - Take the nearest texel
- Bilinear interpolation:
  - Interpolation between 4 nearest texels
  - Need fractional accuracy of coordinates



Minification

- Map many texels to one pixel
  - Aliasing:
    - Reconstructing high-frequency signals with low level frequency sampling
- Filtering
  - Averaging over (many) associated texels
  - Computationally expensive



Space-variant filtering

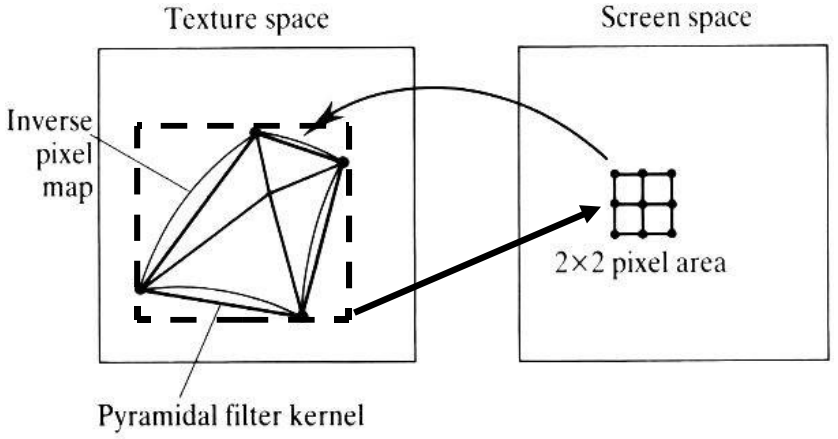
- Mapping from texture space (u, v) to screen space (x, y) not affine
- Filtering changes with position

Space-variant filtering methods

- Direct convolution
  - Numerically compute the Integral
- Pre-filtering
  - Precompute the integral for certain regions – more efficient
  - Approximate footprint with regions

Convolution in texture space

- Texels weighted according to distance from pixel center (*e.g.* pyramidal filter kernel)



Convolution in image space

1. Center the filter function on the pixel (the image space) and find its bounding rectangle.
2. Transform the rectangle to the texture space, where it is a quadrilateral. The sides of this rectangle are assumed to be straight. Find a bounding rectangle for this quadrilateral.
3. Map all pixels inside the texture space rectangle to screen space.
4. Form a weighted average of the mapped texture pixels using a two-dimensional lookup table indexed by each sample's location within the pixel.

Direct convolution methods are slow

- A pixel pre-image can be arbitrarily large along silhouettes or at the horizon of a textured plane
- Horizon pixels can require averaging over thousands of texture pixels
- Texture filtering cost grows in proportion to projected texture area

Speed up

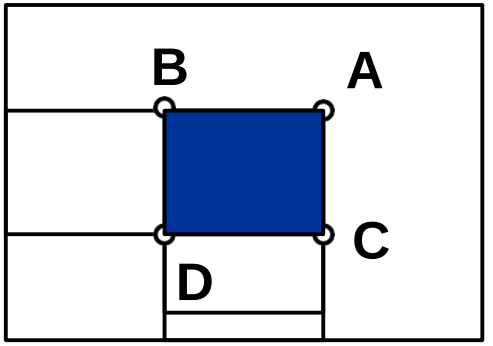
- The texture can be pre-filtered so that during rendering only a few samples will be accessed for each screen pixel

Two data structures are commonly used for pre-filtering:

- Integrated arrays (***summed area tables***)
- Image pyramids (***mipmaps***) Space-variant filtering

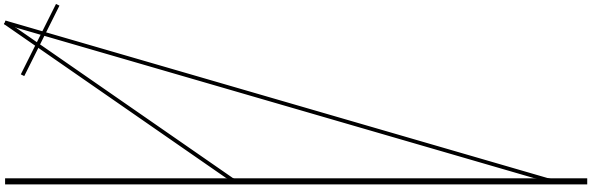
Summed-Area-Tables

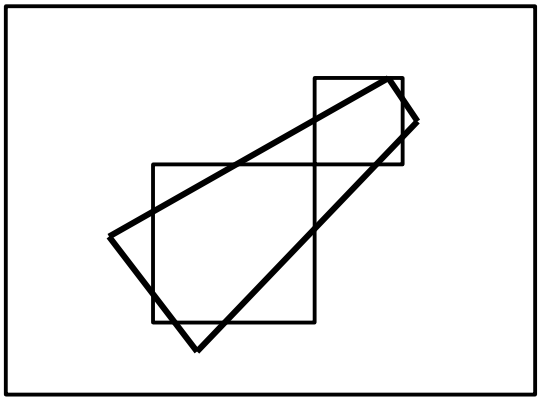
- Per texel, store sum from $(0, 0)$ to (u, v)
- Arbitrary rectangle:
  - Area = $A - B - C + D$
- Many bits per texel (sum!)



Footprint assembly

- Good for space variant filtering
  - *e.g.* inclined view of terrain
- Approximation of the pixel area by rectangular texel-regions
- The more footprints the better accuracy
- Often fixed number of texels because of economical reasons



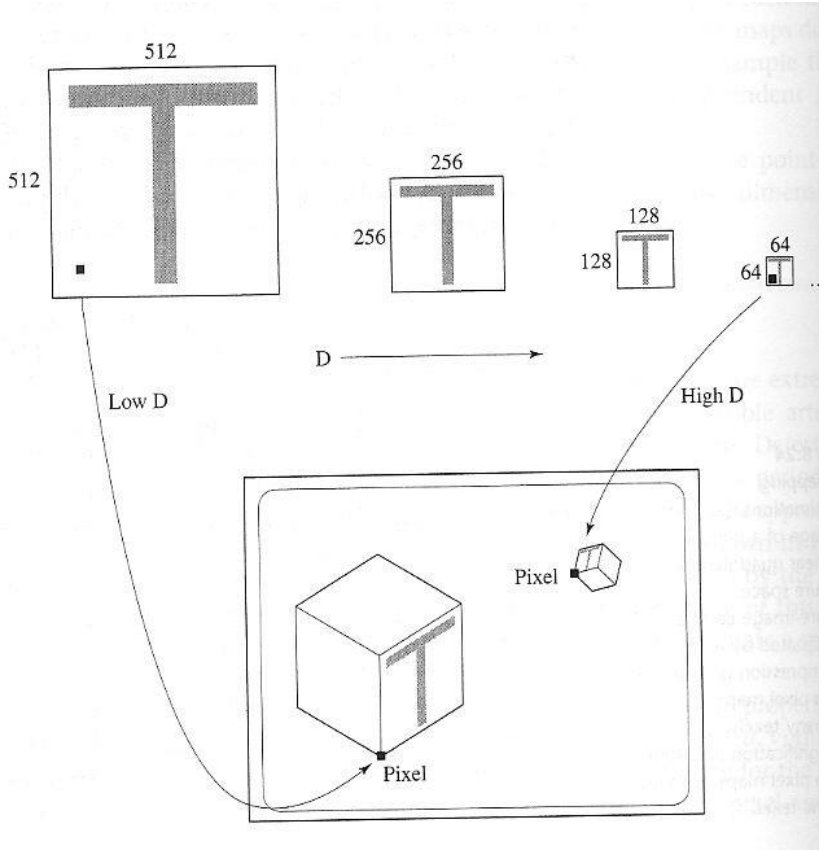


Texture available in multiple resolutions

- Pre-processing step

Rendering: select appropriate texture resolution

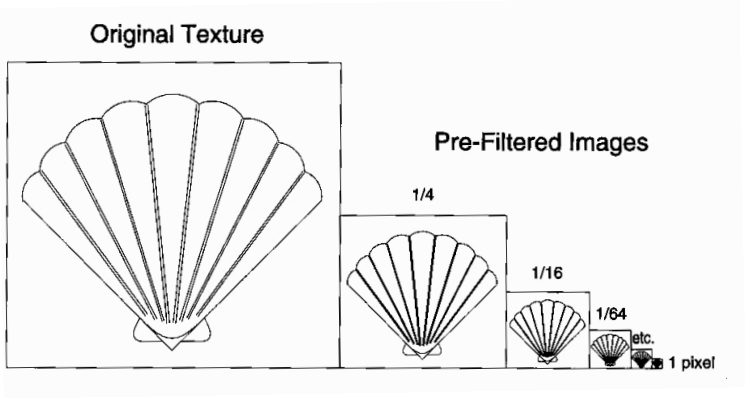
- Selection is usually per pixel!
- Texel size(n) < extent of pixel footprint < texel size(n+1)



Multum In Parvo (MIP): much in little

Hierarchical resolution pyramid

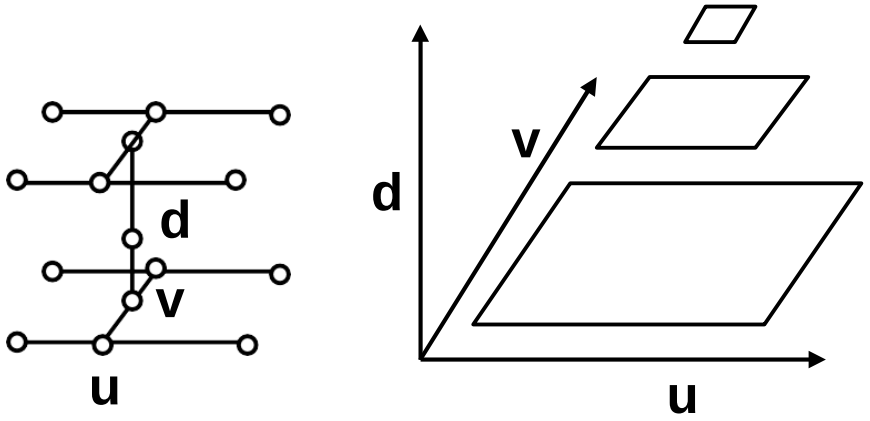
- Repeated averaging over 2x2 texels



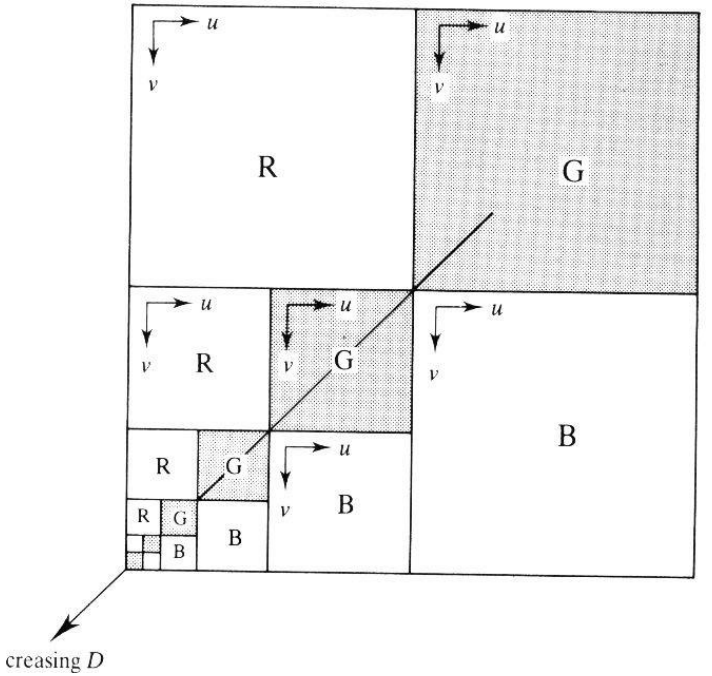
Rectangular arrangement (RGB)

Reconstruction

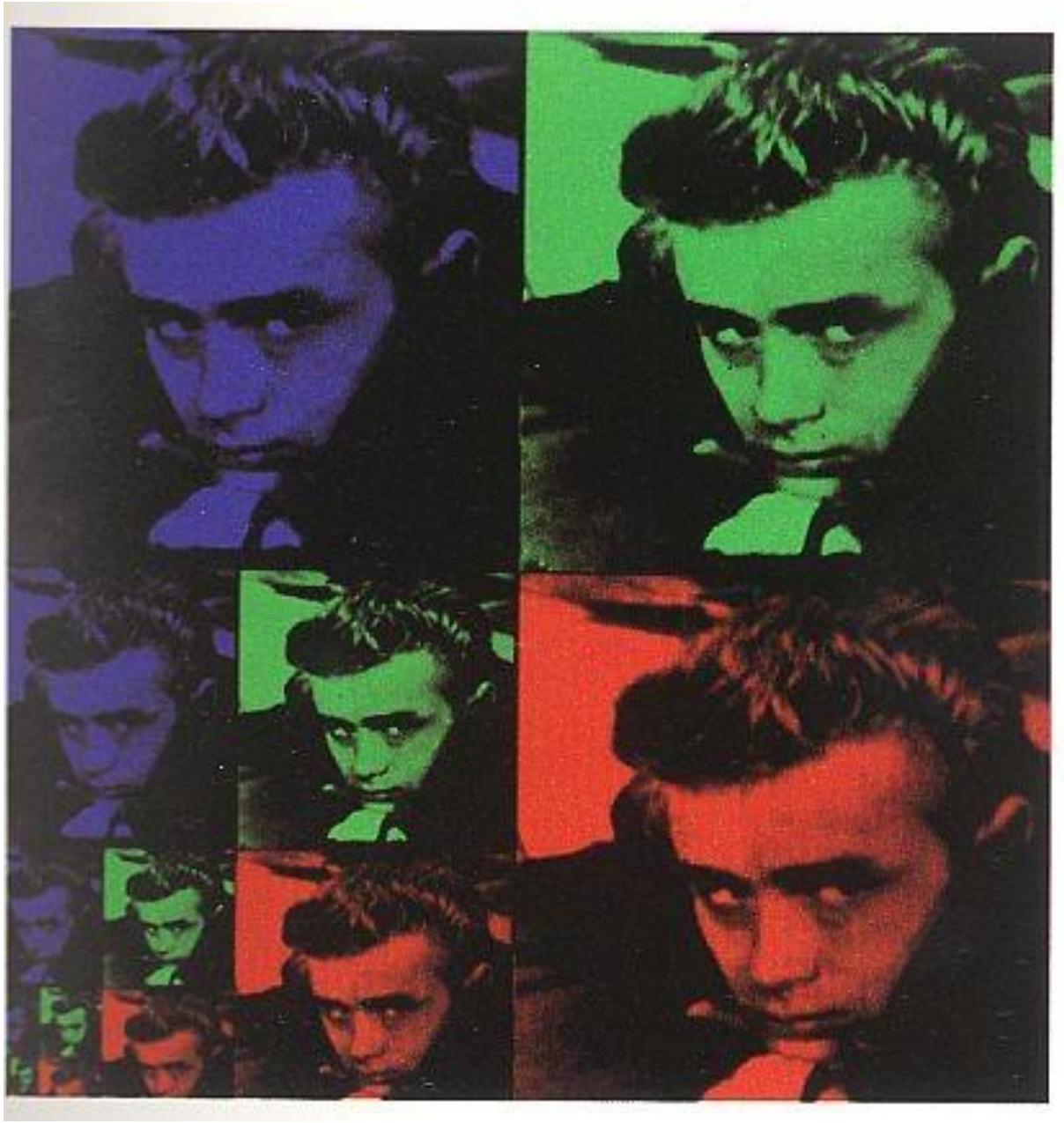
- Tri-linear interpolation of 8 nearest texels
  - Bilinear interpolation in levels *n* and *n+1*
  - Linear interpolation between the two levels



- "Brilinear": Trilinear only near transitions
  - Avoid reading 8 texels, most of the time



Example:



Why do graphics cards get faster by MipMapping?

- Bottleneck is memory bandwidth
- Using of texture caches
- Texture minification required for far geometry

No MipMap

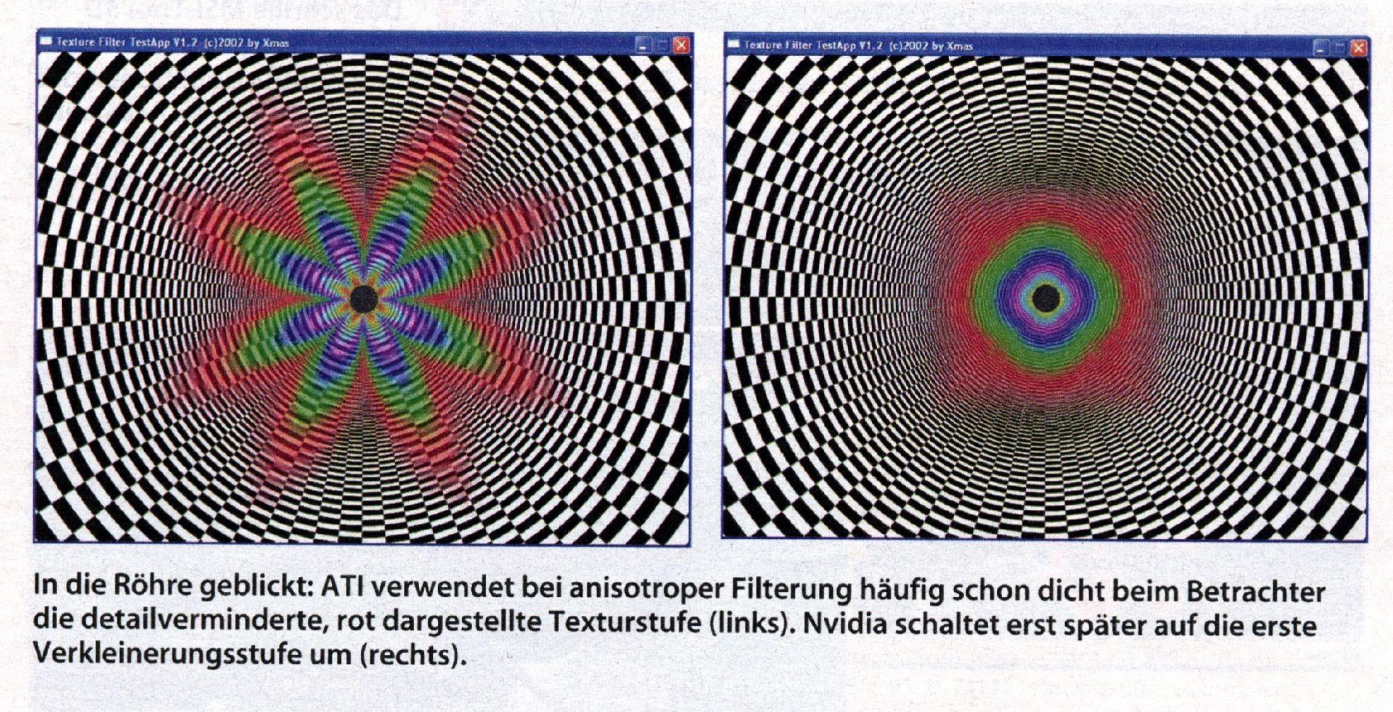
- „Random“ access to texture
- Always 4 new texels

MipMap

- Neighboring pixel moves nearly about one texel
- Access to 8 texels at a time, but
  - Nearly all texels are still in the cache
  - Average: < 1.5 new texels per pixel

Footprint Assembly on GPUs

- Integration Across Footprint of Pixel
- HW: Choose samples that best approximate footprint
- Weighted average of samples



In die Röhre geblickt: ATI verwendet bei anisotroper Filterung häufig schon dicht beim Betrachter die detailverminderte, rot dargestellte Texturstufe (links). Nvidia schaltet erst später auf die erste Verkleinerungsstufe um (rechts).