# Computer Graphics

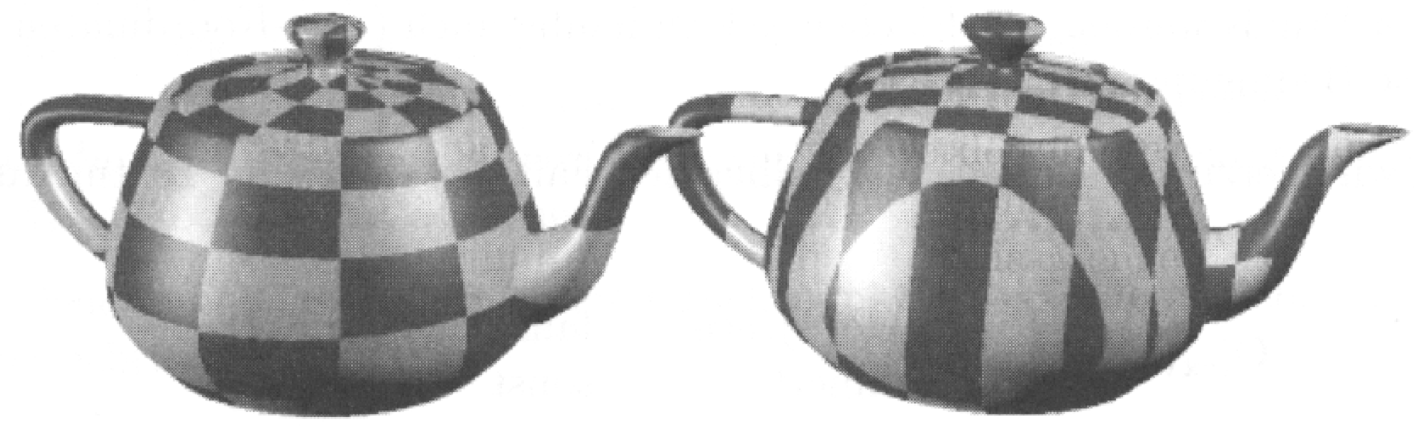
## - Texturing & Procedural Methods -

# Overview

- **Last time**
  - Shading
  - Texturing
- **Today**
  - Texturing (Cont.)
  - Texture synthesis
  - Procedural textures
  - Fractal landscapes
  - Volume effects
- **Next lecture**
  - Alias & signal processing

# Surface Parameterization

- **To apply textures we need 2D coordinates on surfaces**
  ➔ Parameterization
- **Some objects have a natural parameterization**
  – Sphere: spherical coordinates $(\varphi, \theta) = (2\pi\, u, \pi\, v)$
  – Cylinder: cylindrical coordinates $(\varphi, z) = (2\, \pi\, u, H\, v)$
  – Parametric surfaces (such as B-spline or Bezier surfaces → later)
- **Parameterization less obvious for**
  – Polygons, implicit surfaces, …



# Triangle Parameterization

- **Piecewise planar object surface patches**
  - Has implicit parameterization (e.g. barycentric coordinates)
  - But we need more control: Placement of triangle in texture space
- **Assign texture coordinates** $(u,v)$ **to each vertex** $(x_o,y_o,z_o)$
- **Apply viewing projection** $(x_o,y_o,z_o) \rightarrow (x,y)$
- **Yields texture transformation (warping)** $(u,v) \rightarrow (x,y)$

$$x = \frac{au + bv + c}{gu + hu + i} \qquad y = \frac{du + ev + f}{gu + hv + i}$$

  - In homogeneous coordinates

$$(x, y) = (x'/w, y'/w) \qquad \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} u' \\ v' \\ q \end{bmatrix}$$

$$(u, v) = (u'/q, v'/q)$$

  - Transformation coefficients determined by 3 pairs $(u,v) \rightarrow (x,y)$
  - Invertible if points are non-collinear

# Triangle Parameterization II

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} u' \\ v' \\ q \end{bmatrix}$$

- **Inverse transform** $(x,y) \rightarrow (u,v)$

$$\begin{bmatrix} u' \\ v' \\ q \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} ei-fh & ch-bi & bf-ce \\ fg-di & ai-cg & cd-af \\ dh-eg & bg-ah & ae-bd \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w \end{bmatrix}$$

$$(u,v) = (u'/q, v'/q) \qquad (x', y', w) = (x, y, 1)$$

- **Coefficients must be calculated for each triangle**
- **Scan-line rendering**
  - Incremental bilinear interpolation of $(u',v',q)$ in screen space
- **Ray tracing**
  - Evaluation at each intersection

# Cylinder Parameterization

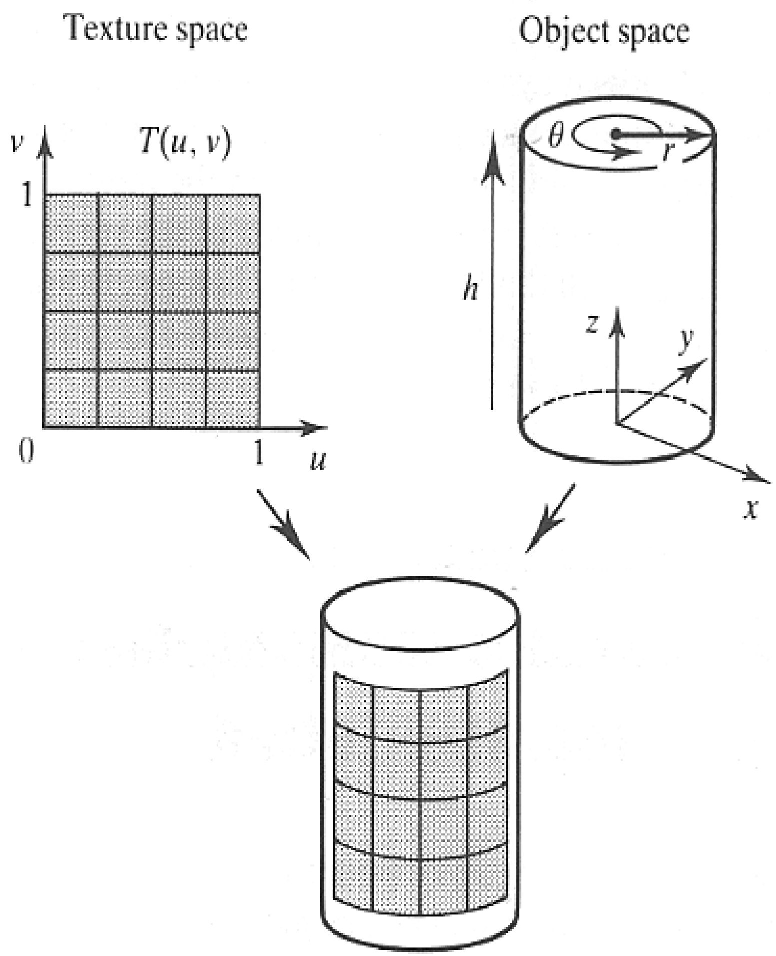
**Transformation from texture space to the cylinder parametric representation can be written as:**

$$(\theta, h) = (2\pi u, vH)$$

**where $H$ is the height of the cylinder.**

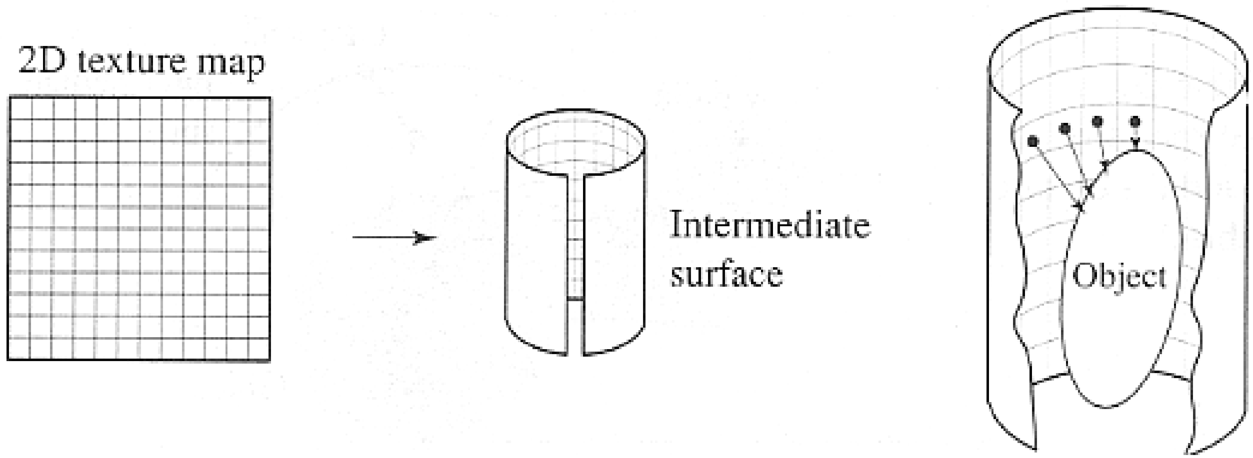
**The surface coordinates in the Cartesian reference frame can be expressed as:**

$$x_o = r\cos\theta,$$
$$y_o = r\sin\theta,$$
$$z_o = h$$



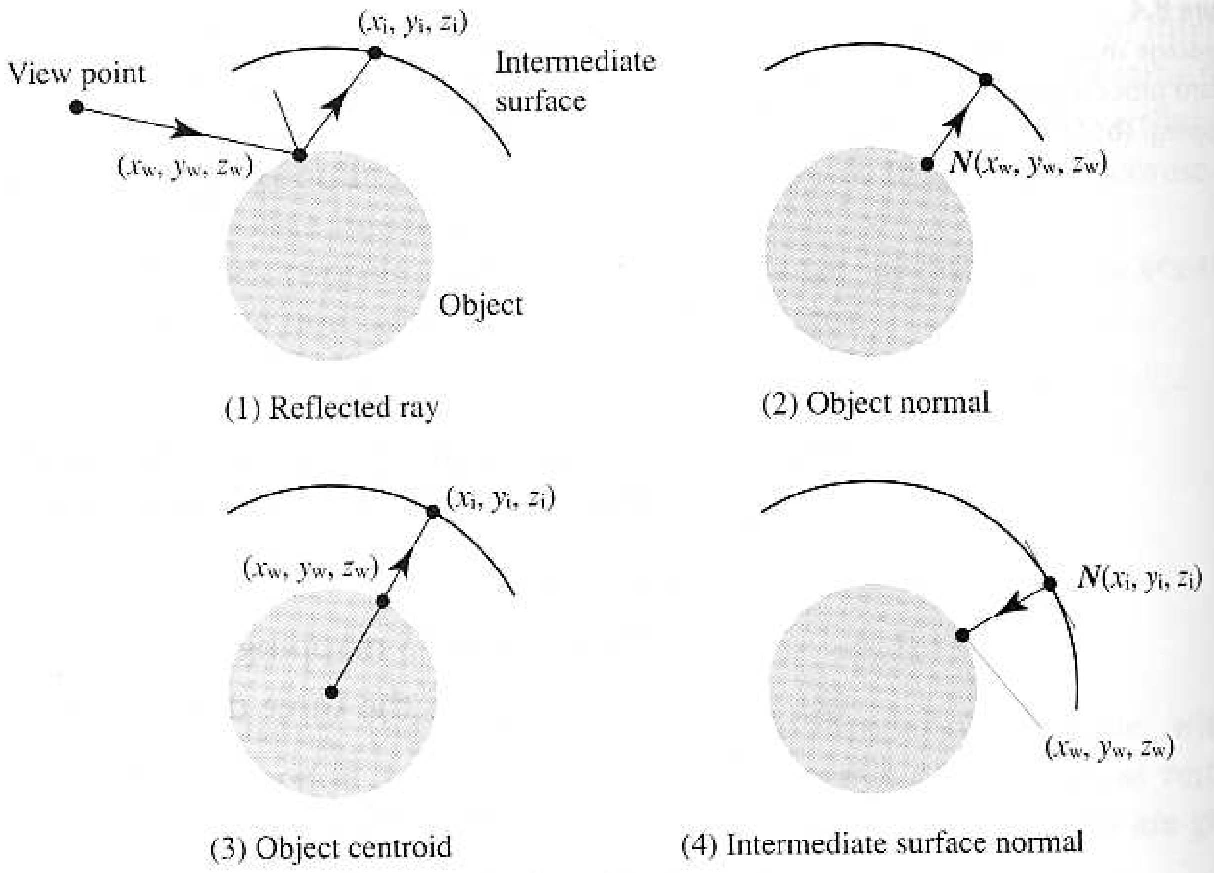
# Two-Stage Mapping

- **Inverse Mapping for arbitrary 3D surfaces too complex**
- **Approximation technique is used:**
  - Mapping from 2D texture space to a simple 3D intermediate surface, which is a reasonable approximation of  the destination surface (e.g., cylinder, sphere) (S mapping)
  - Mapping from the intermediate surface to the destination object surface (O mapping)



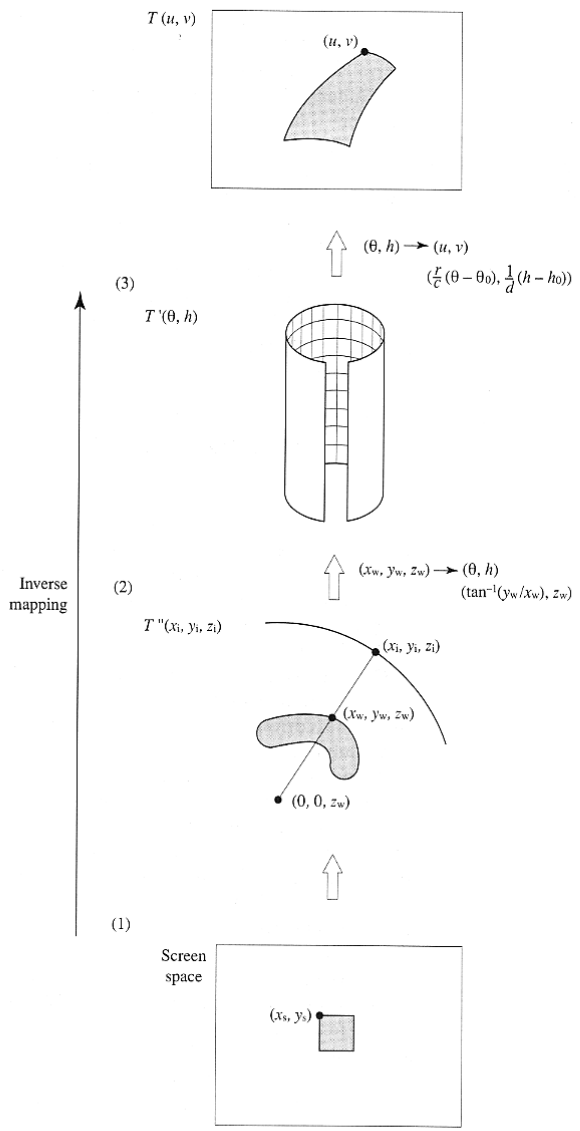
# O-Mapping

- **Determine point on intermediate surface through**
  - Reflected view ray
    - Reflection or environment mapping
  - Normal mapping
  - Line through object centroid
  - Shrinkwrapping
    - Forward mapping
    - Normal mapping from intermediate surface



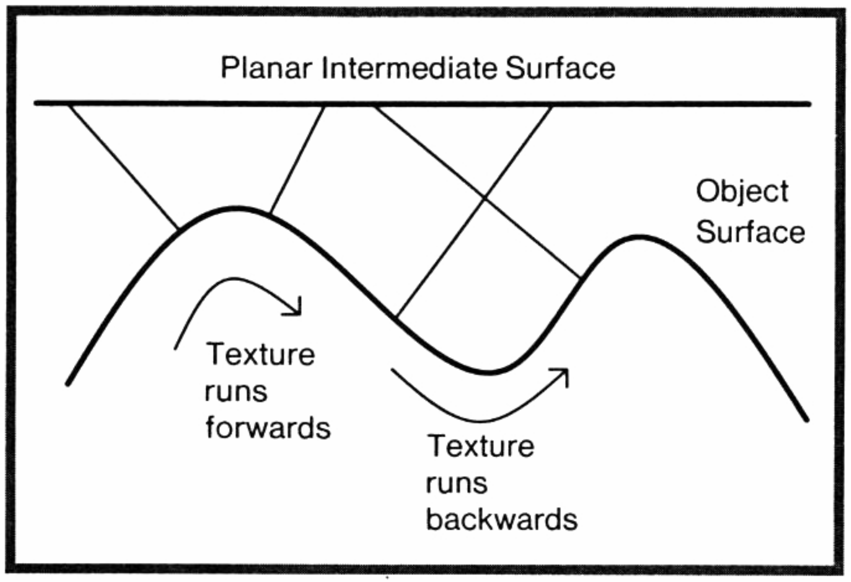
# Shrinkwrap Mapping

- **Inverse two-stage mapping**
- **Map 4 screen pixels to object surface**
- **O-mapping**
  - Shrinkwrapping: Intersection of line from cylinder axis through object point
- **S-mapping**
  - Inverse-map cylinder surface to texture map



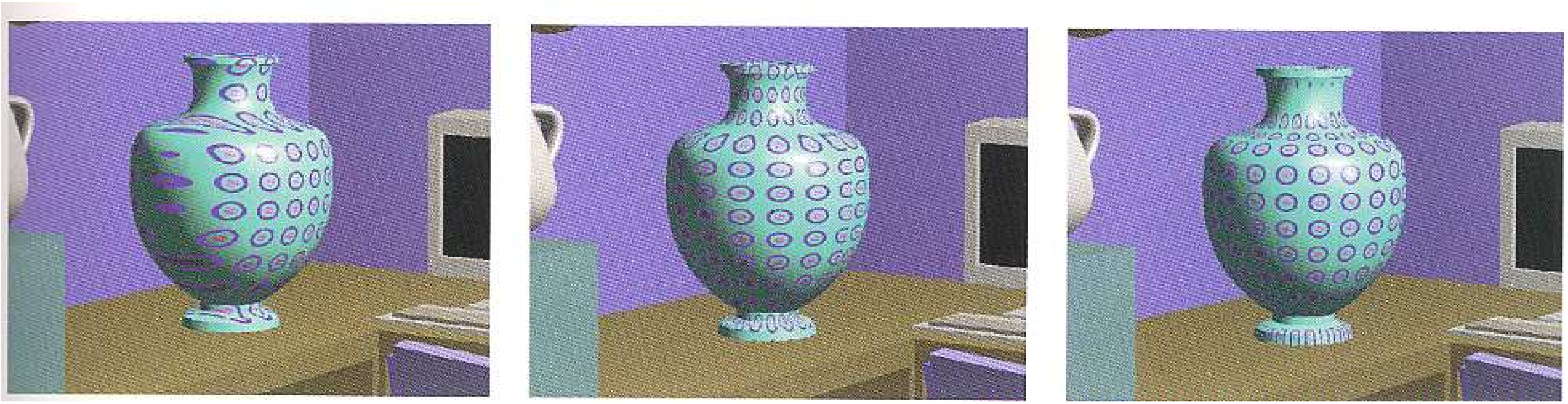
# Two-Stage Mapping: Problems

- **Problems**
  - May introduce undesired texture distortions if the intermediate surface differs much from the destination surface
  - Still often used in practice because of its simplicity




**Surface concavities can cause the texture pattern to reverse if the object normal mapping is used.**

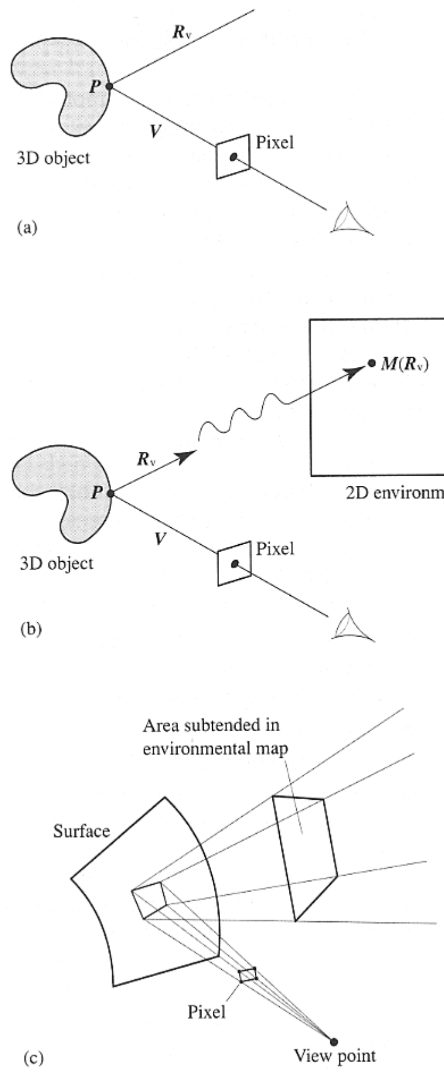
# Two-Stage Mapping: Example



- **Different intermediate surfaces**
- **Plane**
  – Strong distortion where object surface normal ⊥ plane normal
- **Cylinder**
  – Reasonably uniform mapping (symmetry !)
- **Sphere**
  – Problems with concave regions

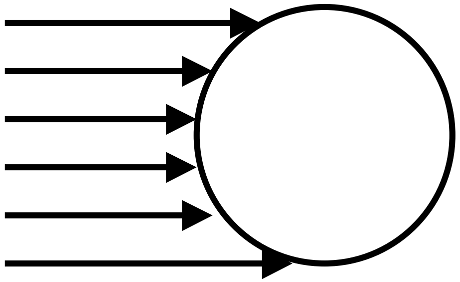
# Reflection Mapping

- **Also called Environment Mapping**
- **Mirror reflections**
  - Surface curvature: beam tracing
  - Map filtering
- **Reflection map parameterization**
  - Intermediate surface in 2-stage mapping
- **Light sources distant from object**
  - Parallax-free illumination
  - No self-reflections, object concavities
- **Option: Separate map per object**
  - Reflections of other objects
  - Maps must be recomputed after changes



# Reflection Map Acquisition

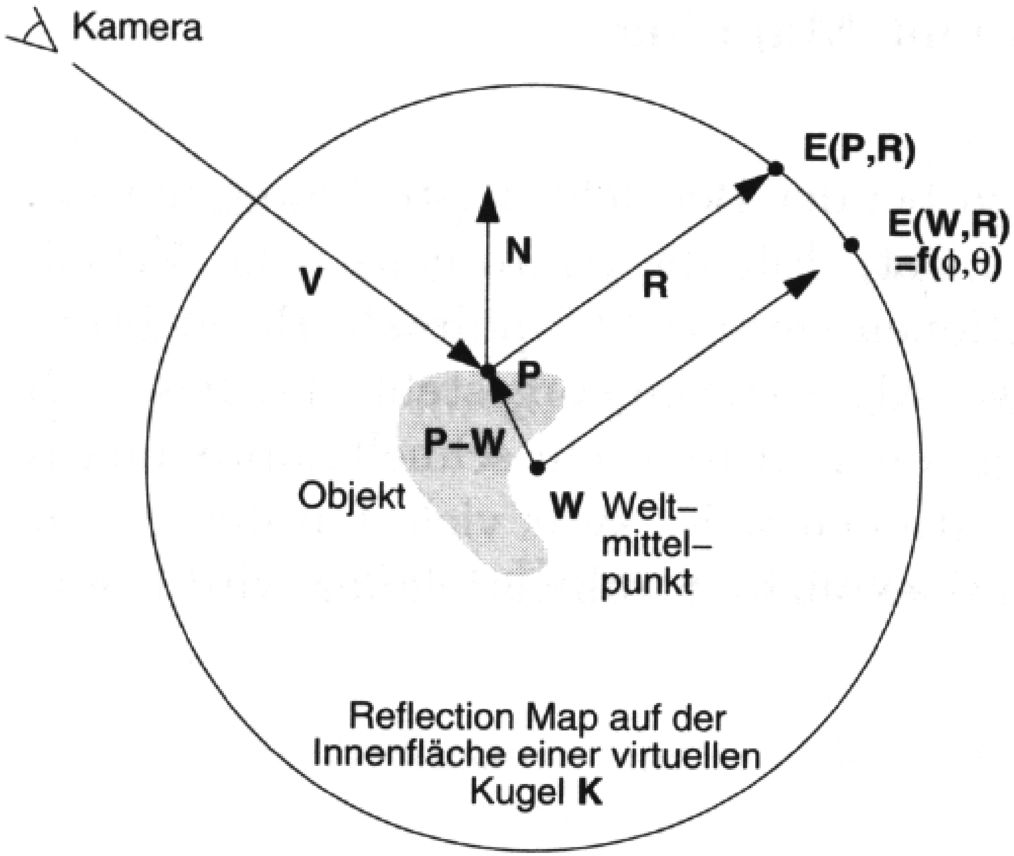
- **Generating spherical maps (original 1982/83)**
  - i.e. photo of a reflecting sphere (gazing ball)



Peter Chou

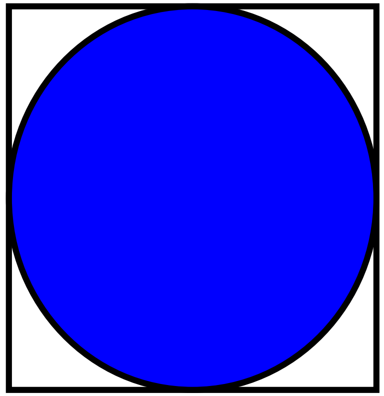
# Reflection Map Rendering

- **Spherical parameterization**
- **O-mapping using reflected view ray intersection**

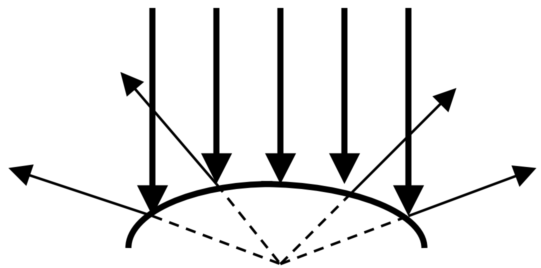


# Reflection Map Parameterization

- **Spherical mapping**
  - Single image
  - Bad utilization of the image area
  - Bad scanning on the edge
  - Artifacts, if map and image do not have the same direction

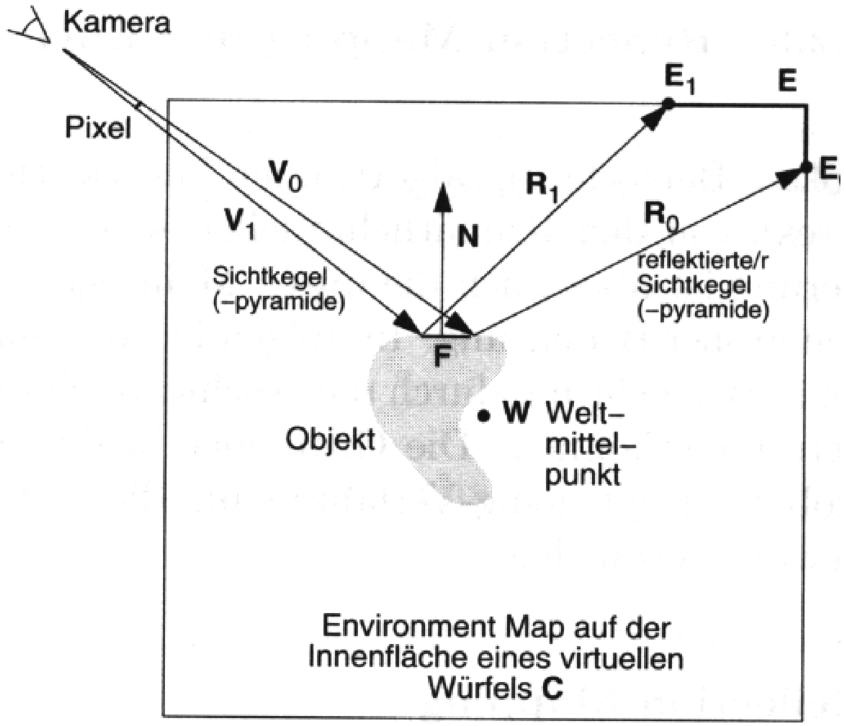


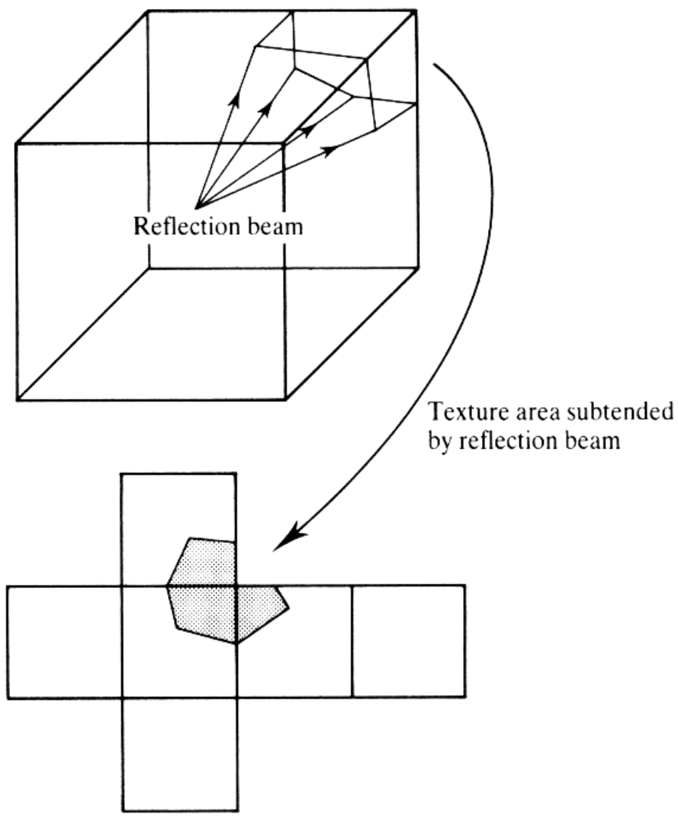
- **Parabolic mapping**
  - Subdivide in 2 images (facing and back facing side)
  - Less bias on the edge
  - Arbitrarily reusable
  - Supported by OpenGL extensions



# Reflection Map Parameterization

- **Cubical environment map, cube map, box map**
  - Enclose object in cube
  - Images on faces are easy to compute
  - Poorer filtering at edges
  - Support in OpenGL



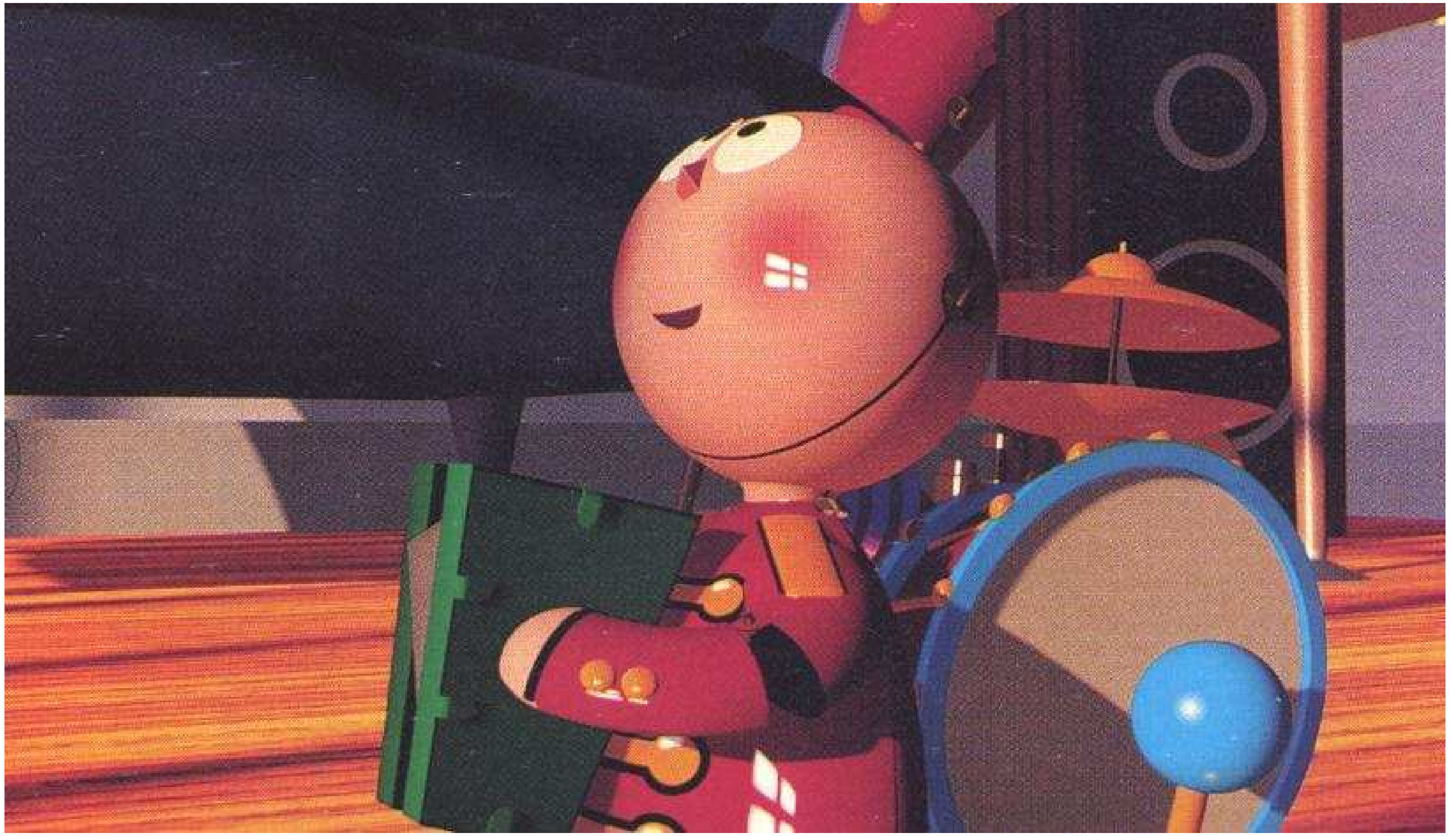



# Reflection Mapping

Terminator II motion picture

# Reflection Mapping Example II

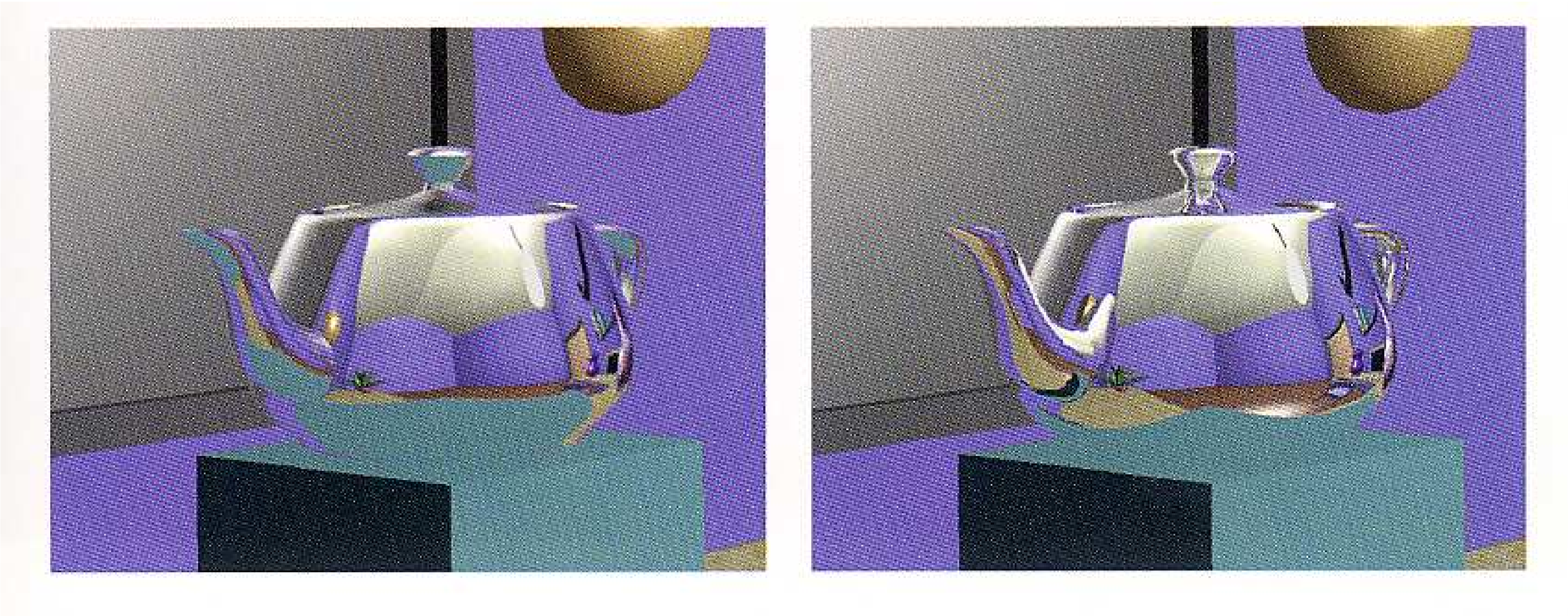
- **Reflection mapping with Phong reflection**
  - Two maps: diffuse & specular
  - Diffuse: index by surface normal
  - Specular: indexed by reflected view vector



RenderMan Companion

# Ray Tracing vs. Reflection Mapping

- **Differences ?**



# Recursive Ray Tracing

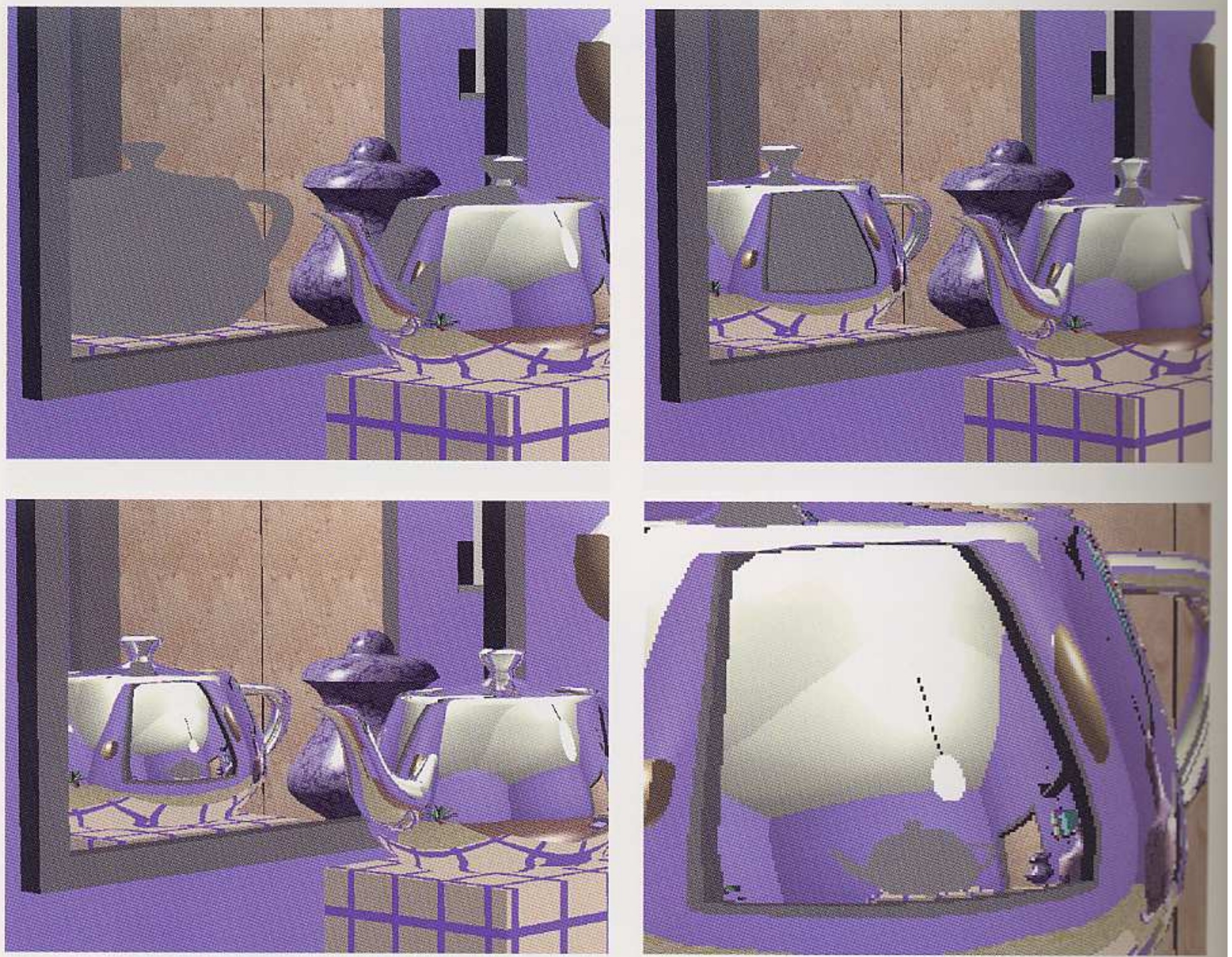
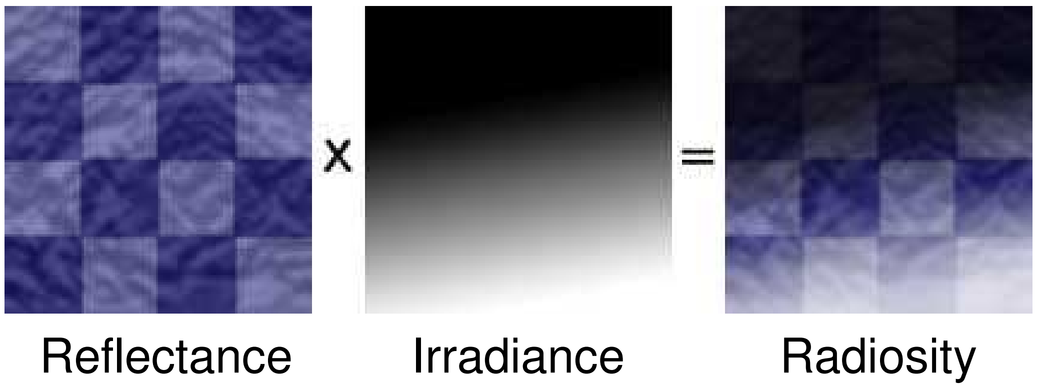
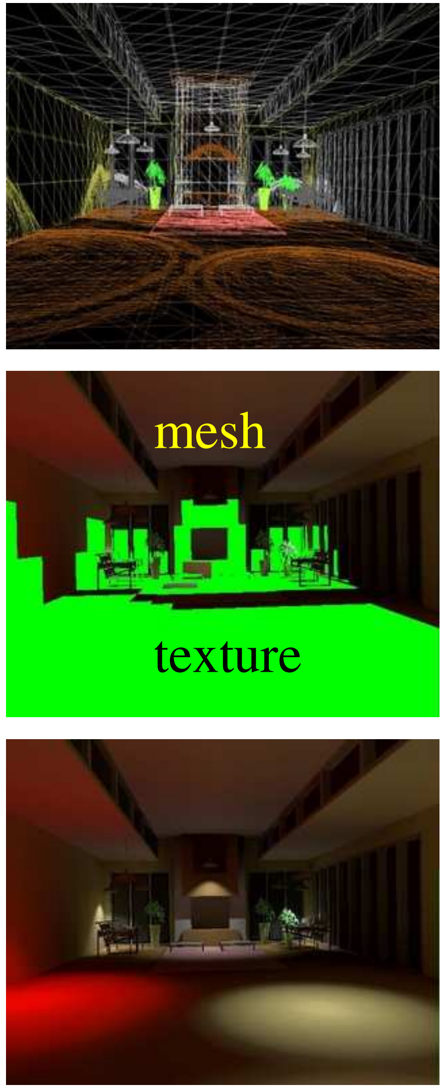


**Figure 18.11**
A recursive depth demonstration. The trace terminates at depth 2, 3, 4 and 5 (zoom image) respectively. 'Unassigned' pixels are coloured grey. Bad aliasing as a function of recursive depth (the light cable) is apparent.

# Light Maps

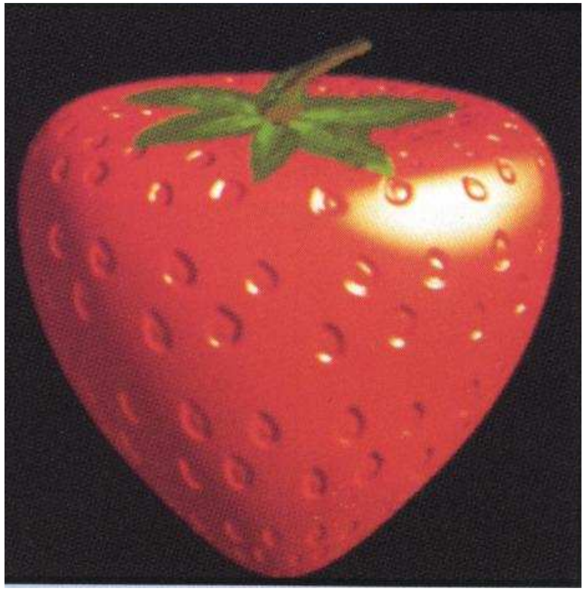
- **Light maps (i.e. in Quake)**
  - Pre-calculated illumination (local irradiance)
    - Often very low resolution
  - Multiplication of irradiance with base texture
    - Diffuse reflectance only
  - Provides surface radiosity
    - View-independent
  - Animated light maps
    - Animated shadows, moving light spots etc.

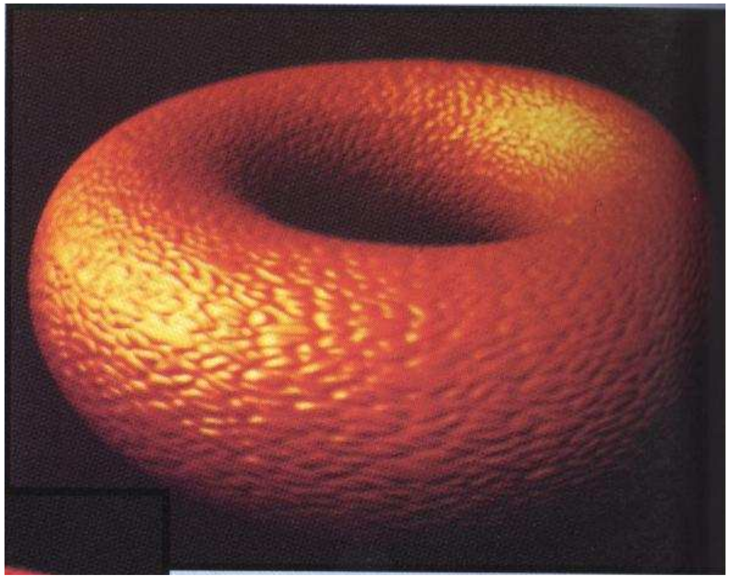


# Bump Mapping

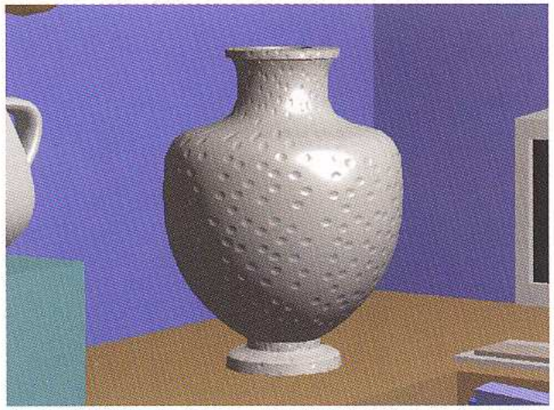
- **Modulation of the normal vector**
  - Surface normals changed only
    - Influences shading only
    - No self-shadowing, contour is ***not*** altered

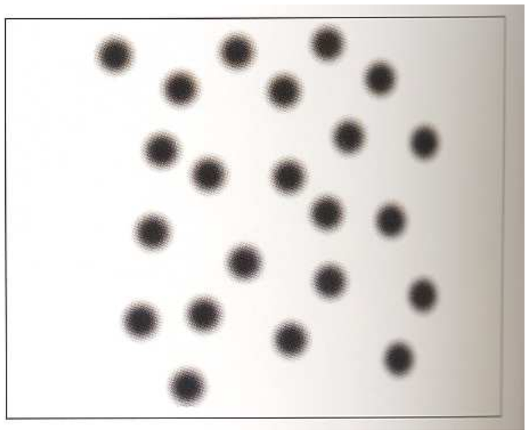


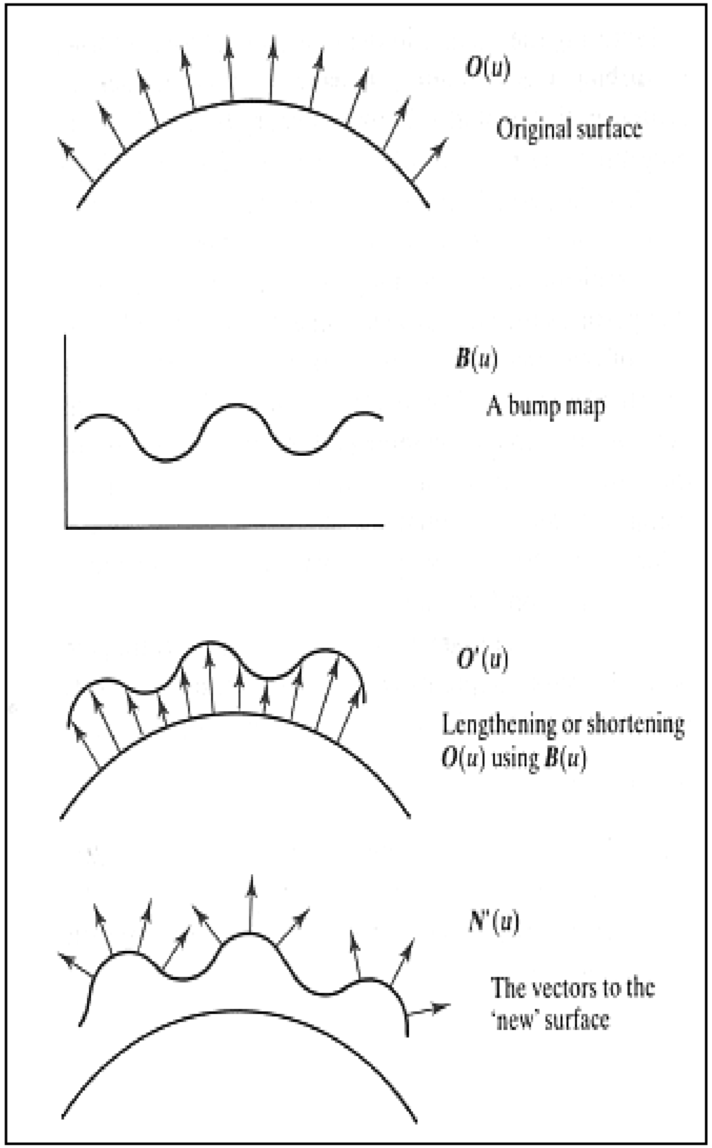


# Bump Mapping II

- **Original surface** $\boldsymbol{O(u,v)}$
  - Surface normals known
- **Bump map** $\boldsymbol{B(u,v) \in R}$
  - Surface is offset in normal direction according to bump map intensity
  - New normal directions are calculated $N'(u,v)$ based on displaced surface $O'(u,v)$
  - Originals surface is rendered with new normals $N'(u,v)$







# Bump Mapping IV

$$O'(u,v) = O(u,v) + B(u,v)\frac{N}{|N|}$$

Now differentiating this equation gives:

$$O'_u = O_u + B_u\frac{N}{|N|} + B\left(\frac{N}{|N|}\right)_u$$

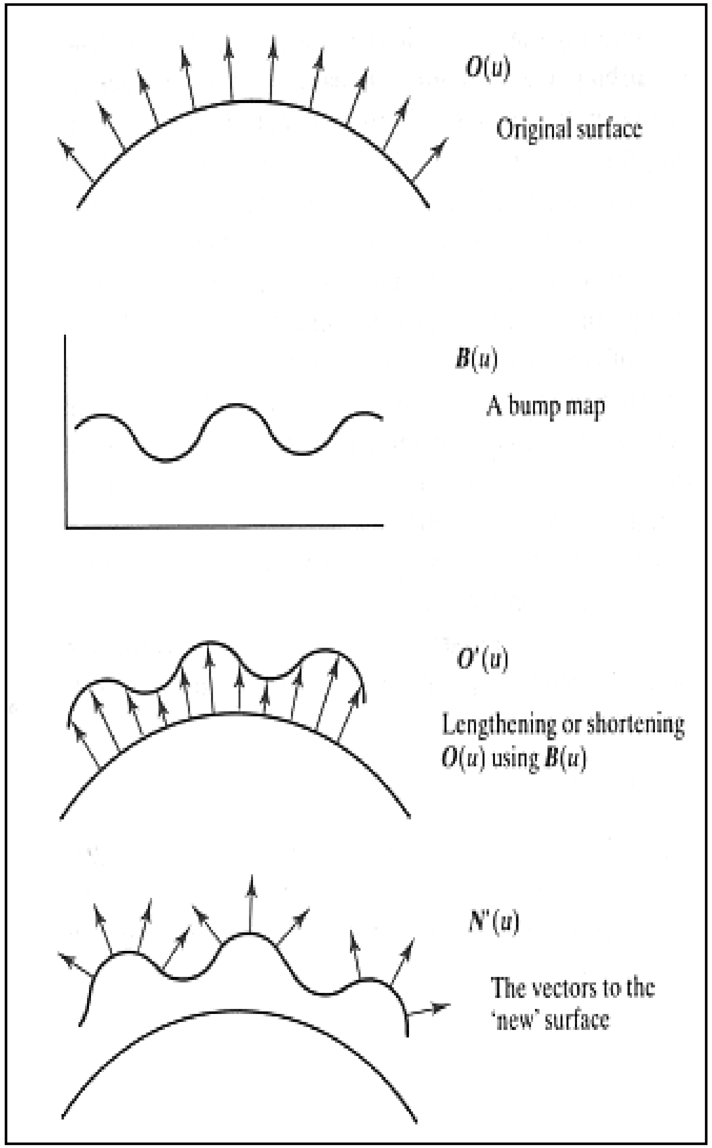
$$O'_v = O_v + B_v\frac{N}{|N|} + B\left(\frac{N}{|N|}\right)_v$$

If $B$ is small (that is, the bump map displacement function is small compared with its spatial extent) the last term in each equation can be ignored and

$$N'(u,v) = O_u \times O_v + B_u\left(\frac{N}{|N|} \times O_v\right) + B_v\left(O_u \times \frac{N}{|N|}\right) + B_u B_v\left(\frac{N \times N}{|N|^2}\right)$$

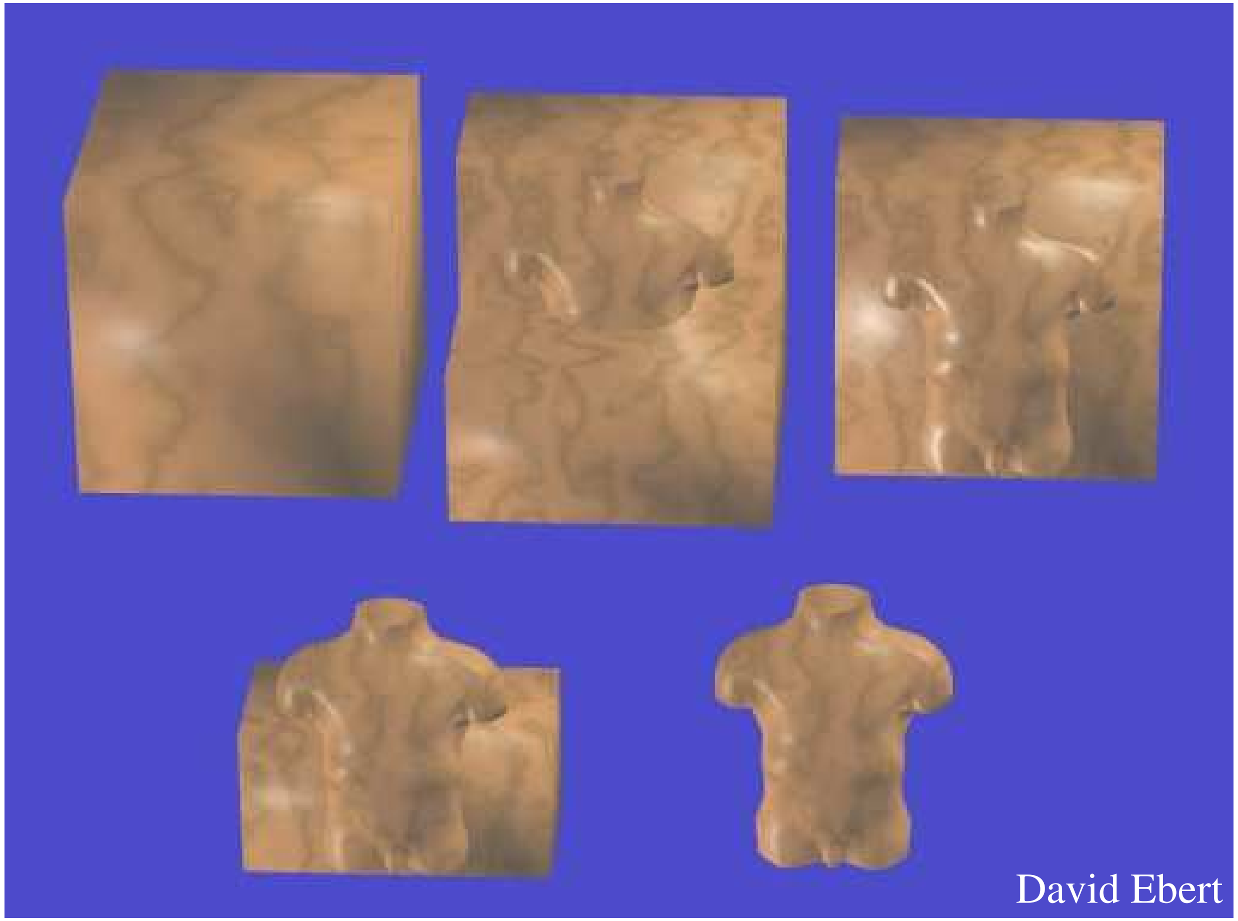
The first term is the normal to the surface and the last term is zero, giving:

$$D = B_u(N \times O_v) - B_v(N \times O_u)$$



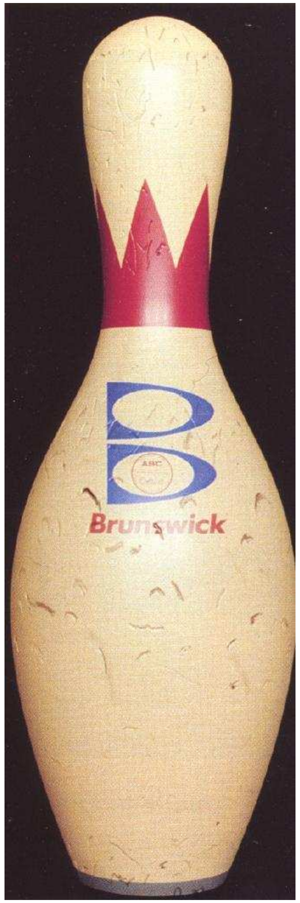
# 3-D Textures

- **“Carving object shape out of material block”**



# Texture Examples

- **Complex optical effects**
  - Combination of multiple textures





RenderMan Companion

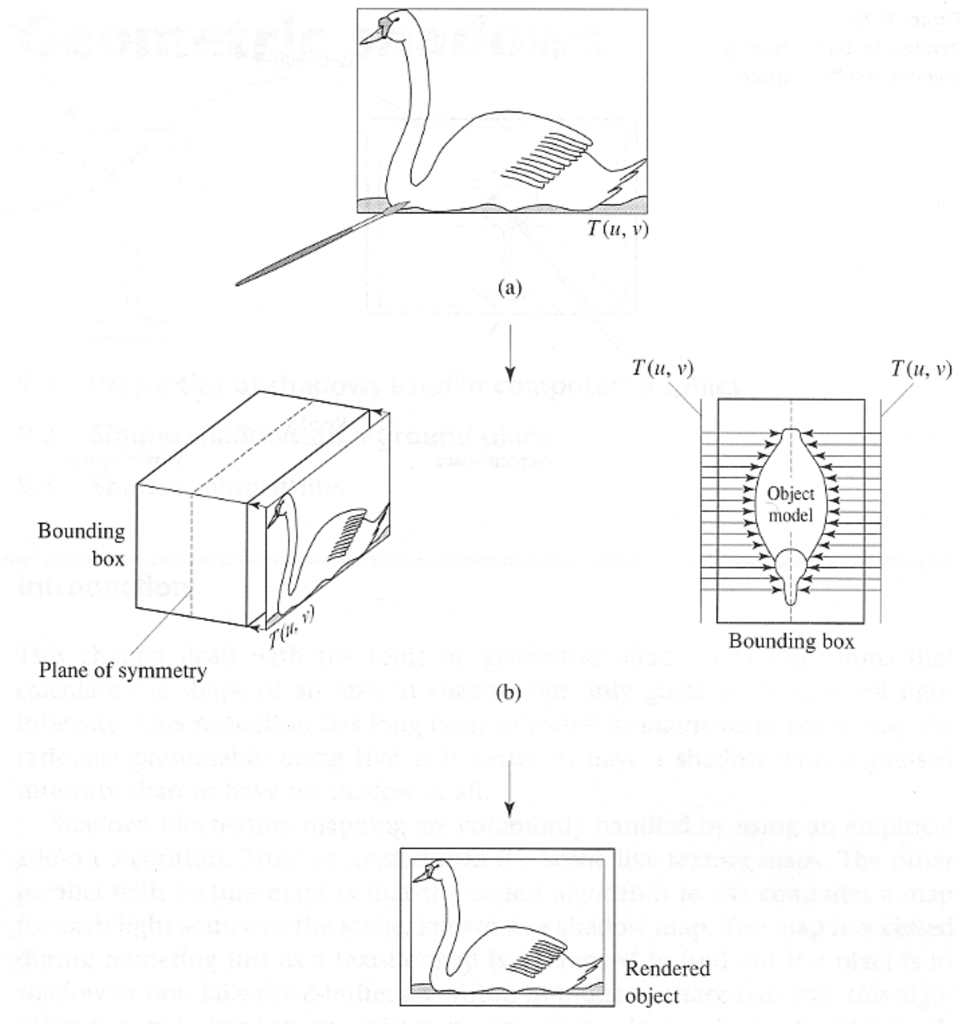
# Texture Examples

- **Solid 3D textures (wood, marble)**
- **Bump map (middle)**

RenderMan Companion

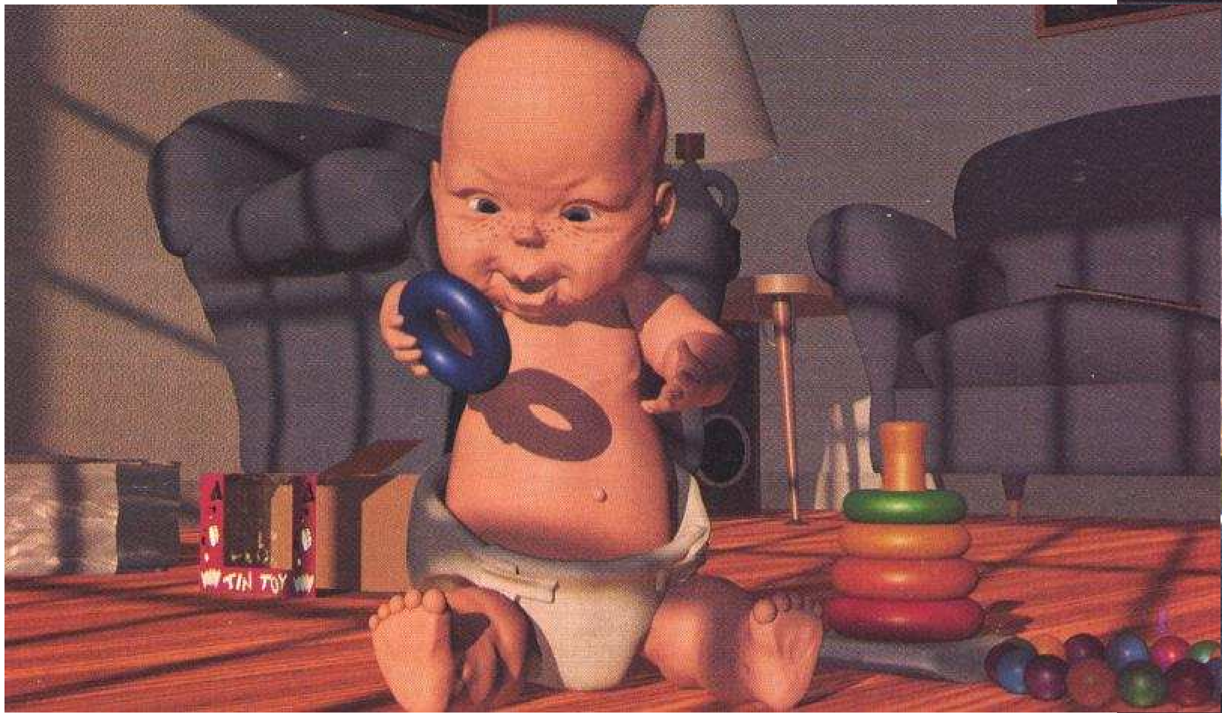
# Projective Textures

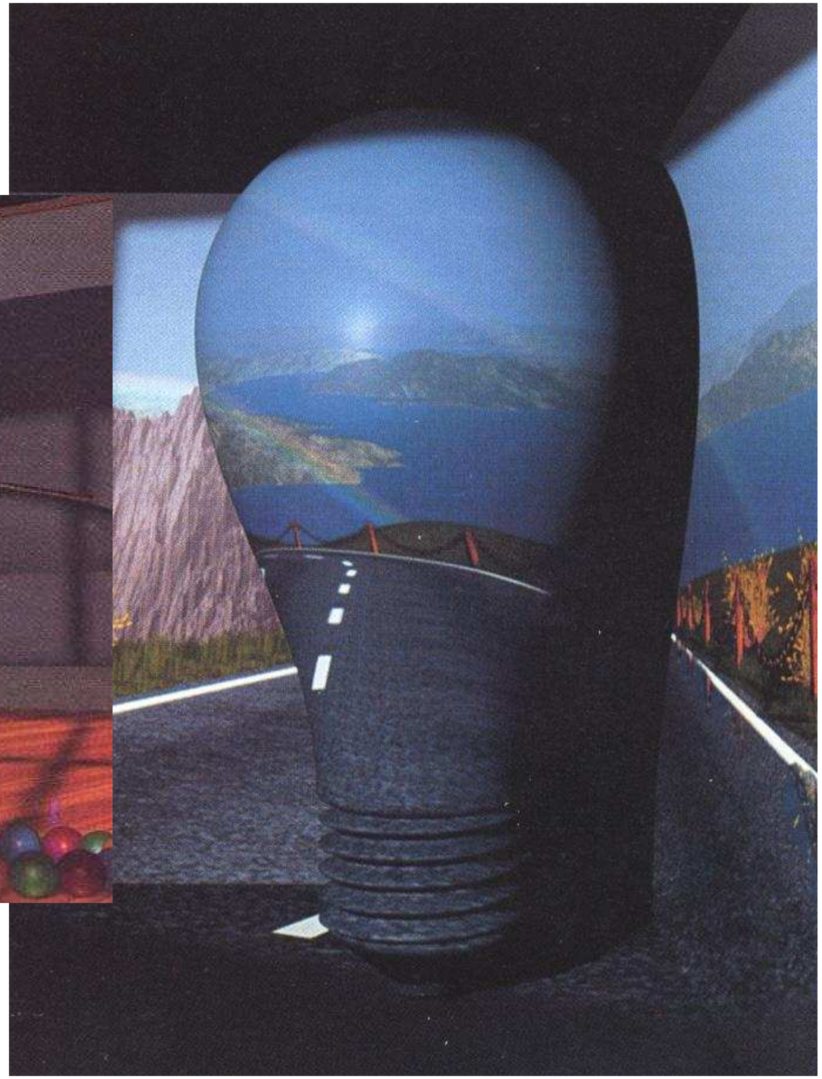
- **Project texture onto object surfaces**
  - Slide projector
- **Parallel or perspective projection**
- **Use photographs as textures**
- **Multiple images**
  - View-dependent texturing



RenderMan Companion

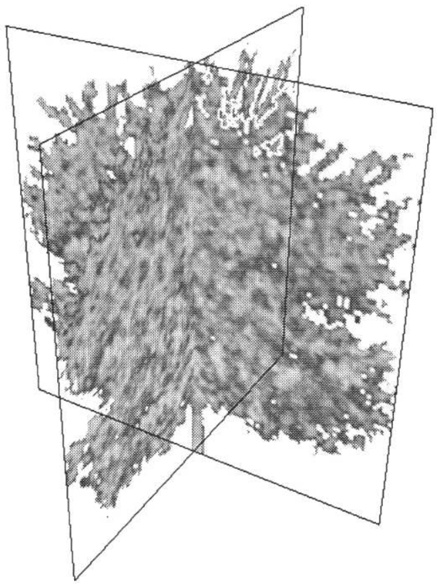
# Projective Texturing: Examples

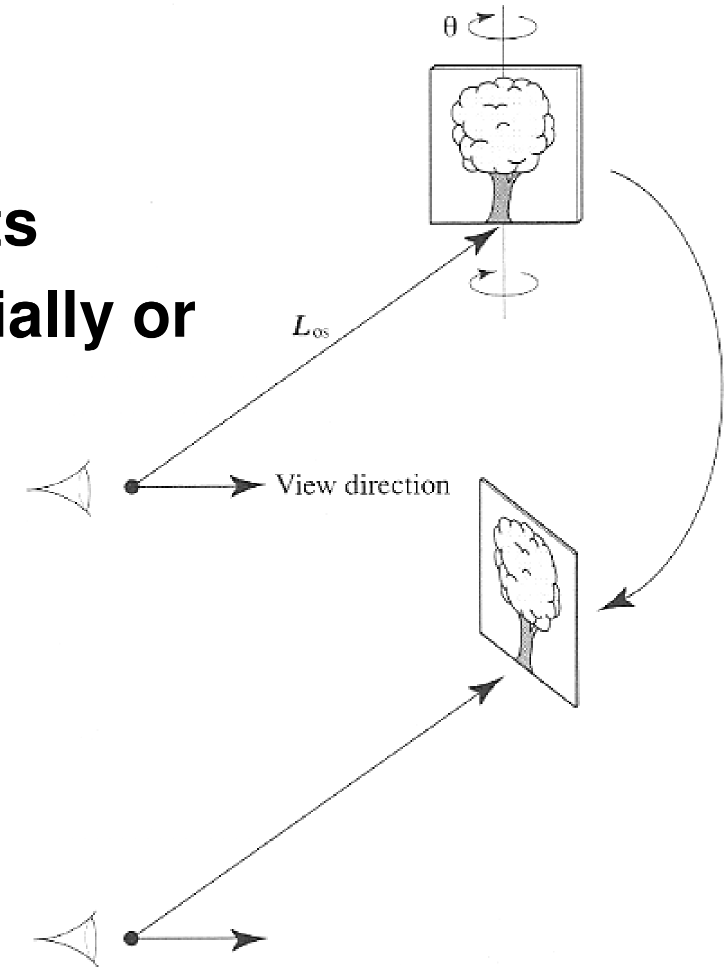


# Billboards

- **Single textured polygons**
  - Often with transparency texture
- **Rotates, always facing viewer**
- **Used for rendering distant objects**
- **Best results if approximately radially or spherically symmetric**





# Procedural Methods

# Texture Maps vs. Procedural Textures

- **Texture maps (photos, simulations, videos, ...)**
  - Simple acquisition
  - Illumination during acquisition
  - Limited resolution, aliasing
  - High memory requirements
  - Mapping difficult
- **Procedural textures**
  - Non-trivial programming
  - Flexibility
  - Parametric control
  - Unlimited resolution, antialiasing possible
  - Low memory requirements
  - Low-cost visual complexity
  - Adapts to arbitrary geometry

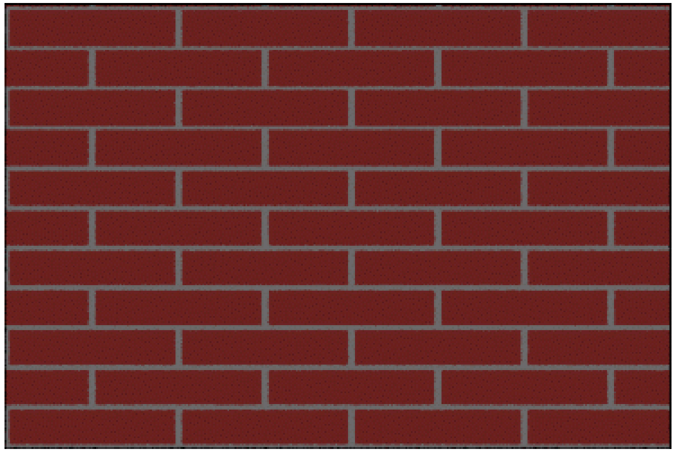
Ken Perlin

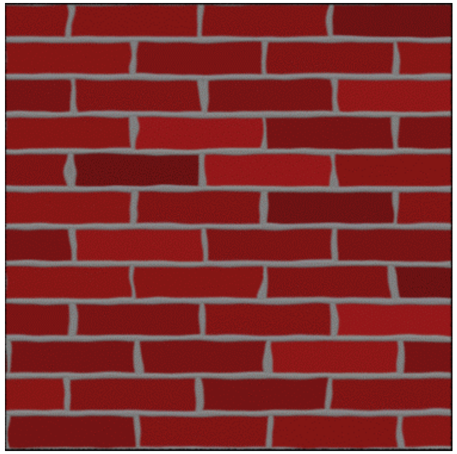
# Procedural Textures

- **Analytic scalar function of world coordinates** $(x,y,z)$
- **Texturing: evaluation of function on object surface**
  - Ray tracing: 3D intersection point with surface
- **Textures of natural objects**
  - Similarity between different patches
    - Repetitiveness, coherence
  - Similarity on different resolution scales
    - Self-similarity
  - But never completely identical
    - Additional disturbances, turbulence, noise
- **Procedural texture function**
  - Mimics statistical properties of natural textures
  - Purely empirical approach
    - Looks convincing, but has nothing to do with material's physics

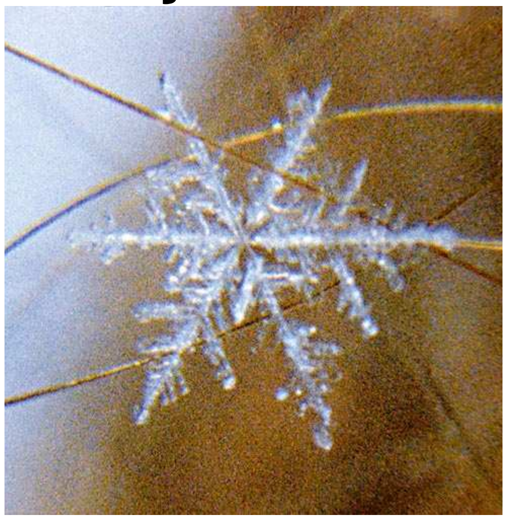
# Texture Examples

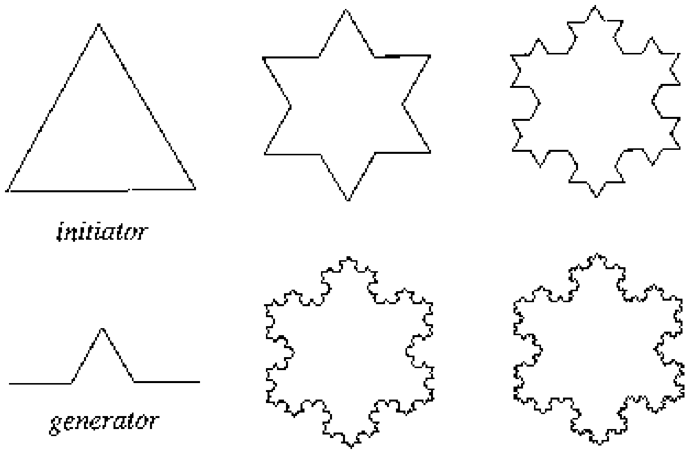
- **Translational similarity**





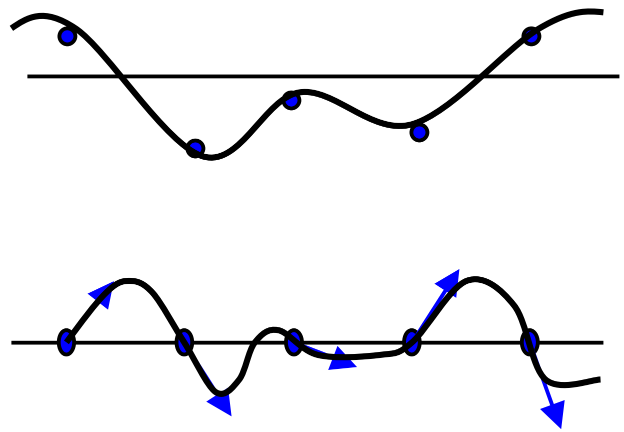
- **Similarity on different scales**





# 3D / Solid Noise: Perlin Noise

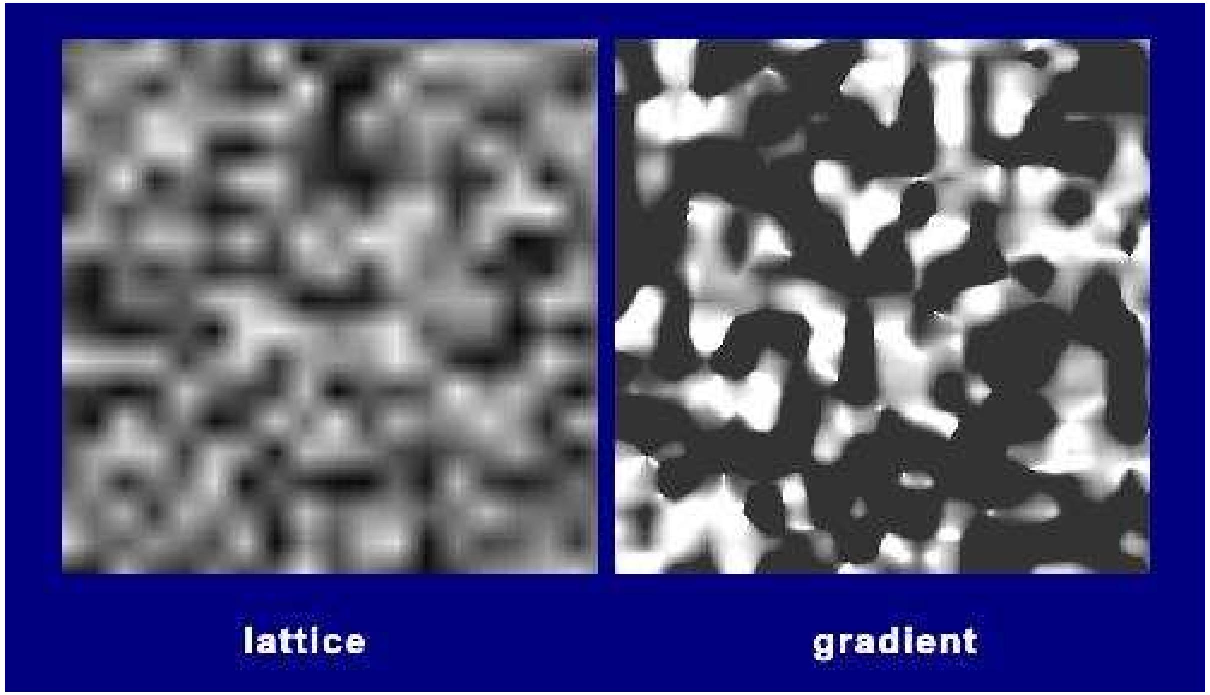
- **Noise(x,y,z)**
  - Statistical invariance under rotation
  - Statistical invariance under translation
  - Narrow bandpass limit in frequency
- **Integer lattice (i,j,k)**
  - Random number at each lattice point (i,j,k)
    - Look-up table or hashing function
  - Gradient lattice noise
    - Random gradient vectors



- **Evaluation at (x,y,z)**
  - Tri-linear interpolation
  - Cubic interpolation (Hermite spline → later)
- **Unlimited domain**
  - Lattice replicated to fill entire space
- **Fixed fundamental frequency of ~1 Hz over lattice**
- **Smooth interpolation of interim values**

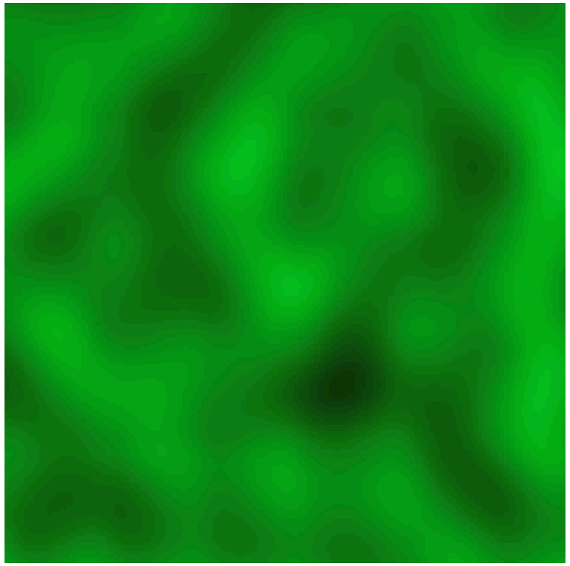
# Gradient vs. Value Noise

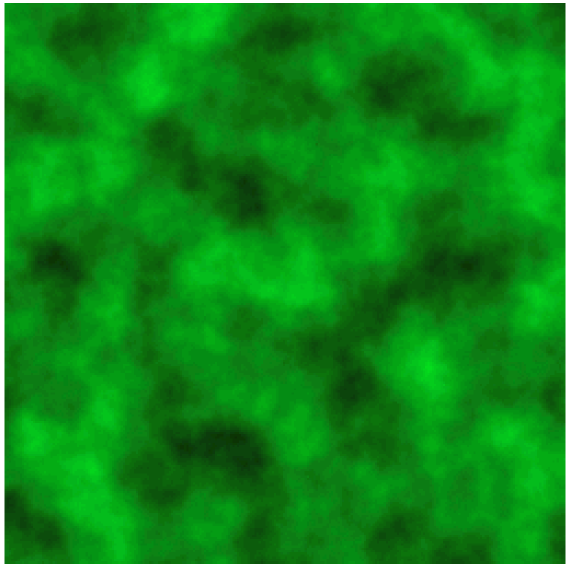
- **Gradient noise better than value noise**
  - less regularity artifacts
  - more high frequencies in noise spectrum
  - even tri-linear interpolation produces good results



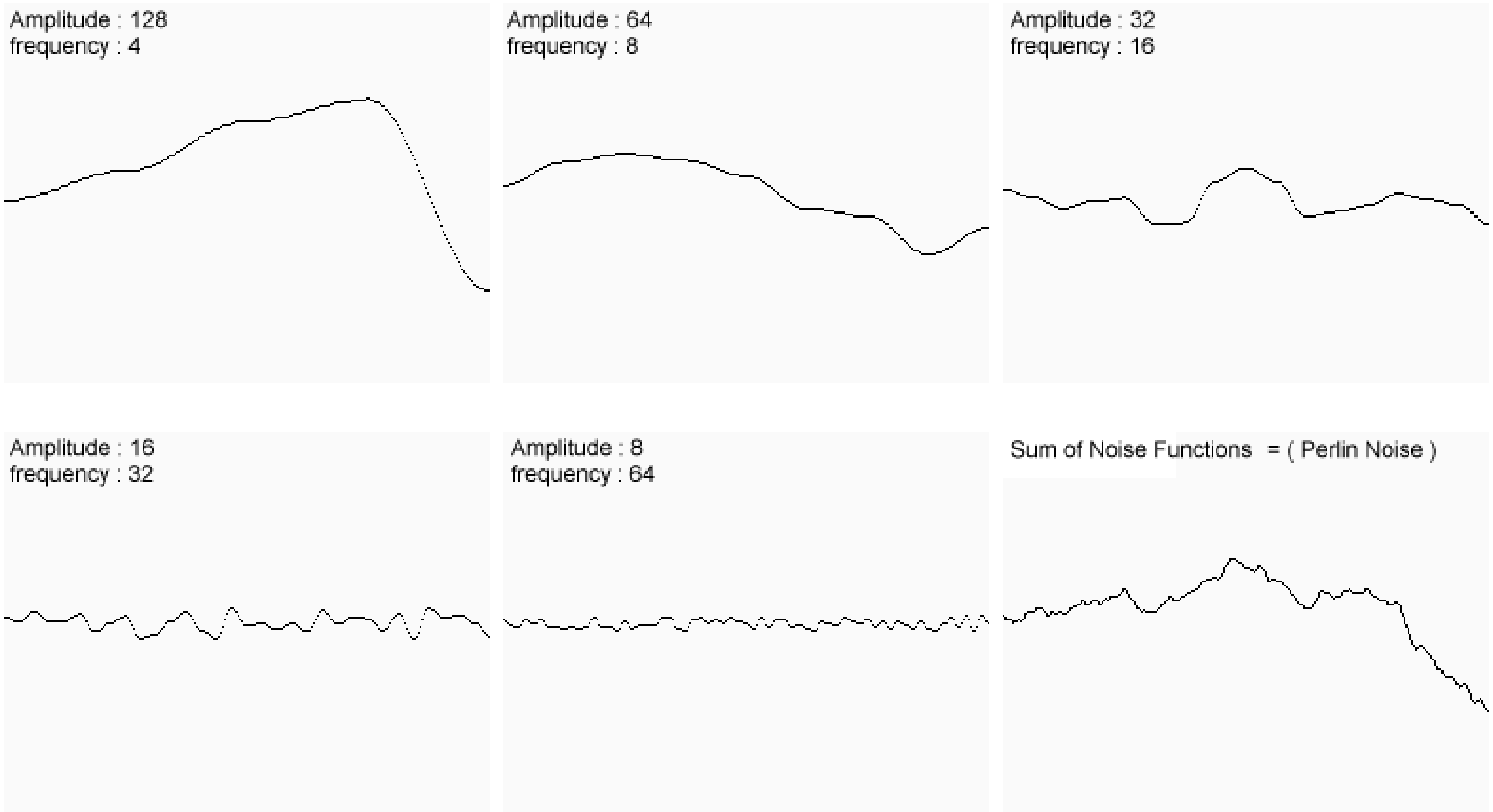
# Turbulence Function

- **Noise function**
  - “White” frequency spectrum
- **Natural textures**
  - Decreasing power spectrum towards high frequencies
- **Turbulence from noise**
  - Turbulence(x) = $\sum^{k}_{i=0} \mathrm{abs}(\mathrm{noise}(2^i x) / 2^i)$
  - Summation truncation
    - $1/2^{k+1}$ < size of one pixel (band limit)
  - 1. Term: noise(x)
  - 2. Term: noise(2x)/2
  - …
  - Power spectrum: 1/f
  - (Brownian motion: $1/f^2$)

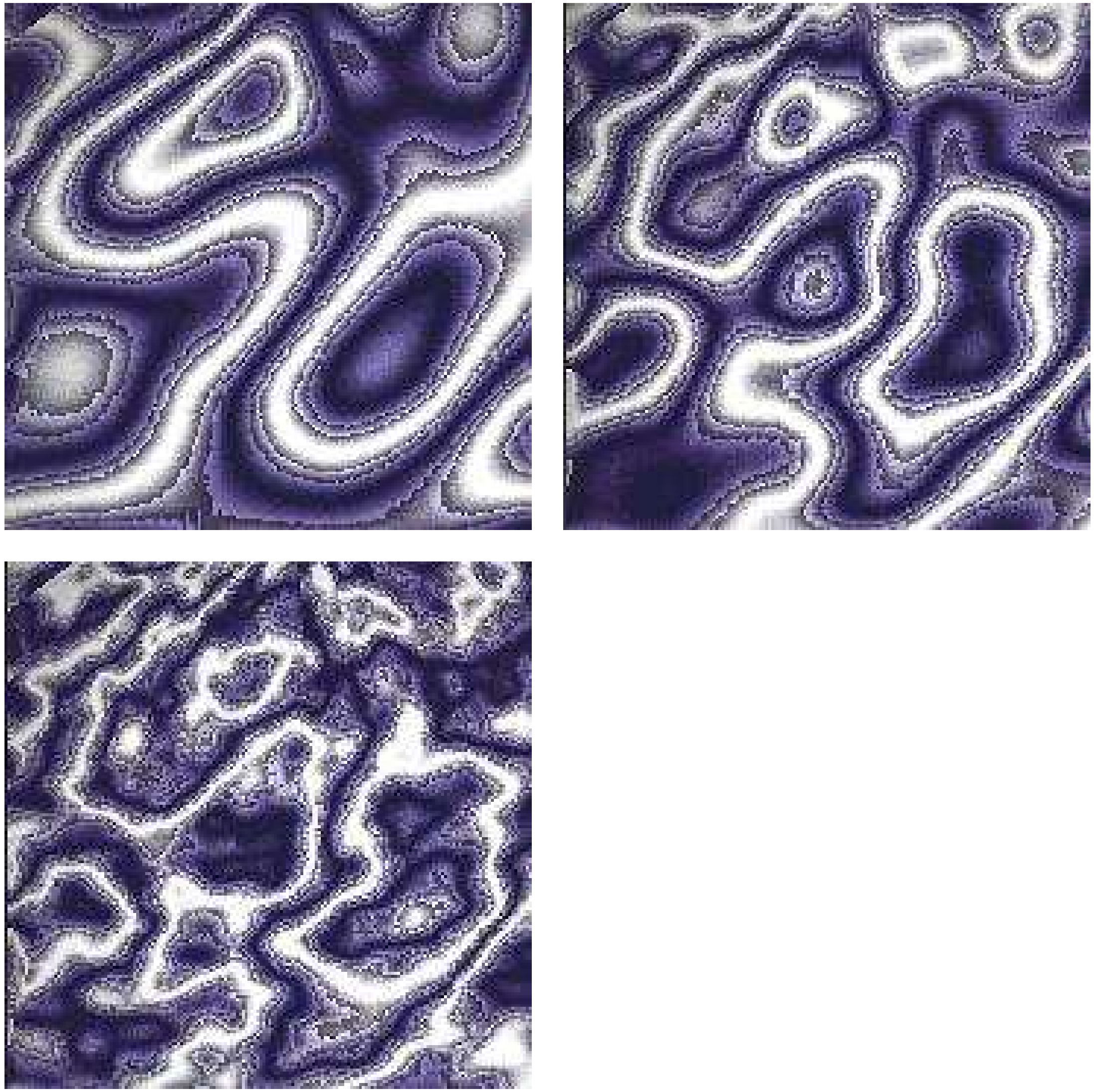




# Synthesis of Turbulence (1D)



# Synthesis of Turbulence (2D)



ERROR: stackunderflow
OFFENDING COMMAND: ~

STACK: