



Lecture 8:

Texturing 2

Contents

1. Texturing
2. Procedural Textures
3. Fractal Landscapes



To apply textures we need 2D coordinates on surfaces

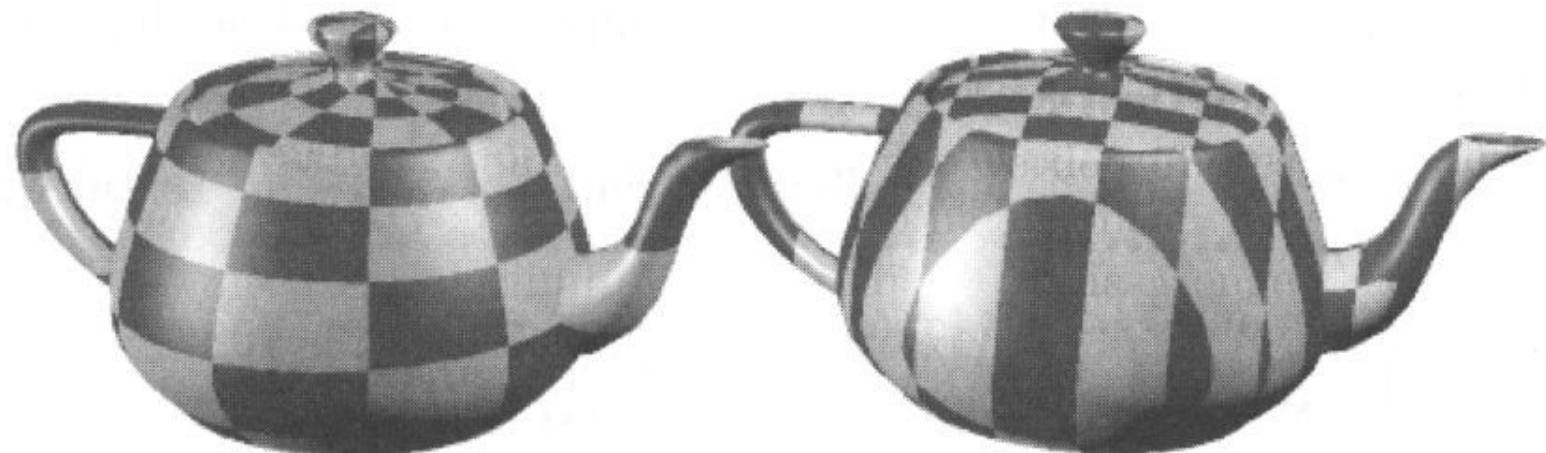
- → Parameterization!

Some objects have a natural parameterization

- **Sphere:** spherical coordinates $(\theta, \varphi) = (2\pi \cdot u, \pi \cdot v)$
- **Cylinder:** cylindrical coordinates $(\theta, z) = (2\pi \cdot u, H \cdot v)$
- Parametric surfaces (such as B-spline or Bezier surfaces → later)

Parameterization less obvious for

- Polygons, implicit surfaces, ...





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_0, y_0, z_0)

Apply viewing projection $(x_0, y_0, z_0) \rightarrow (x, y)$

Yields texture transformation (warping) $(u, v) \rightarrow (x, y)$

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

- In homogeneous coordinates

$$(x, y) = \left(\frac{x'}{w}, \frac{y'}{w} \right)$$

$$(u, v) = \left(\frac{u'}{q}, \frac{v'}{q} \right)$$

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

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



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

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

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

- $(u, v) = \left(\frac{u'}{q}, \frac{v'}{q} \right)$ $(x', y', w) = (x, y, 1)$

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

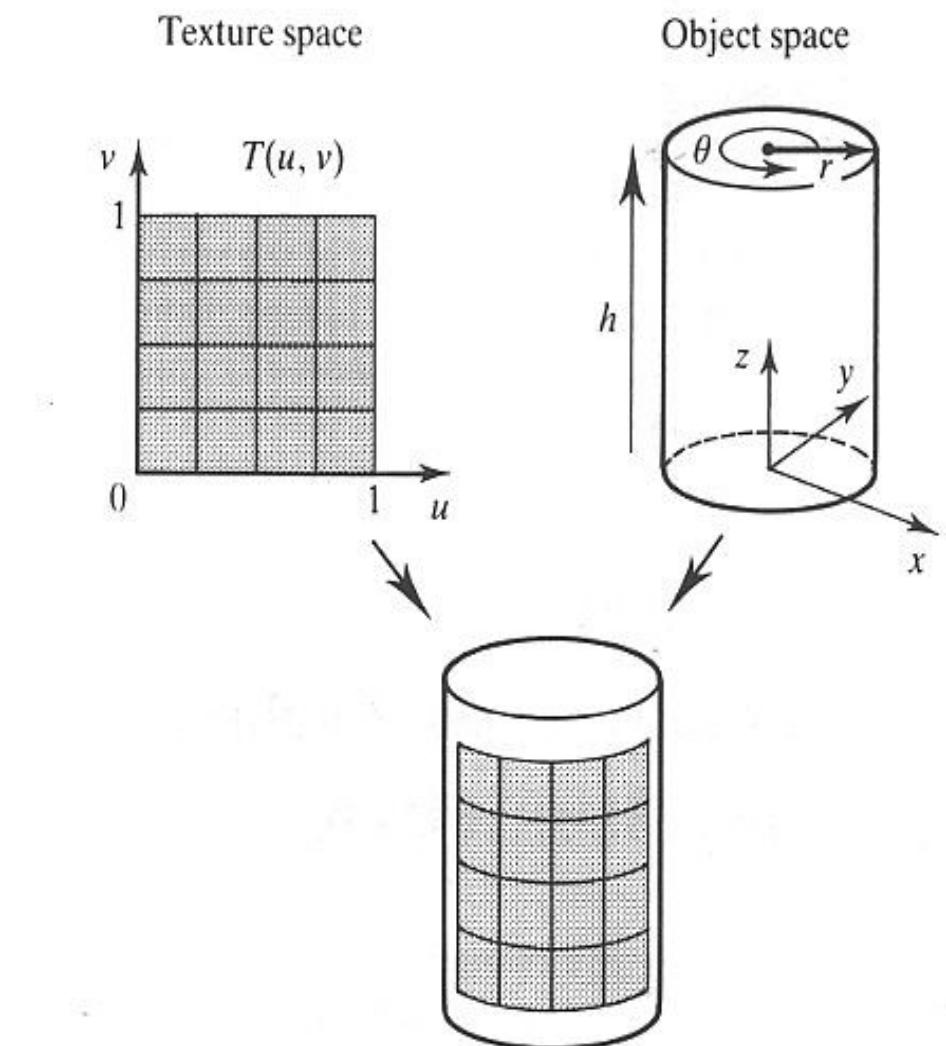


Transformation from texture space to the cylinder parametric representation can be written as:

- $(\theta, z) = (2\pi \cdot u, H \cdot v)$
- where H is the height of the cylinder.

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

- $x_0 = r \cos \theta$
- $y_0 = r \sin \theta$
- $z_0 = h$

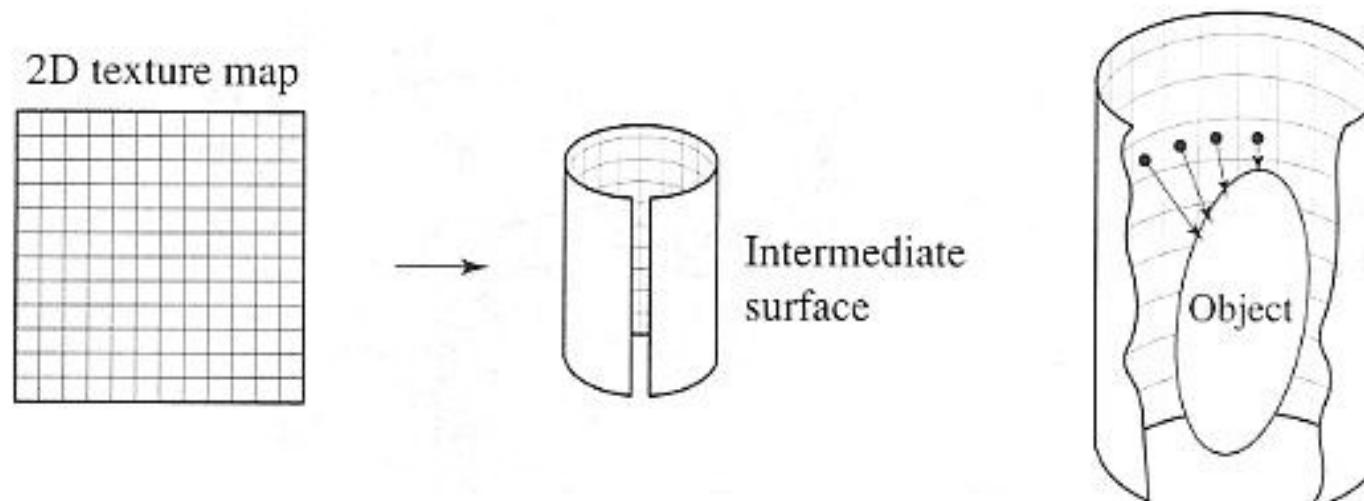




Inverse Mapping for arbitrary 3D surfaces too complex

Approximation technique is used:

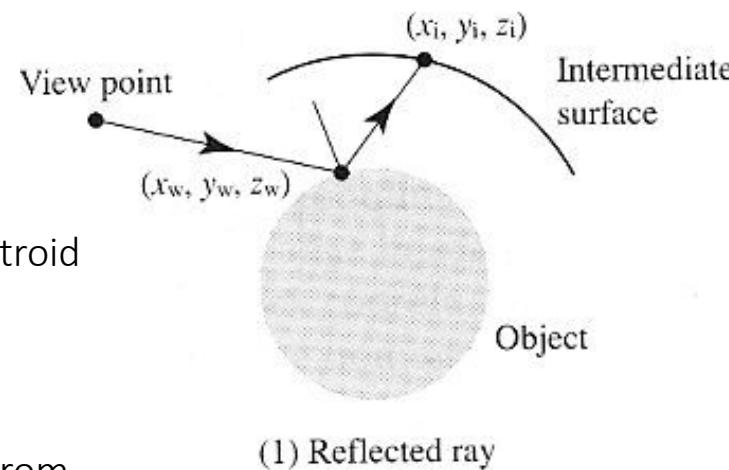
- **S – mapping:** Mapping from 2D texture space to a simple 3D intermediate surface, which is a reasonable approximation of the destination surface (e.g., cylinder, sphere)
- **O – mapping:** Mapping from the intermediate surface to the destination object surface



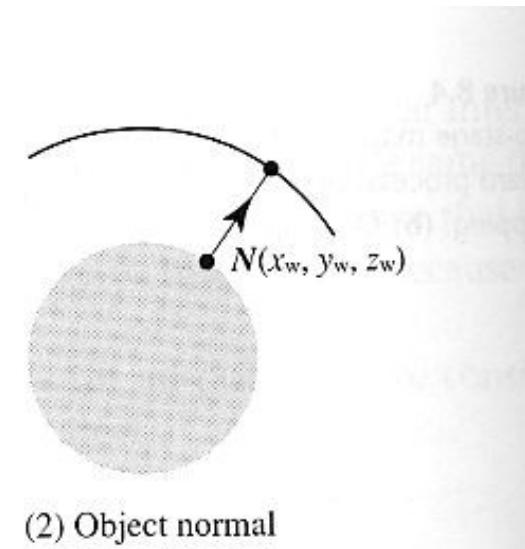


Determine point on intermediate surface through

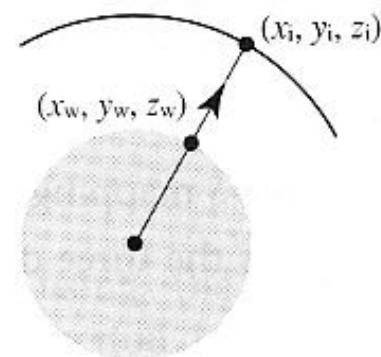
- Reflected view ray
 - Reflection or environment mapping
- Normal mapping
- Line through object centroid
- Shrinkwrapping
 - Forward mapping
 - Normal mapping from intermediate surface



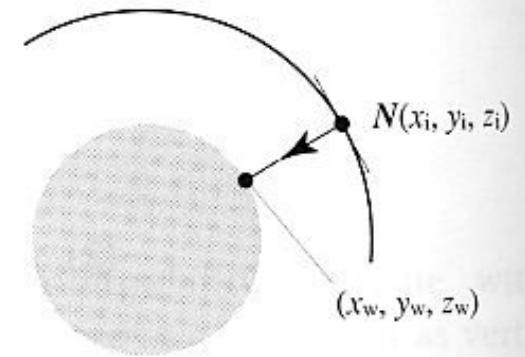
(1) Reflected ray



(2) Object normal



(3) Object centroid

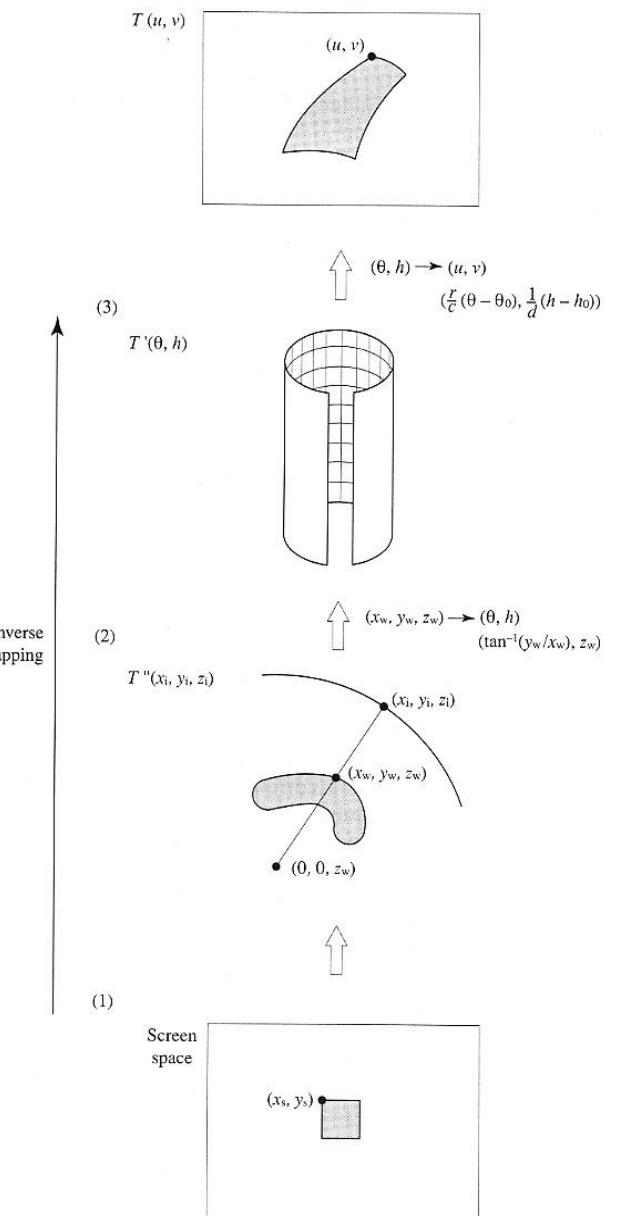


(4) Intermediate surface normal



Inverse two-stage mapping

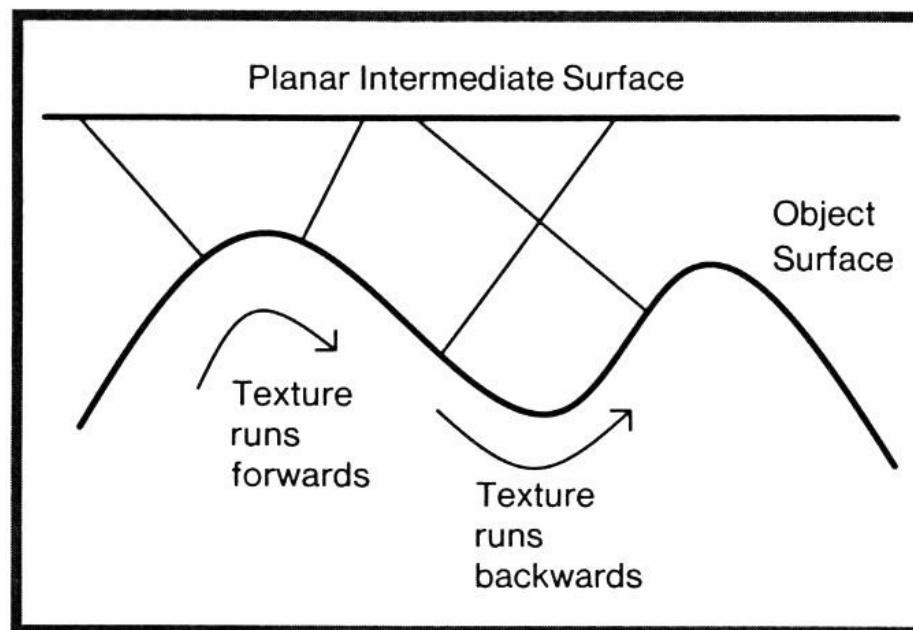
- Map 4 screen pixels to object surface
- O - mapping
 - Shrinkwrapping: Intersection of line from cylinder axis through object point
- S - mapping
 - Inverse-map cylinder surface to texture map





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.

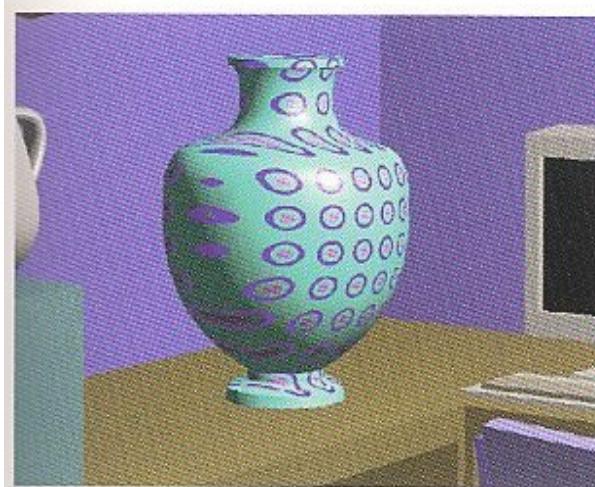


Two-Stage Mapping

Example:

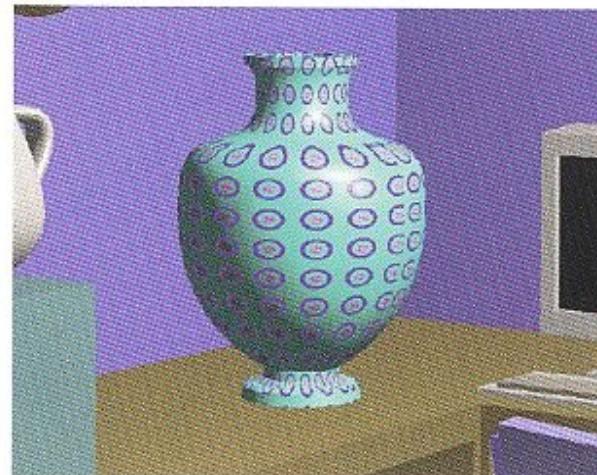
- different intermediate surfaces:

Plane



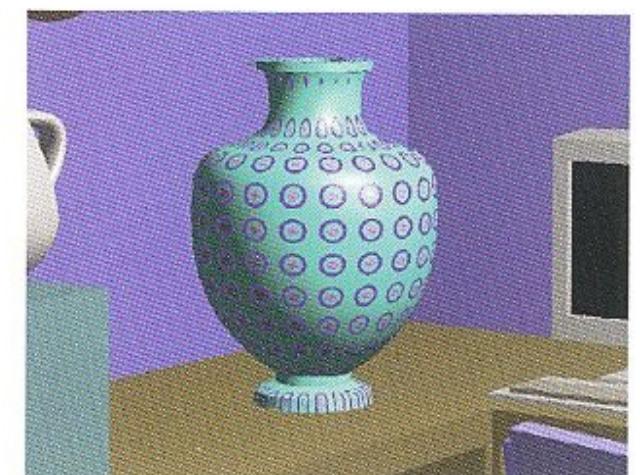
Strong distortion where
object surface normal \perp
plane normal

Cylinder



Reasonably uniform mapping
(symmetry!)

Sphere



Problems with concave
regions



Project texture onto object surfaces

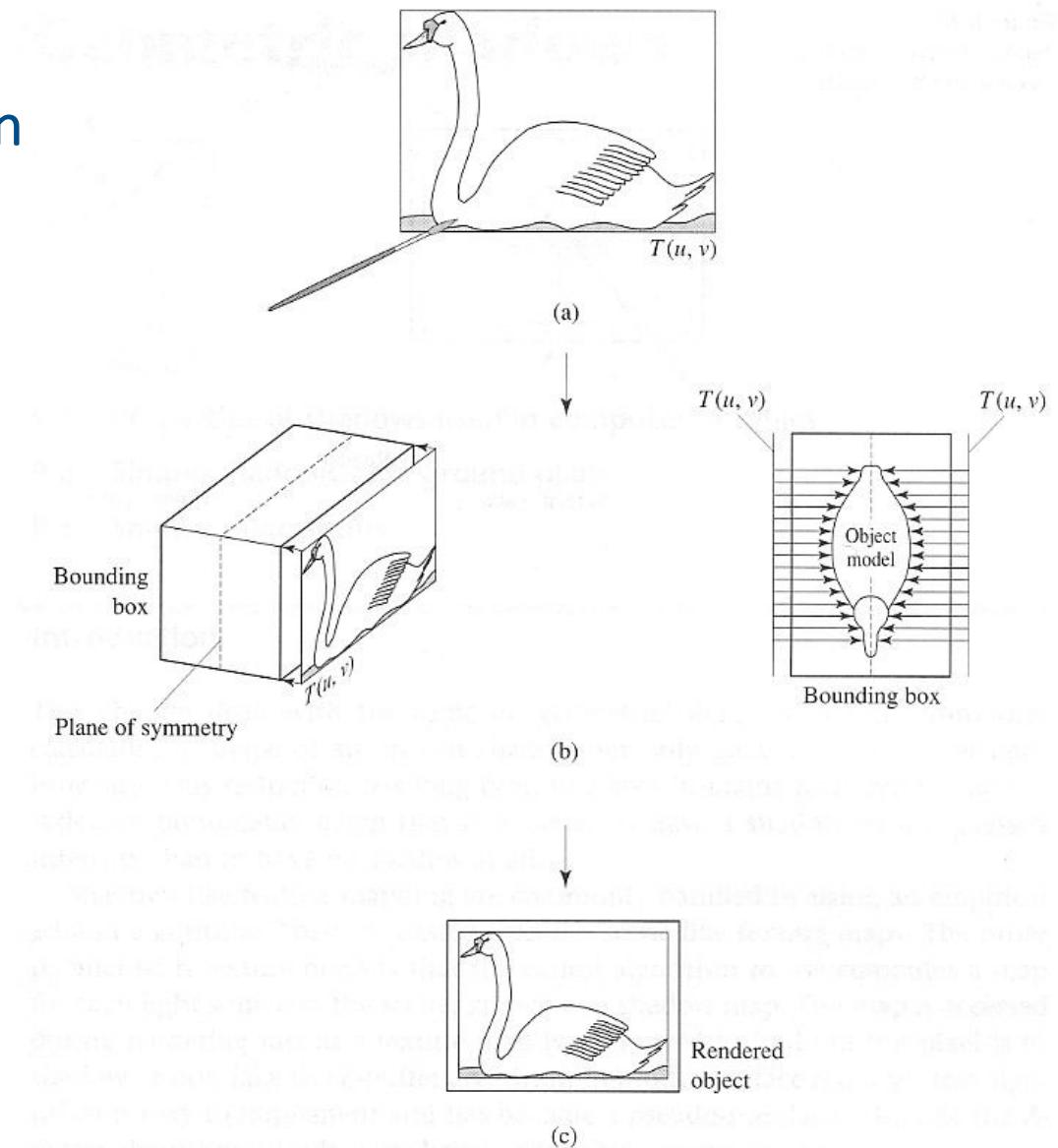
- Slide projector

Parallel or perspective projection

Use photographs as textures

Multiple images

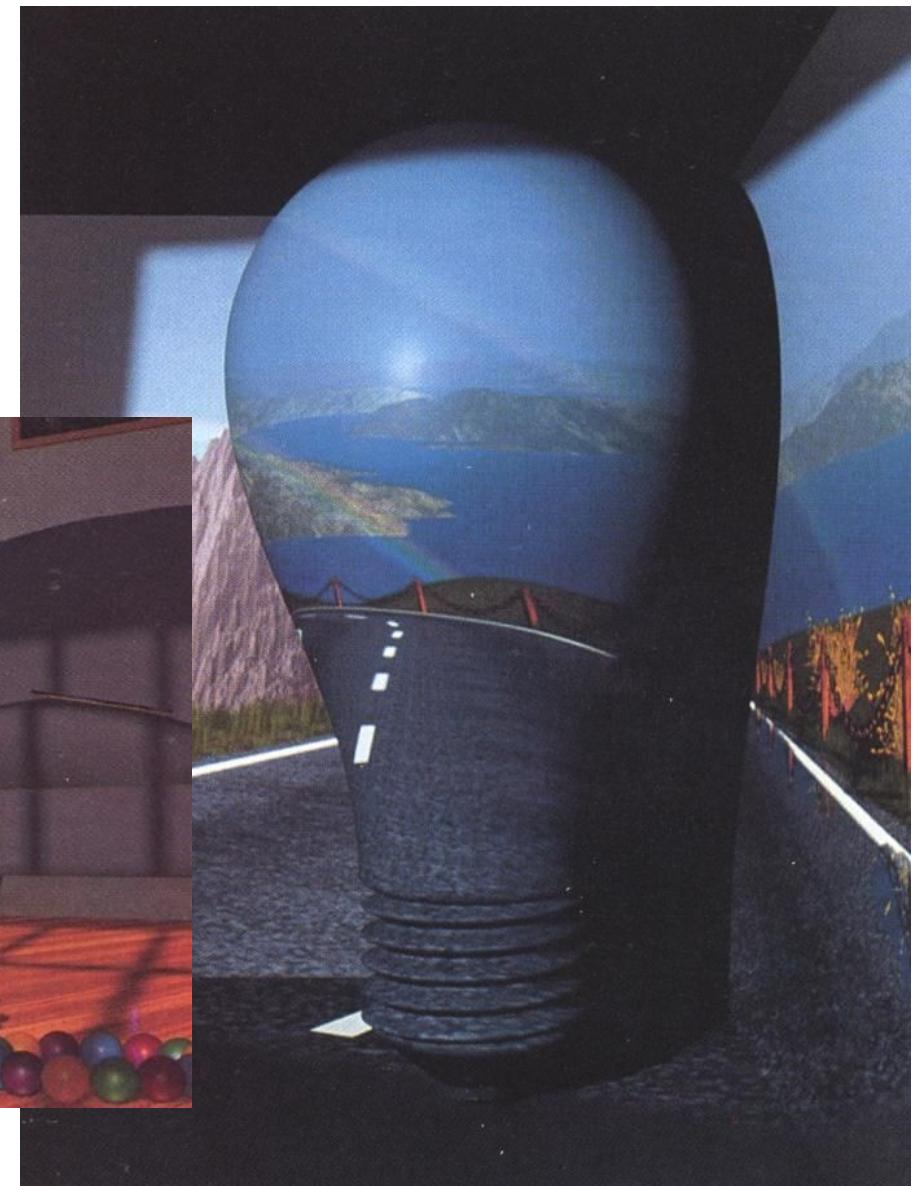
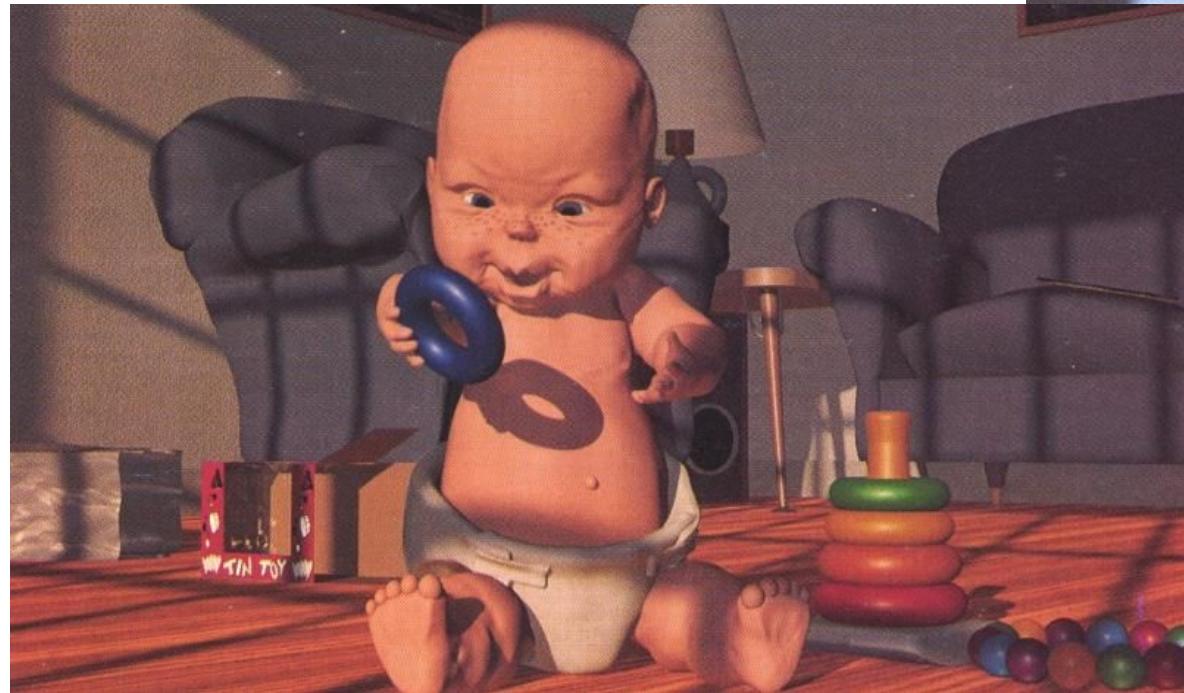
- View-dependent texturing





Projective Texturing

Examples:





Also called Environment Mapping

Mirror reflections

- Surface curvature: beam tracing
- Map filtering

Reflection map parameterization

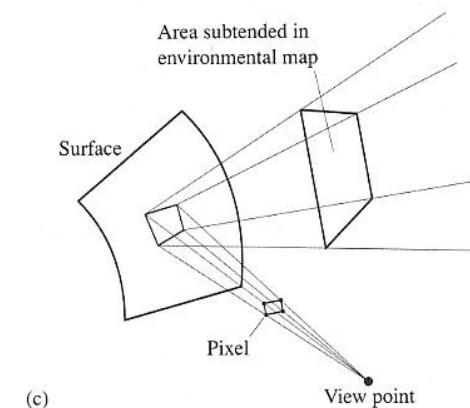
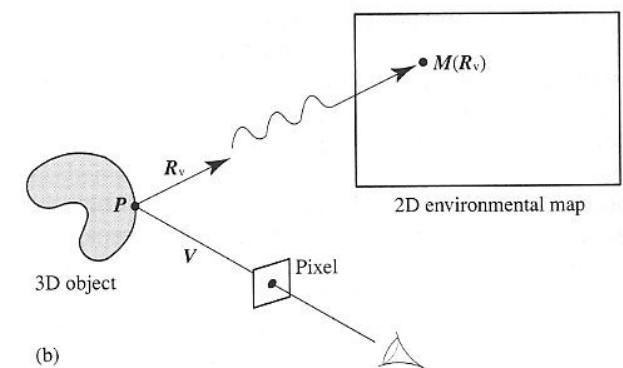
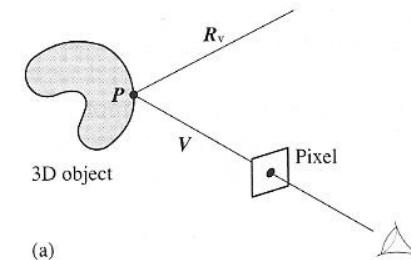
- Intermediate surface in 2-stage mapping

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





Generating spherical maps (original 1982 / 83)

- *i.e.* photo of a reflecting sphere (gazing ball)

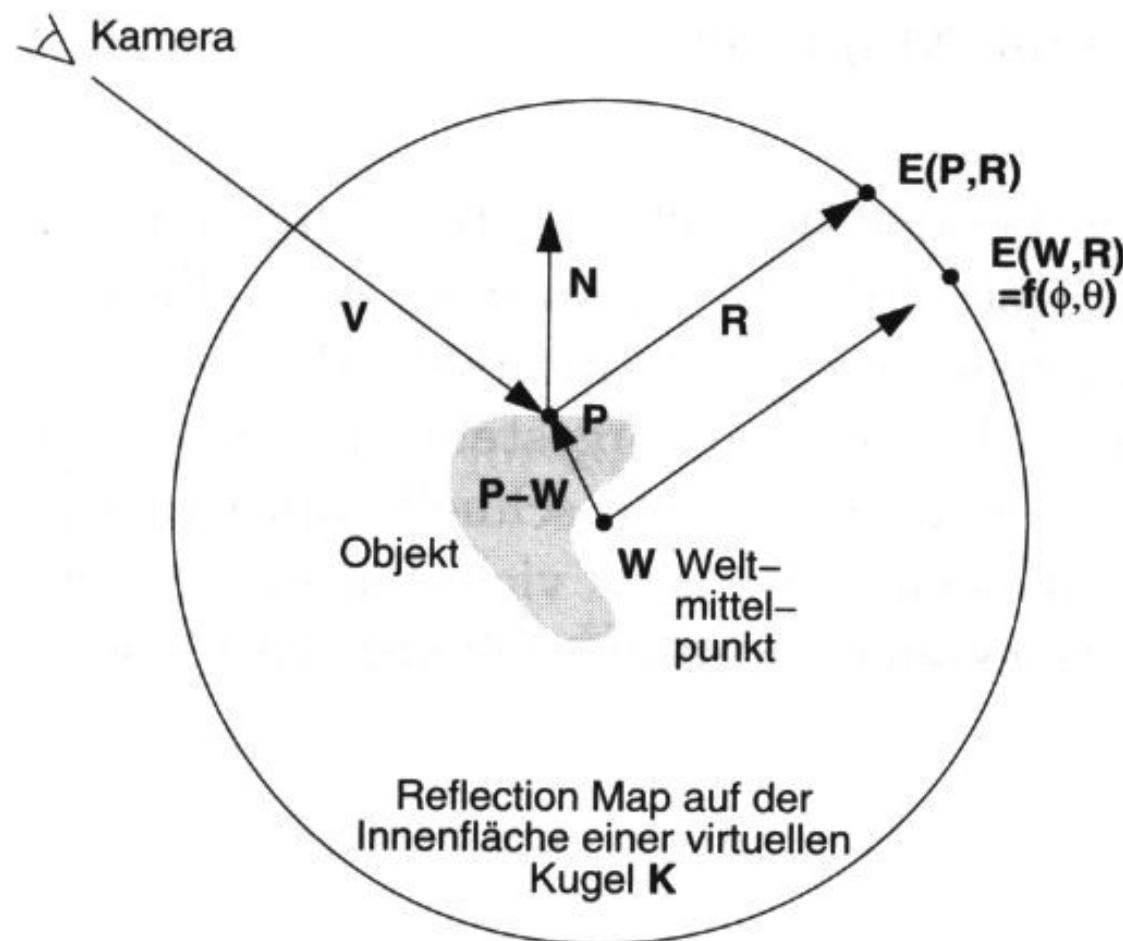


Peter Chou



Spherical parameterization

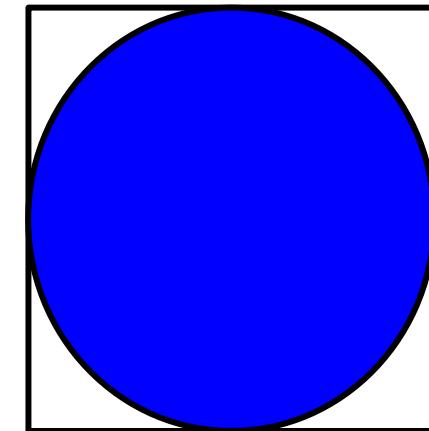
O-mapping using reflected view ray intersection





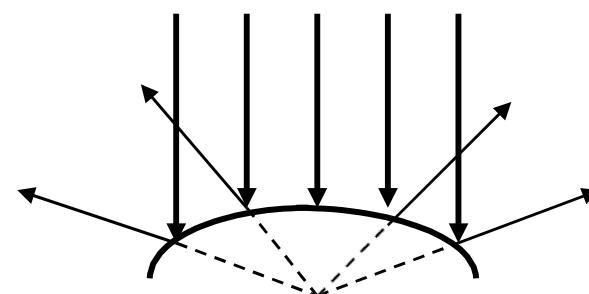
Spherical mapping

- 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

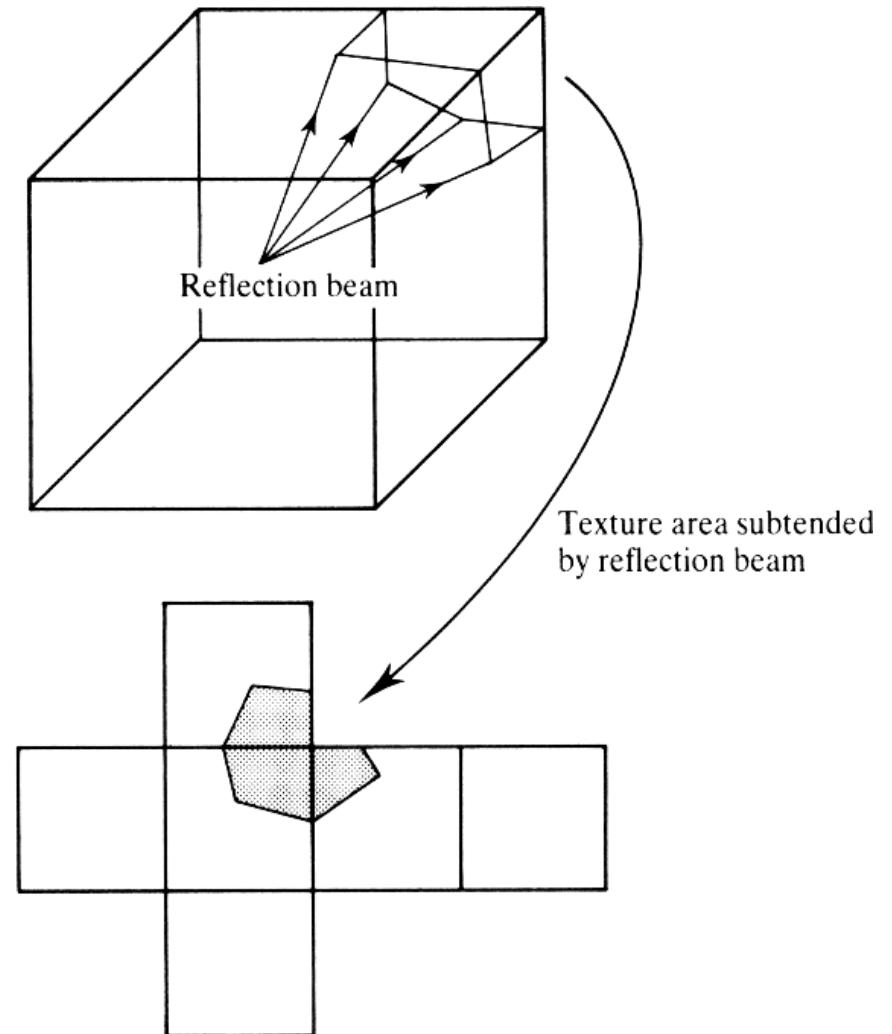
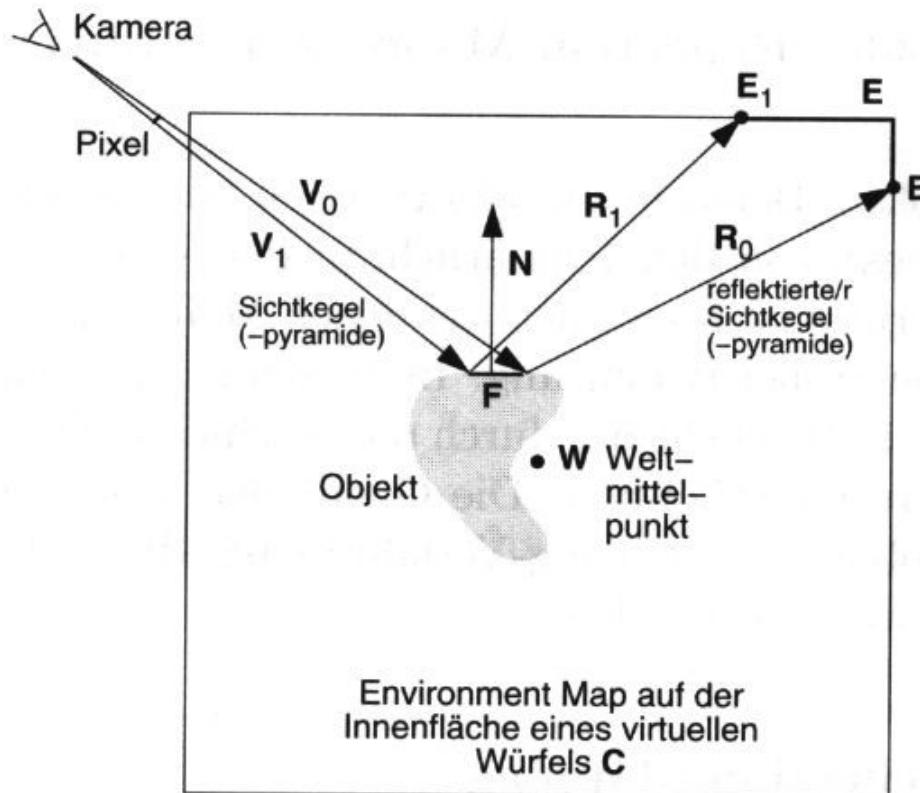
- Subdivide in 2 images (facing and back facing side)
- Less bias on the edge
- Arbitrarily reusable
- Supported by OpenGL extensions





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





Example:



Terminator II motion picture



Example 2: Reflection mapping with Phong reflection

- Two maps: diffuse & specular
- Diffuse: index by surface normal
- Specular: indexed by reflected view vector



RenderMan
Companion



Differences ?



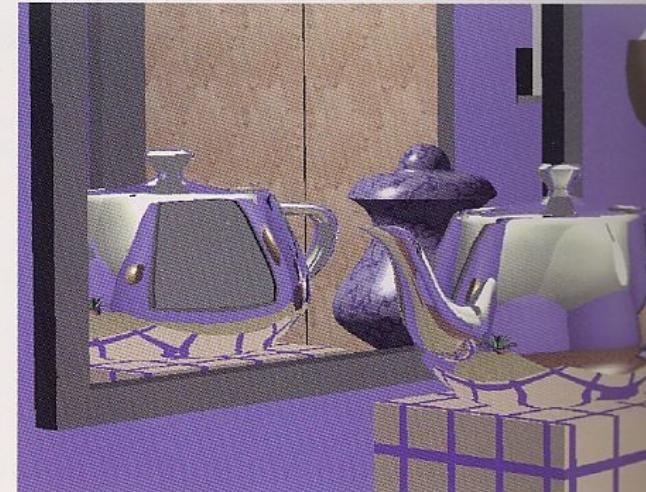


Recursive Ray Tracing

Raytracing
depth 0



Raytracing
depth 1



Raytracing
depth 2

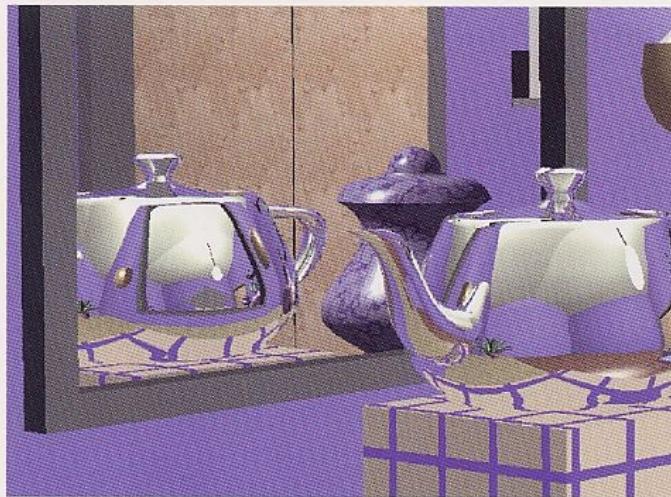


Figure 18.11

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.



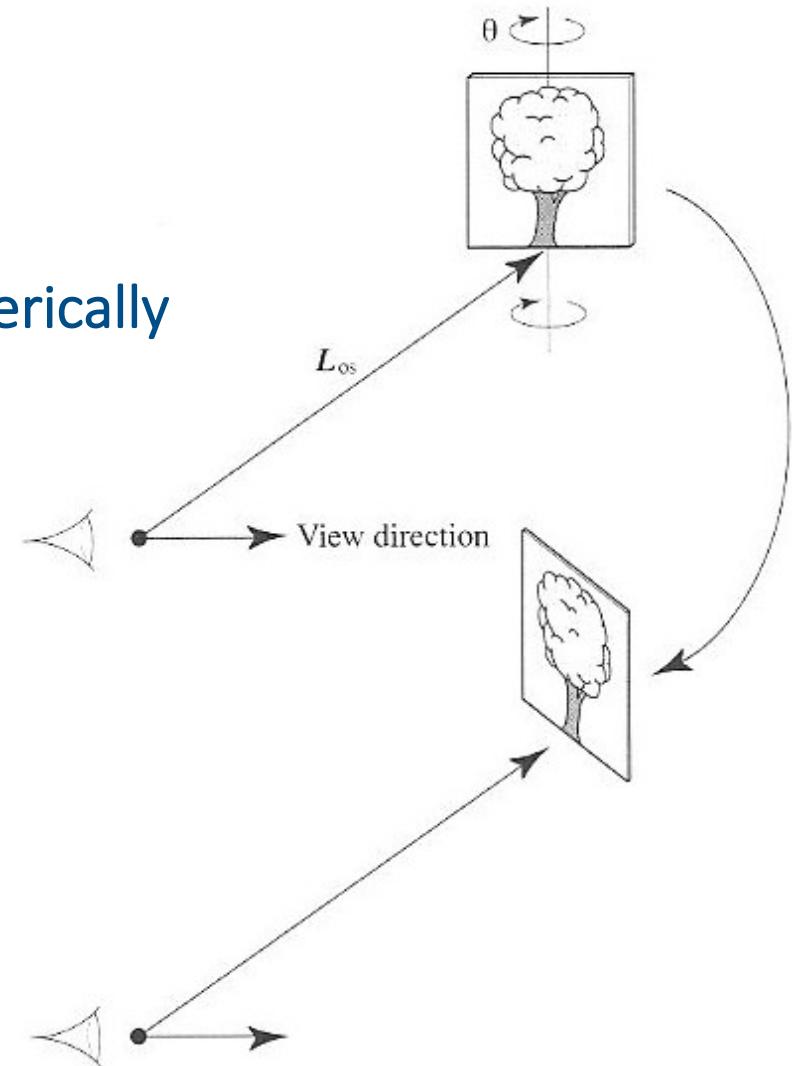
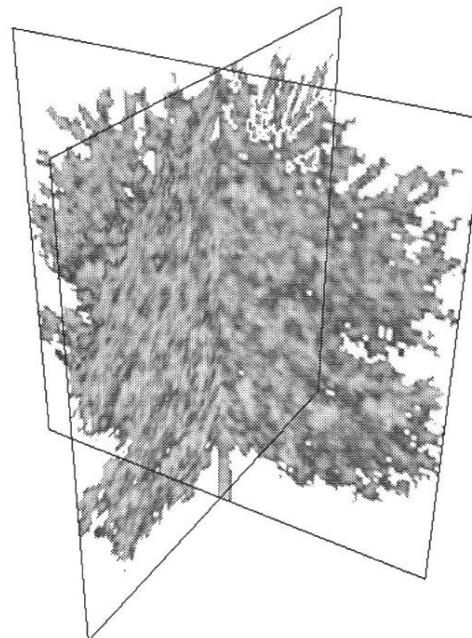
Single textured polygons

- Often with transparency texture

Rotates, always facing viewer

Used for rendering distant objects

Best results if approximately radially or spherically symmetric

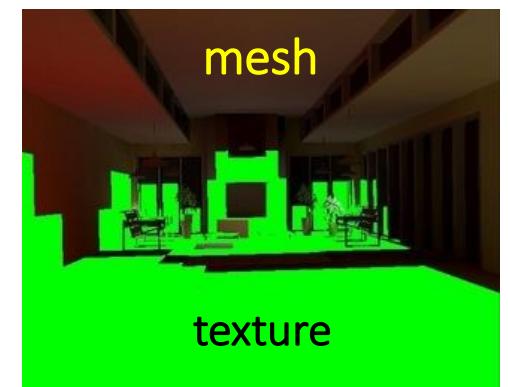
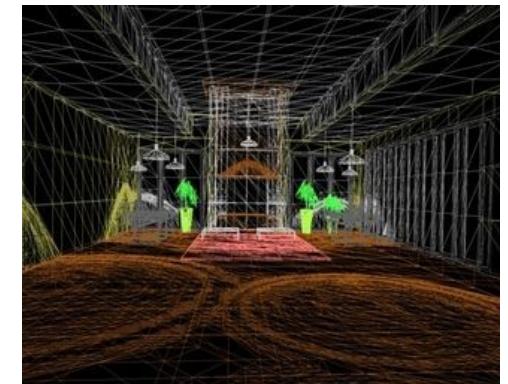




Light maps (*i.e.* in Quake)

- 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.*

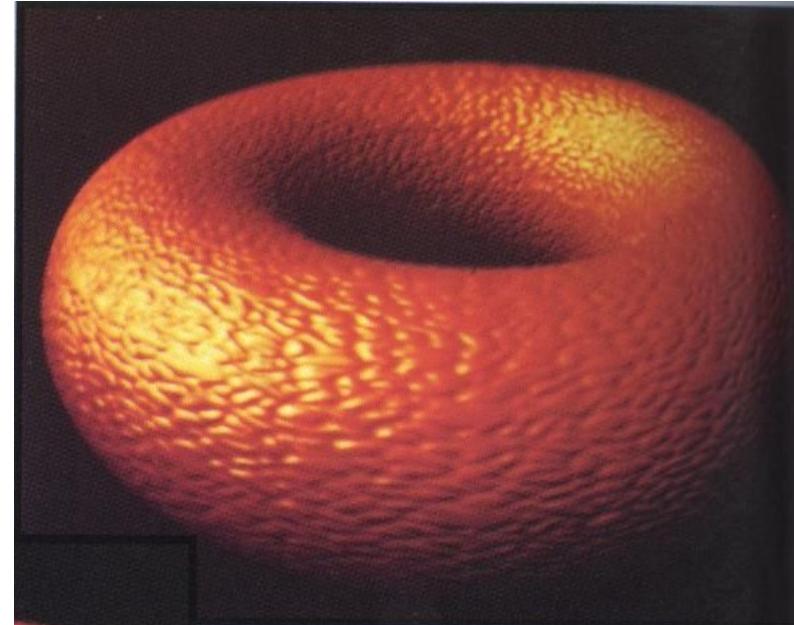
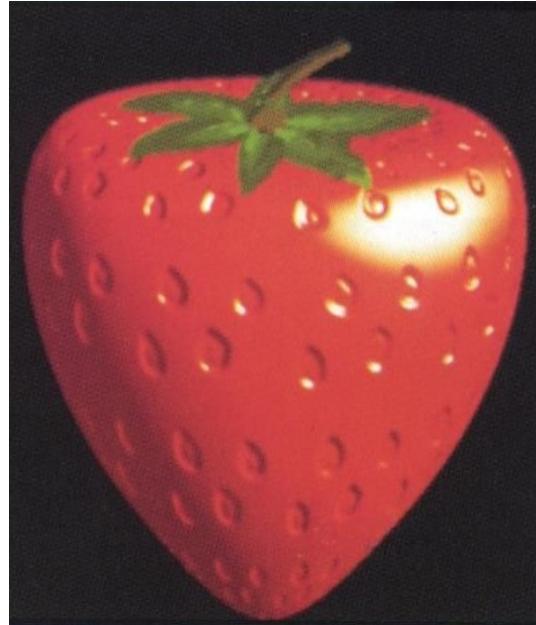
$$\begin{array}{ccc}
 \text{Reflectance} & \times & \text{Irradiance} \\
 \text{Radiosity}
 \end{array}$$





Modulation of the normal vector

- Surface normals changed only
 - Influences shading only
 - No self-shadowing, contour is **not** altered



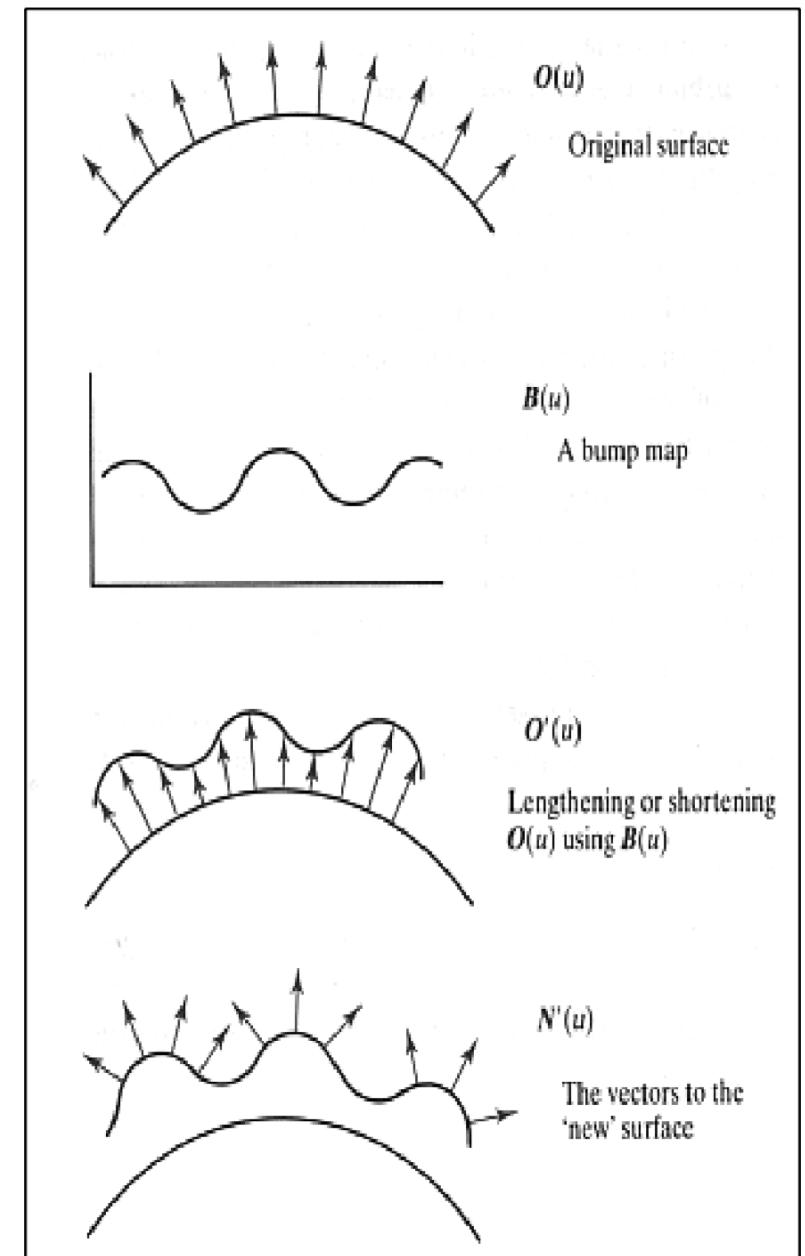
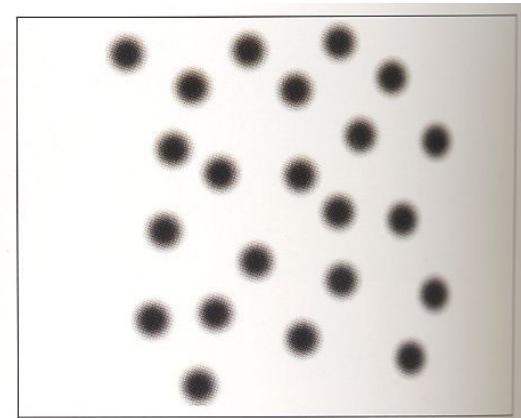
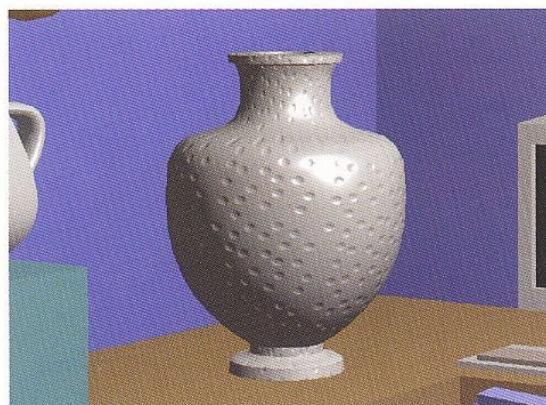


Original surface $O(u, v)$

- Surface normals known

Bump map $B(u, v) \in R$

- 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 $N'(u, v)$





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

Now differentiating this equation gives:

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

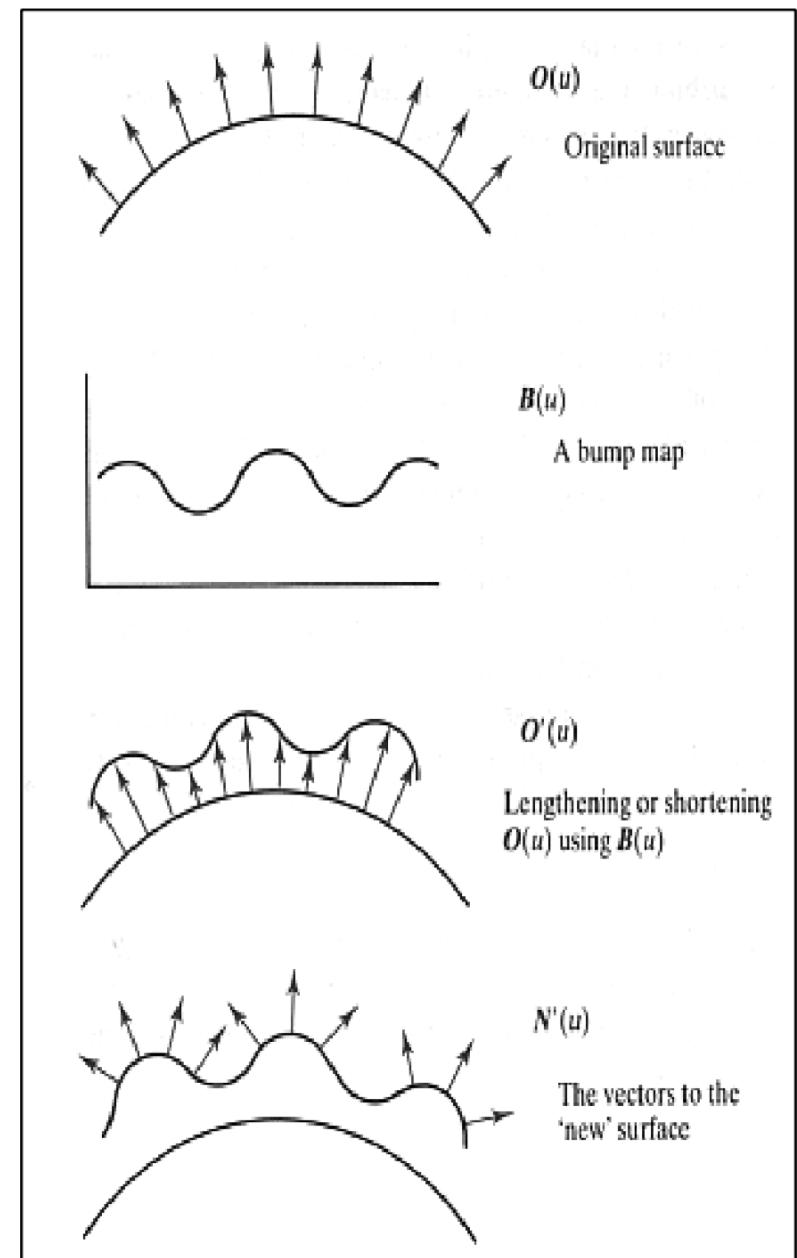
$$\mathbf{O}'_v = \mathbf{O}_v + B_v \frac{\mathbf{N}}{|\mathbf{N}|} + B \left(\frac{\mathbf{N}}{|\mathbf{N}|} \right)_v$$

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

$$\begin{aligned} \mathbf{N}'(u, v) &= \mathbf{O}_u \times \mathbf{O}_v + B_u \left(\frac{\mathbf{N}}{|\mathbf{N}|} \times \mathbf{O}_v \right) + B_v \left(\mathbf{O}_u \times \frac{\mathbf{N}}{|\mathbf{N}|} \right) \\ &\quad + B_u B_v \left(\frac{\mathbf{N} \times \mathbf{N}}{|\mathbf{N}|^2} \right) \end{aligned}$$

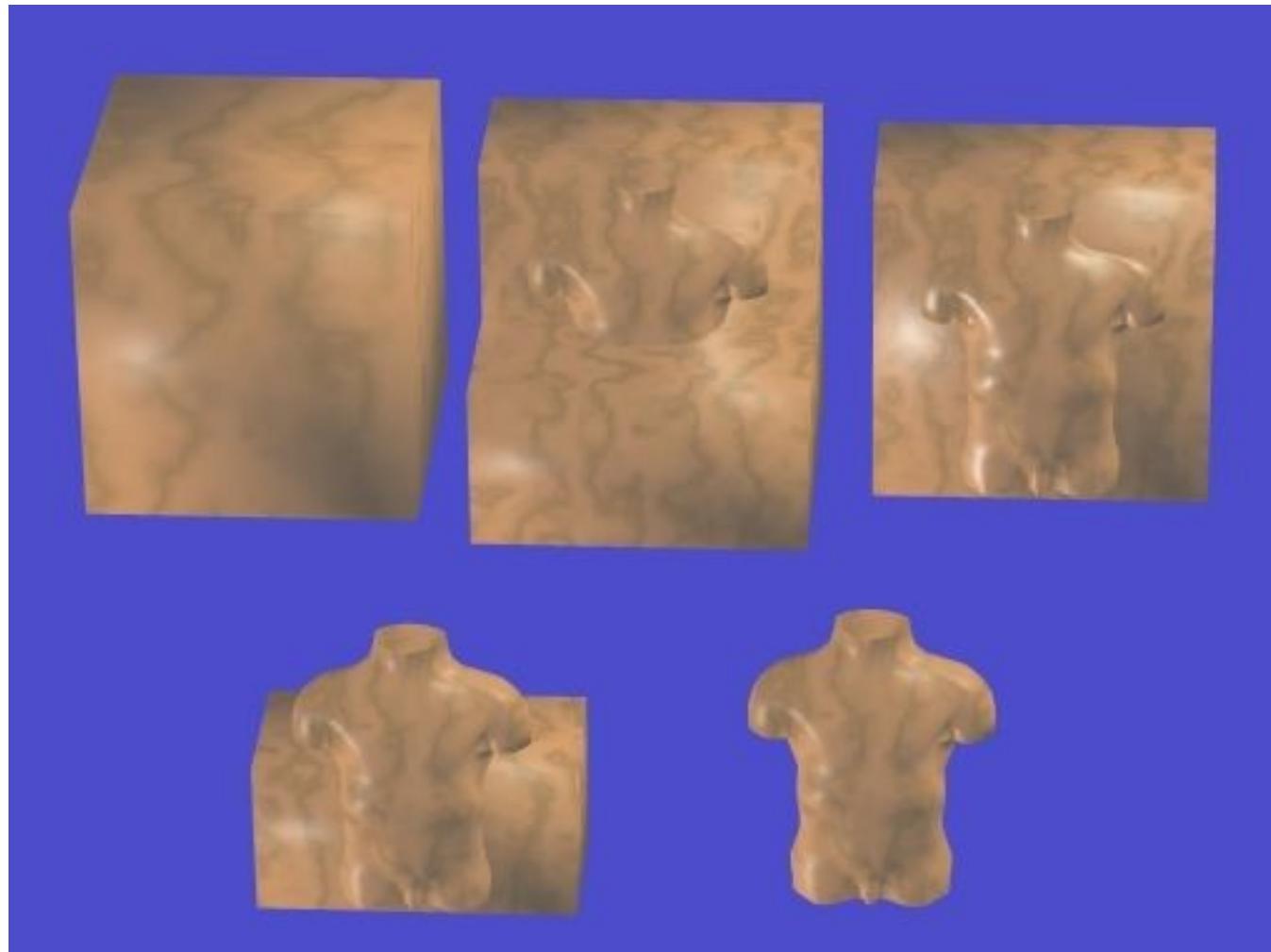
The first term is the normal to the surface and the last term is zero, giving:

$$\mathbf{D} = B_u (\mathbf{N} \times \mathbf{O}_v) - B_v (\mathbf{N} \times \mathbf{O}_u)$$





Carving object shape out of material block





Solid 3D textures (wood, marble)

Bump map (middle)

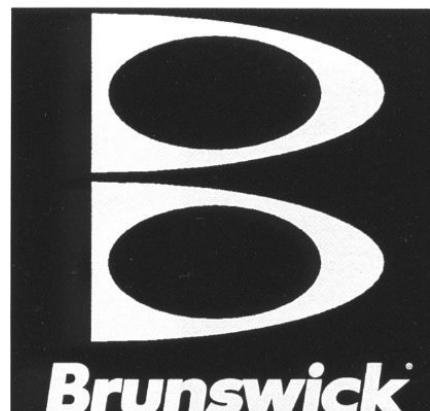
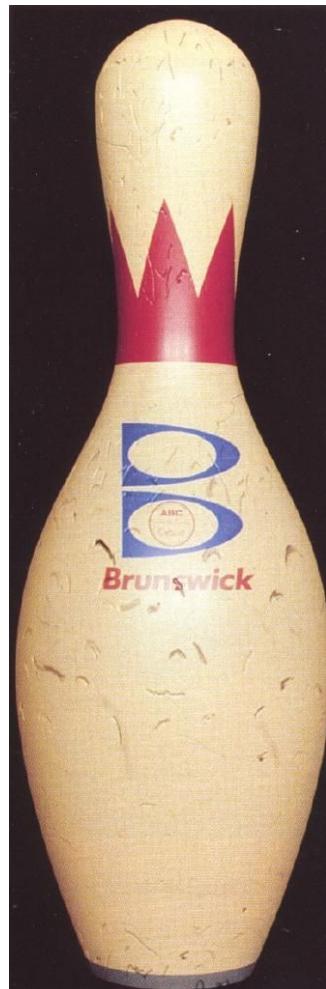


RenderMan Companion

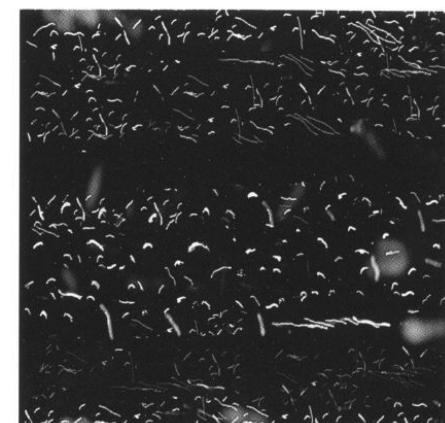


Complex optical effects

- Combination of multiple textures



RenderMan Companion



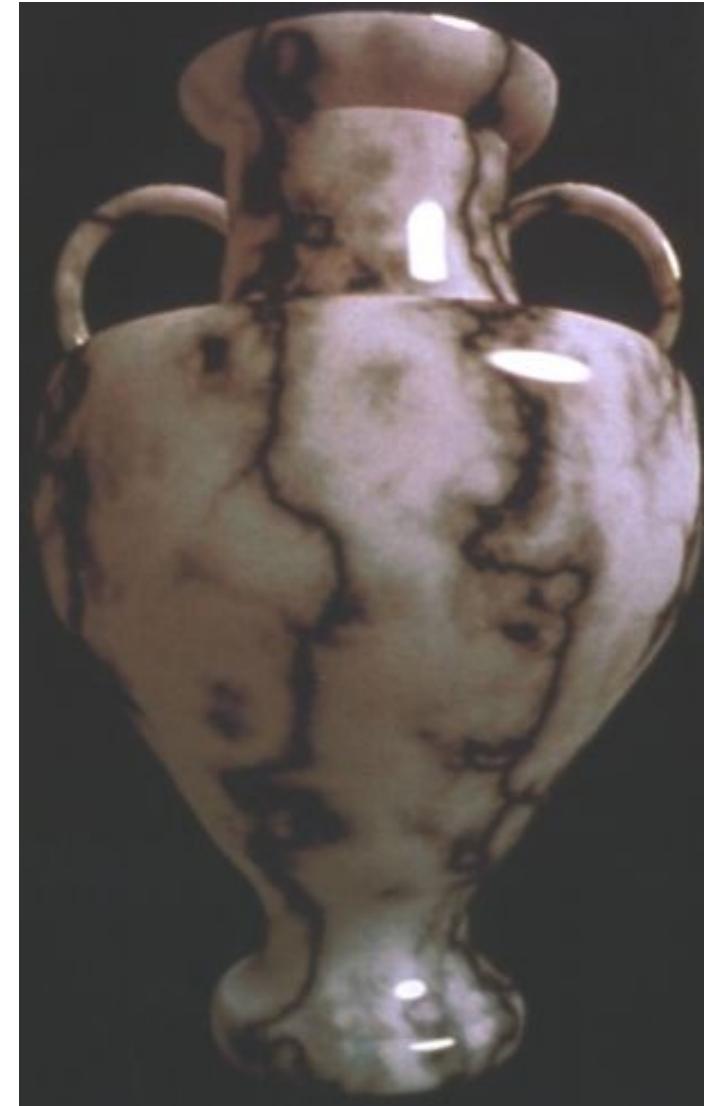


Texture maps (photos, simulations, videos, ...)

- Simple acquisition
- Illumination during acquisition
- Limited resolution, aliasing
- High memory requirements
- Mapping difficult

Procedural textures

- Non-trivial programming
- Flexibility
- Parametric control
- Unlimited resolution, anti-aliasing possible
- Low memory requirements
- Low-cost visual complexity
- Adapts to arbitrary geometry





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

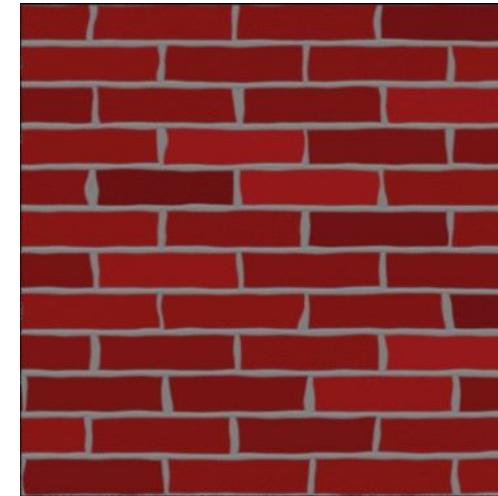
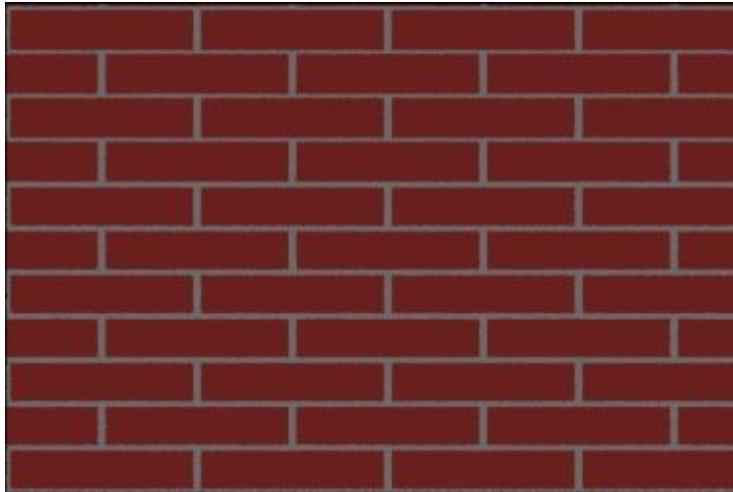
- Similarity between different patches
 - Repetitiveness, coherence
- Similarity on different resolution scales
 - Self-similarity
- But never completely identical
 - Additional disturbances, turbulence, noise

Procedural texture function

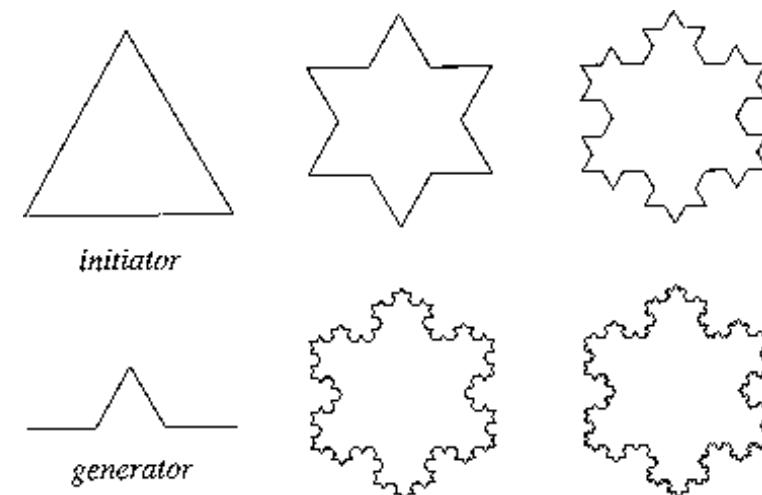
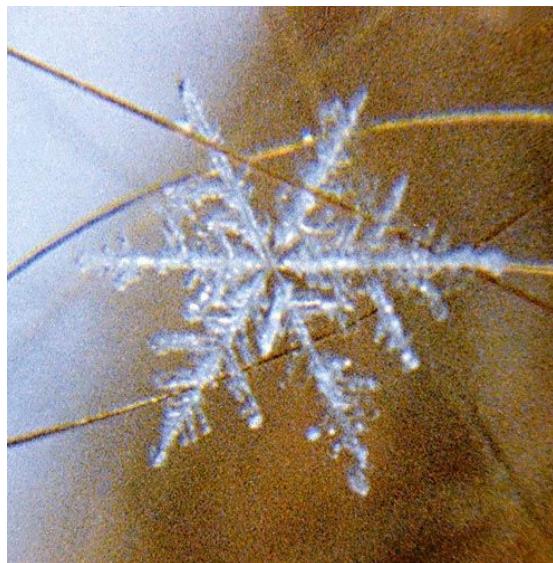
- Mimics statistical properties of natural textures
- Purely empirical approach
 - Looks convincing, but has nothing to do with material's physics



Translational similarity



Similarity on different scales





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)

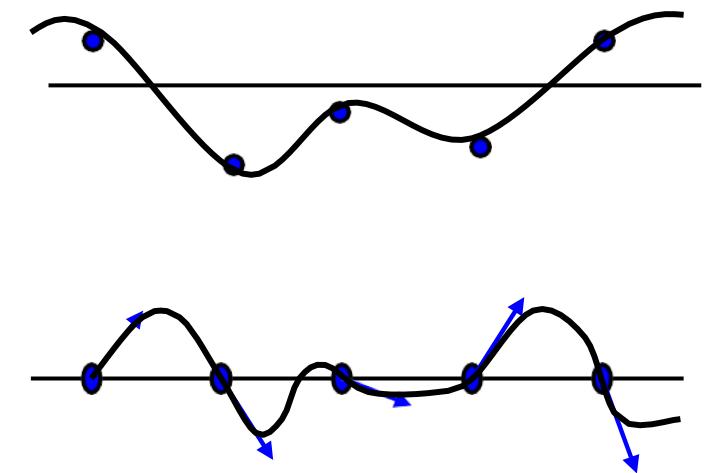
- Tri-linear interpolation
- Cubic interpolation (Hermite spline → later)

Unlimited domain

- Lattice replicated to fill entire space

Fixed fundamental frequency of ~ 1 Hz over lattice

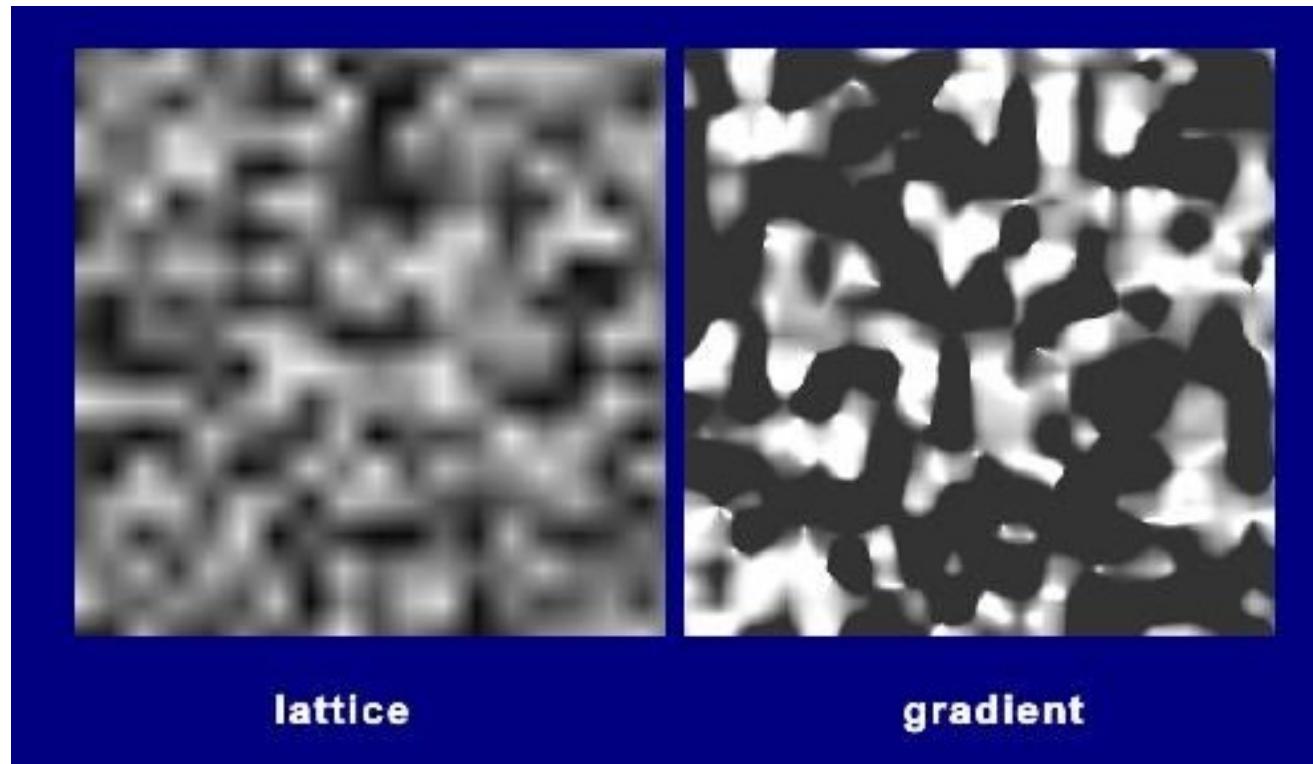
Smooth interpolation of interim values





Gradient noise better than value noise

- less regularity artifacts
- more high frequencies in noise spectrum
- even tri-linear interpolation produces good results





Noise function

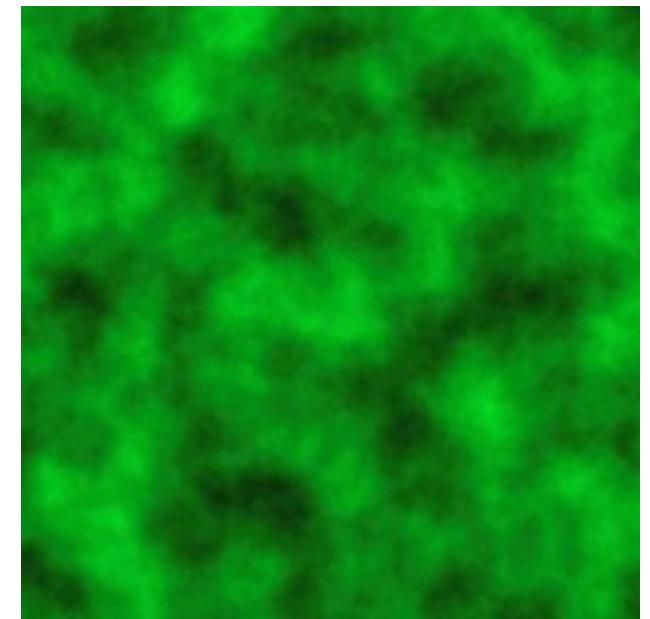
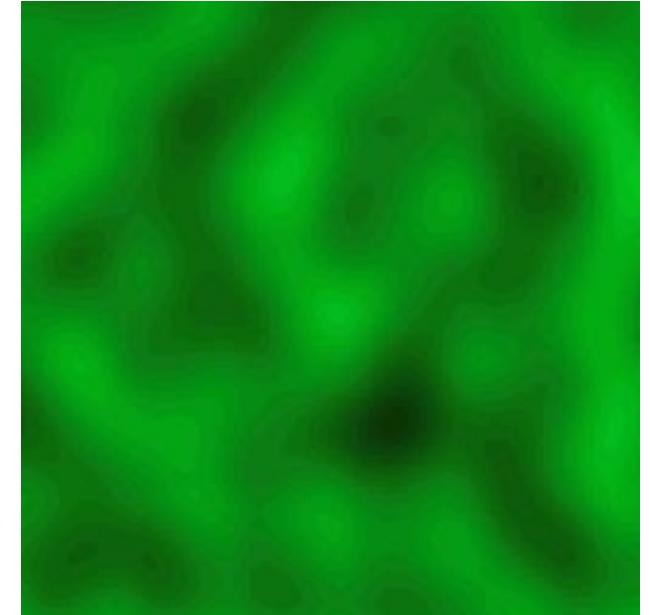
- “White” frequency spectrum

Natural textures

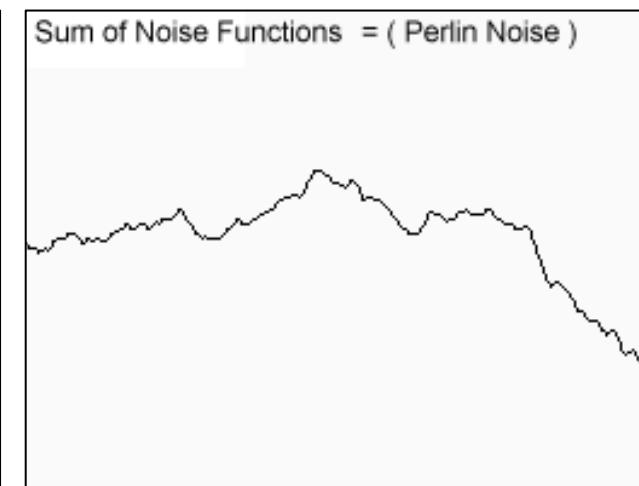
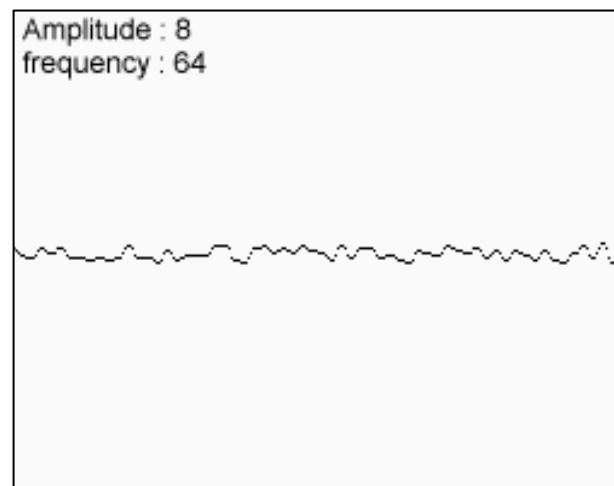
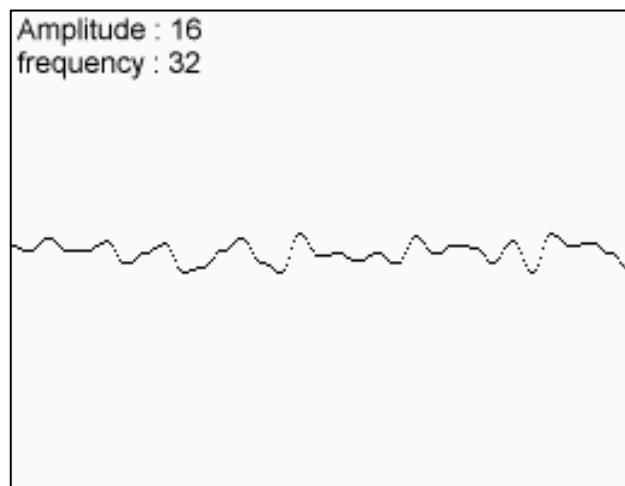
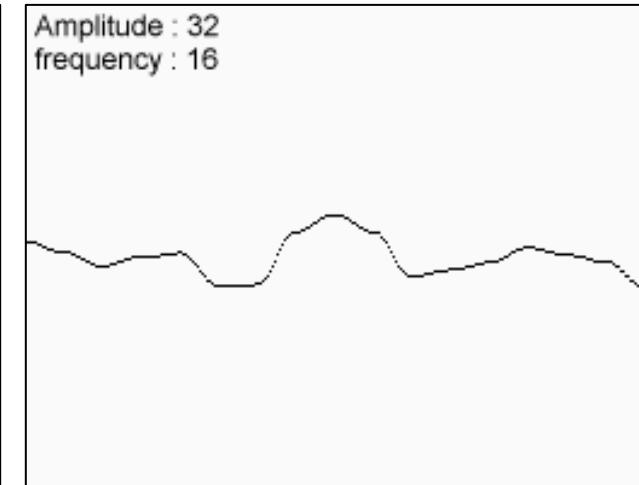
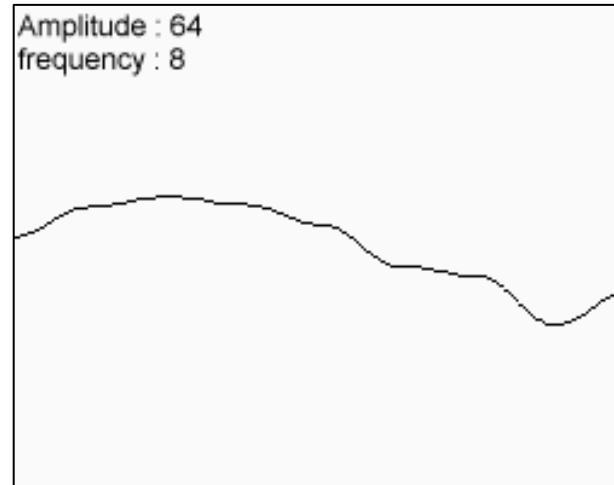
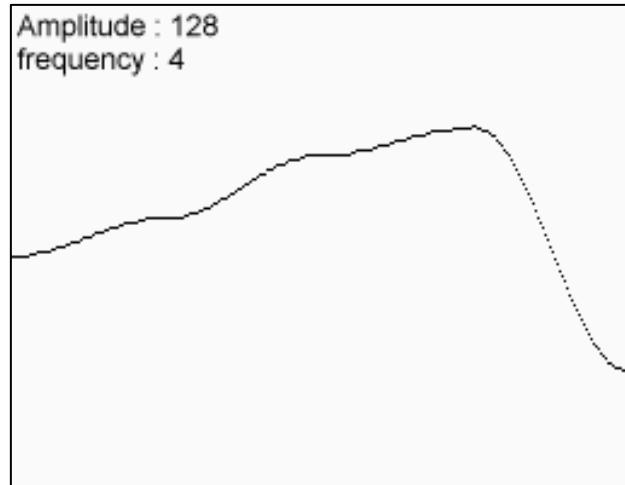
- Decreasing power spectrum towards high frequencies

Turbulence from noise

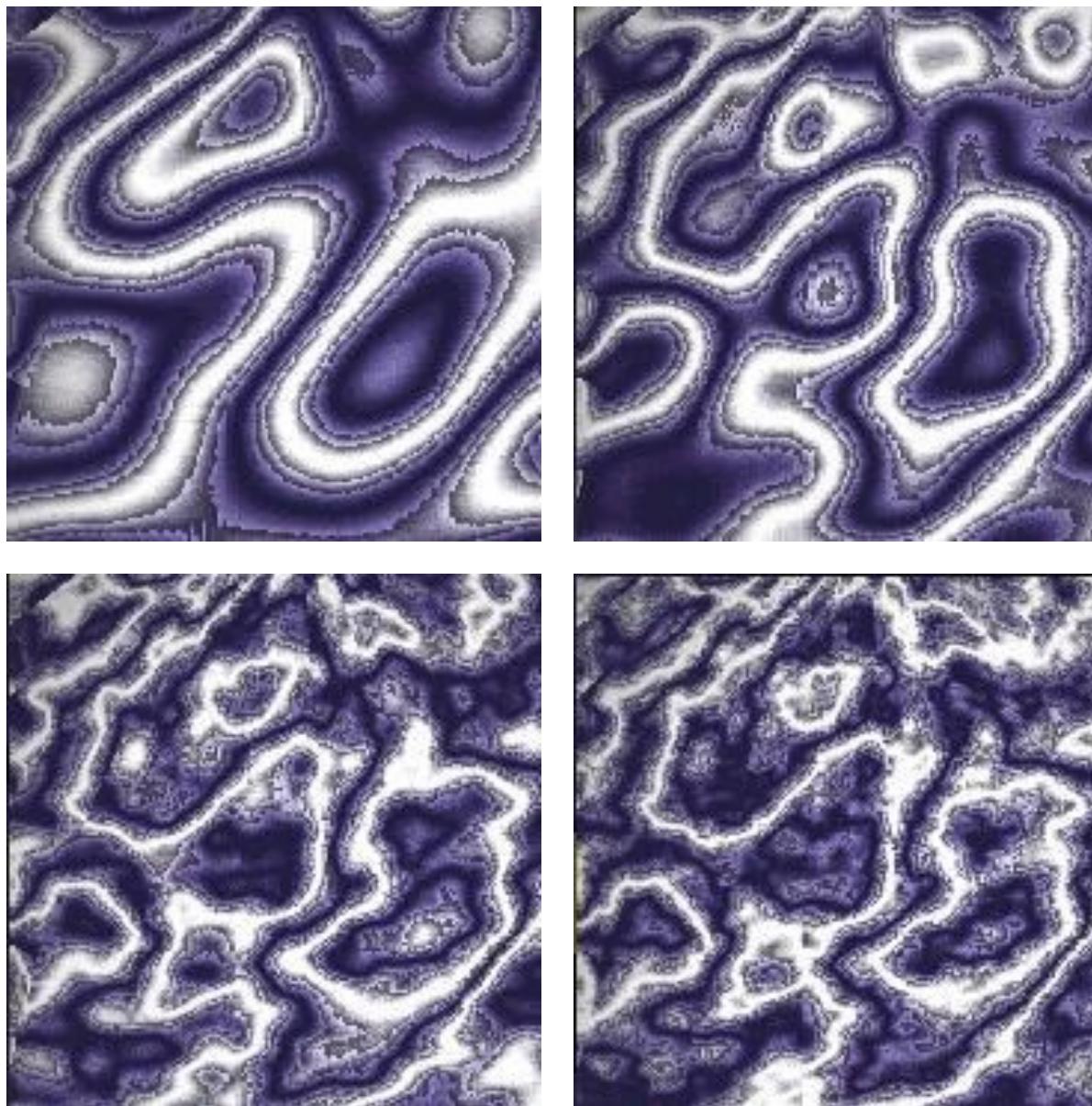
- $Turbulence(x) = \sum_{i=0}^k \left| \frac{\text{noise}(2^i x)}{2^i} \right|$
- Summation truncation
 - $\frac{1}{2^{k+1}} < \text{size of one pixel}$ (band limit)
- 1. Term: $\text{noise}(x)$
- 2. Term: $\frac{\text{noise}(2x)}{2}$
- ...
- Power spectrum: $\frac{1}{f}$
- (Brownian motion: $\frac{1}{f^2}$)



Synthesis of Turbulence (1D)



Synthesis of Turbulence (2D)





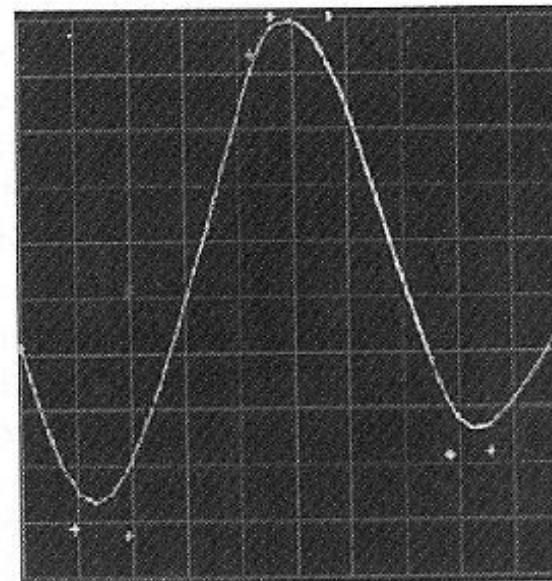
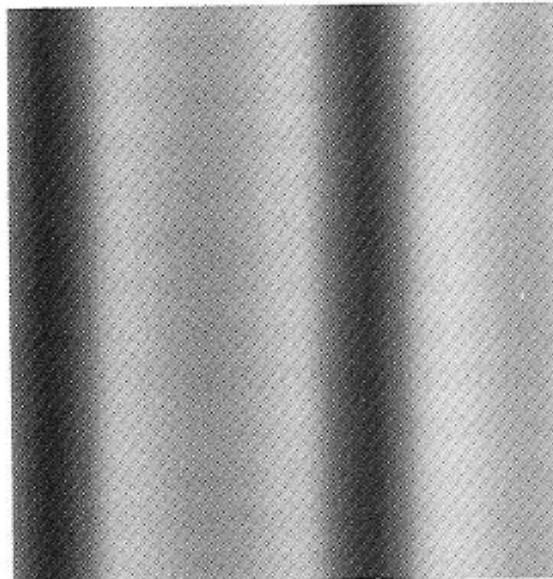
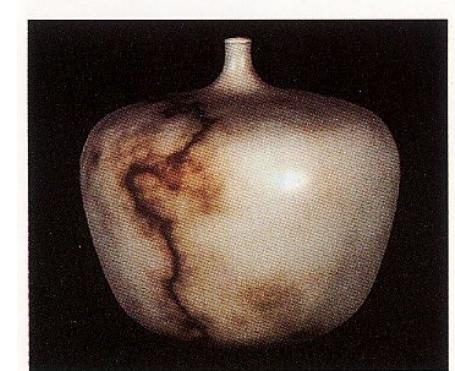
Overall structure: alternating layers of white and colored marble

- $f_{\text{marble}}(x, y, z) := \text{marble_color}(\sin x)$
- marble_color : transfer function

Realistic appearance: simulated turbulence

- $f_{\text{marble}}(x, y, z) := \text{marble_color}(\sin(x + \text{turbulence}(x, y, z)))$

Moving object: turbulence function also transformed





Bark

- Turbulated sawtooth function
- Bump mapping

Clouds

- White blobs
- Turbulated transparency along edge
- Transparency mapping

Animation

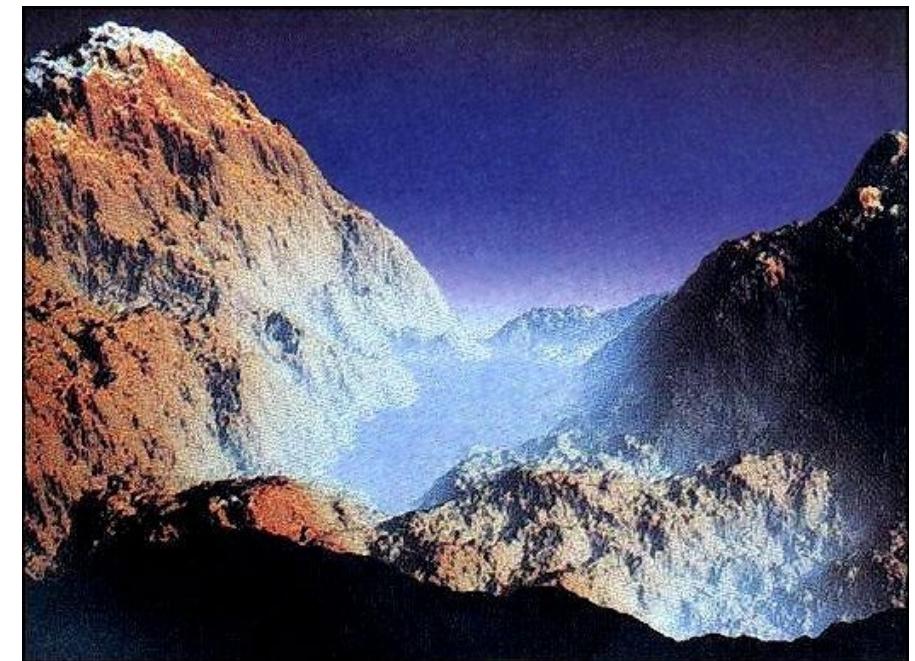
- Vary procedural texture function's parameters over time



Procedural generation of geometry

Complex geometry at virtually no memory cost

- Can be difficult to ray trace !!





Coarse triangle mesh approximation

1:4 triangle subdivision

- Vertex insertion at edge-midpoints

New vertex perturbation

- Displacement along normal
- Random amplitude
- Perturbation scale depends on subdivision level
 - Decreasing power spectrum
 - Parameter for model roughness

Recursive subdivision

- Level of detail (LOD) determined by # subdivisions

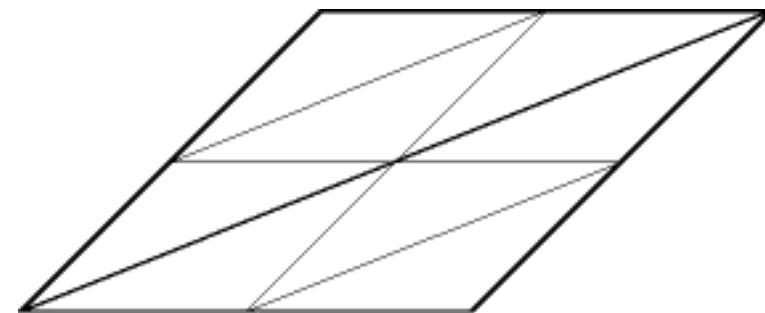
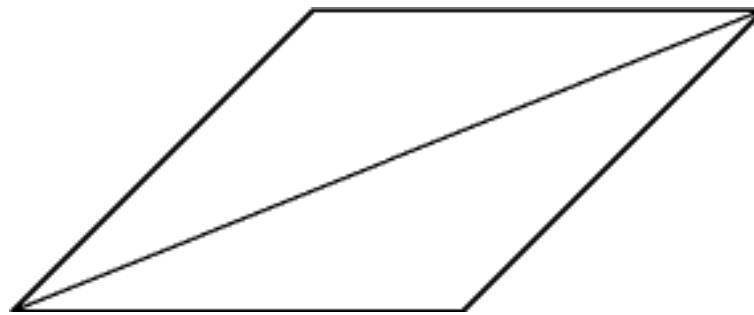
All done inside renderer !

- LOD generated locally when/where needed (bounding box test)
- Minimal I/O cost (coarse mesh only)



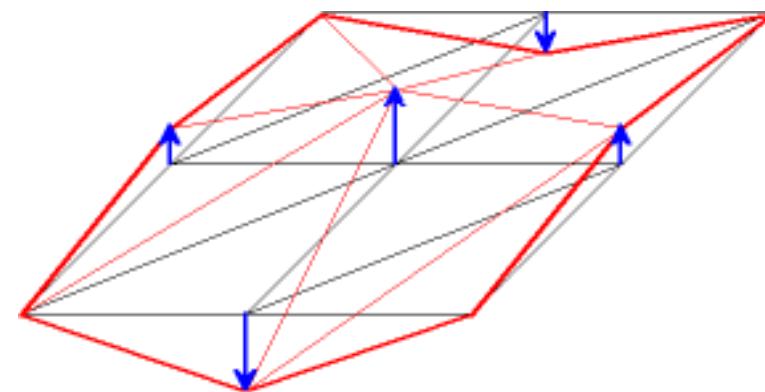
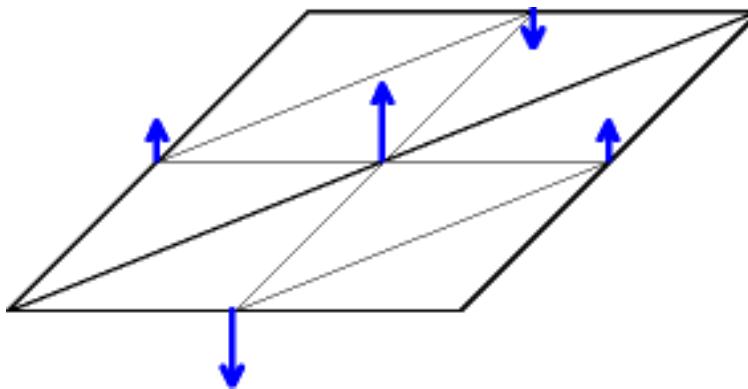
Triangle subdivision

- Insert new vertices at edge midpoints
- 1:4 triangle subdivision



Vertex displacement

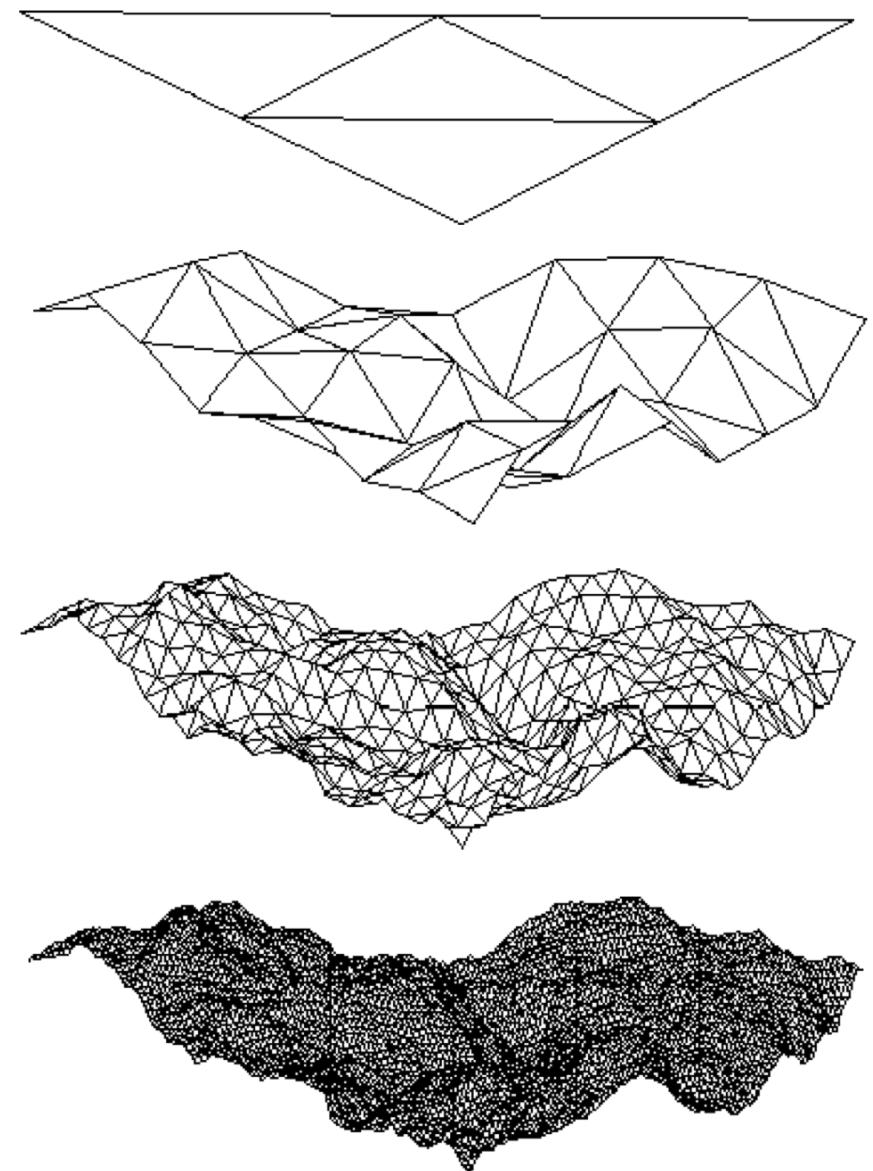
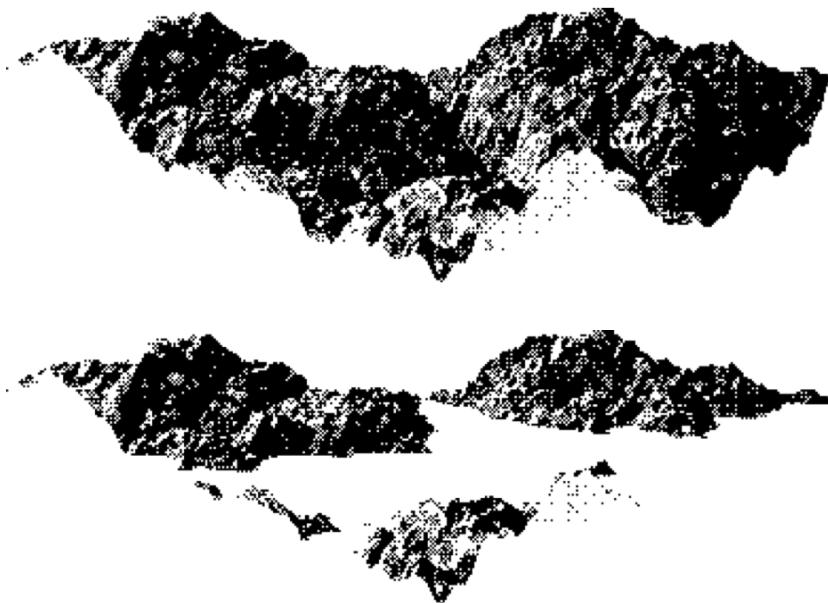
- Along original triangle normal





Fractal Landscape Generation

- Base mesh
- Repeated subdivision & vertex displacement
- Shading
- + Water surface
- + Fog
- + ...





Fractal terrain generated on-the-fly

Problem: where is ray-surface interaction?

- Triangle mesh not a-priori known

Solution: bounding boxes

- Maximum possible bounding box around each triangle
- Decreasing displacement amplitude: finite bounding box

Algorithm

- Intersect ray with bounding box
- Subdivide corresponding triangle
- Compute bounding boxes of 4 new triangles
- Test against 4 new bounding boxes
- Iterate until termination criterion fulfilled (LOD / pixel size)

Assignment 2 (Theoretical part)



Submission deadline: Friday, 11. October 2019 9:45 (before the lecture)

Programming Assignments should be submitted via GitHub (please send your TA the link to your version (fork) of the repository). Every assignment sheet counts 100 points (theory and practice).

2.2 Encapsulate camera and primitives from main application logic (Points 5)

2.3 The Surface-Shader Concept (Points 10 + 10)

2.4 Phong Shading and Point Light sources (Points 30)

2.5 Shadows (Points 20)

2.6 Area Lights (Points 20)

Goto: <https://github.com/Jacobs-University/eyden-tracer-02> (for the task details)

Need Help? Ask TAs! and / or join CG Slack workspace!

- To join the **CG Slack workspace**, please write me at sergej.kosov@gmail.com a short (or empty) e-mail with subject: “Join CG SLACK”. I will reply with *email invitation*.



Assistants

- Kechun Gu
 - Email: k.gu@jacobs-university.de
- Minji Kim
 - Email: m.kim@jacobs-university.de