# Games with Texture Mapping

**CS 4620 Lecture 13**

# Recall first definition…

**Texture mapping:** a technique of defining surface properties (especially shading parameters) in such a way that they vary as a function of position on the surface.



# A refined definition

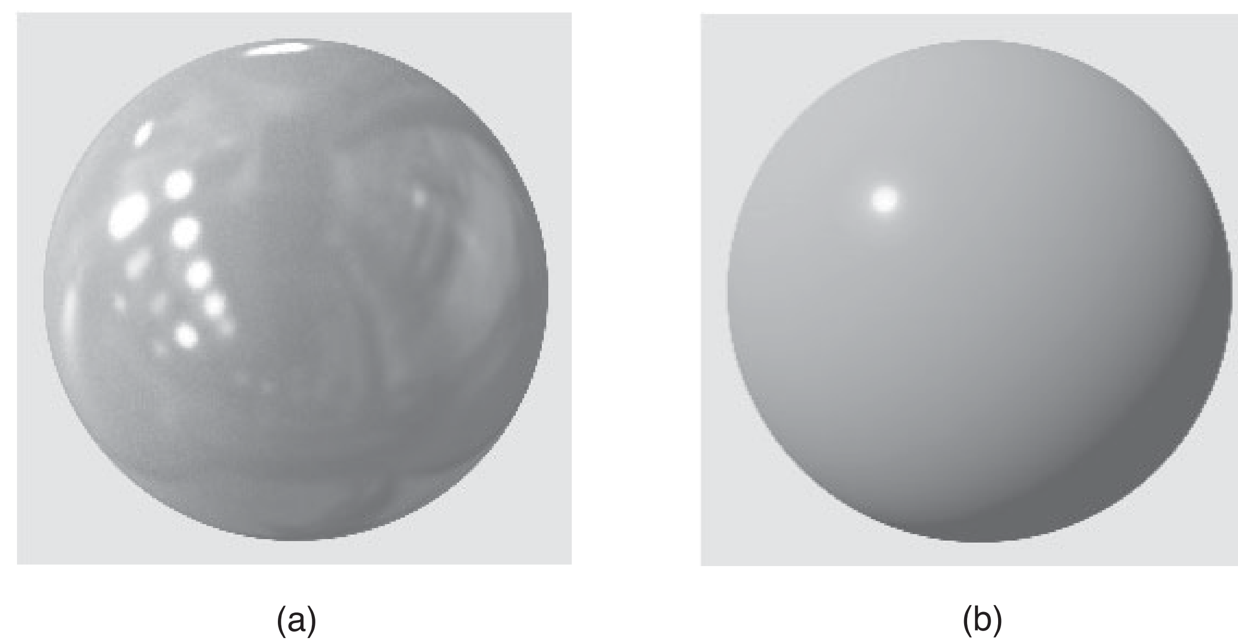
**Texture mapping:** a set of techniques for defining functions on surfaces, for a variety of uses.

- **Let's look at some examples of more general uses of texture maps.**

# Reflection mapping

- **Early (earliest?) non-decal use of textures**
- **Appearance of shiny objects**
  – Phong highlights produce blurry highlights for glossy surfaces.
  – A polished (shiny) object reflects a sharp image of its environment.
- **The whole key to a shiny-looking material is providing something for it to reflect.**




[Dror, Willsky, & Adelson 2004]

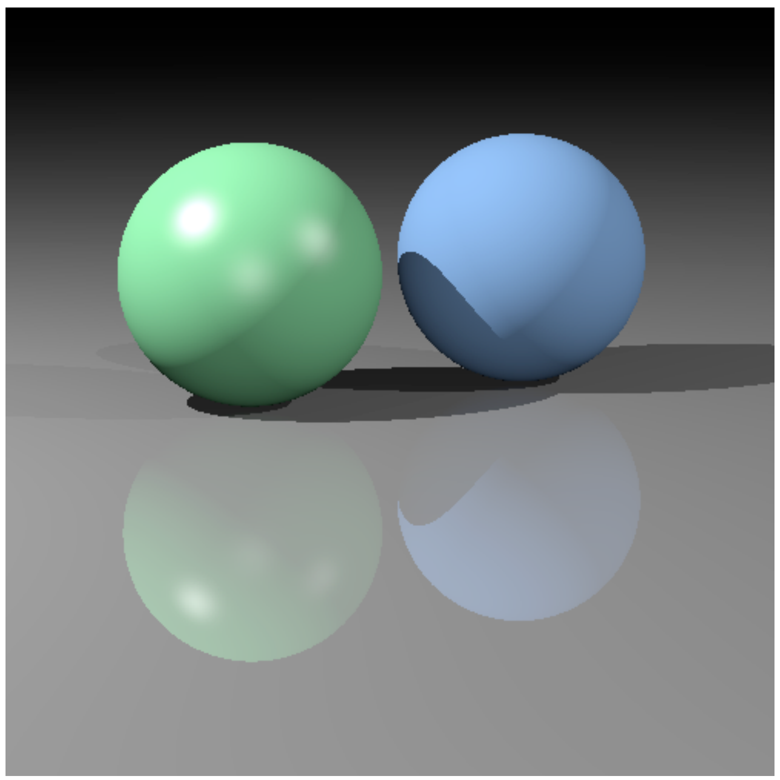
Figure 2. (a). A shiny sphere rendered under photographically acquired real-world illumination. (b). The same sphere rendered under illumination by a point light source.

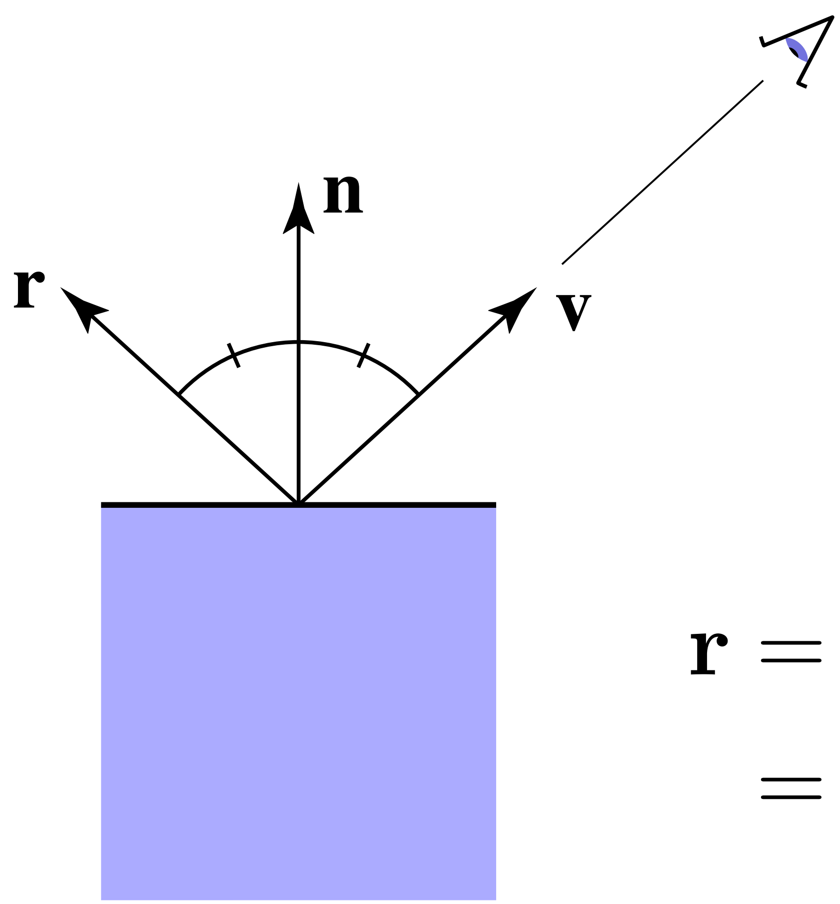


# Reflections in ray tracing

- **Recall how we can make mirror reflections in ray tracing**






$$\begin{aligned}\mathbf{r} &= \mathbf{v} + 2((\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}) \\ &= 2(\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}\end{aligned}$$



# Reflection mapping

- **If scene is infinitely far away, the color seen by the reflection ray depends only on the direction of the ray**
  - a two-dimensional function
  - represent it with a texture!
- **Environment map: texture that maps directions to colors**
  - one option: axes are (theta, phi)
  - better option: cube map


**A spherical panorama, aka. enironment map**

openfootage.net

# Environment map

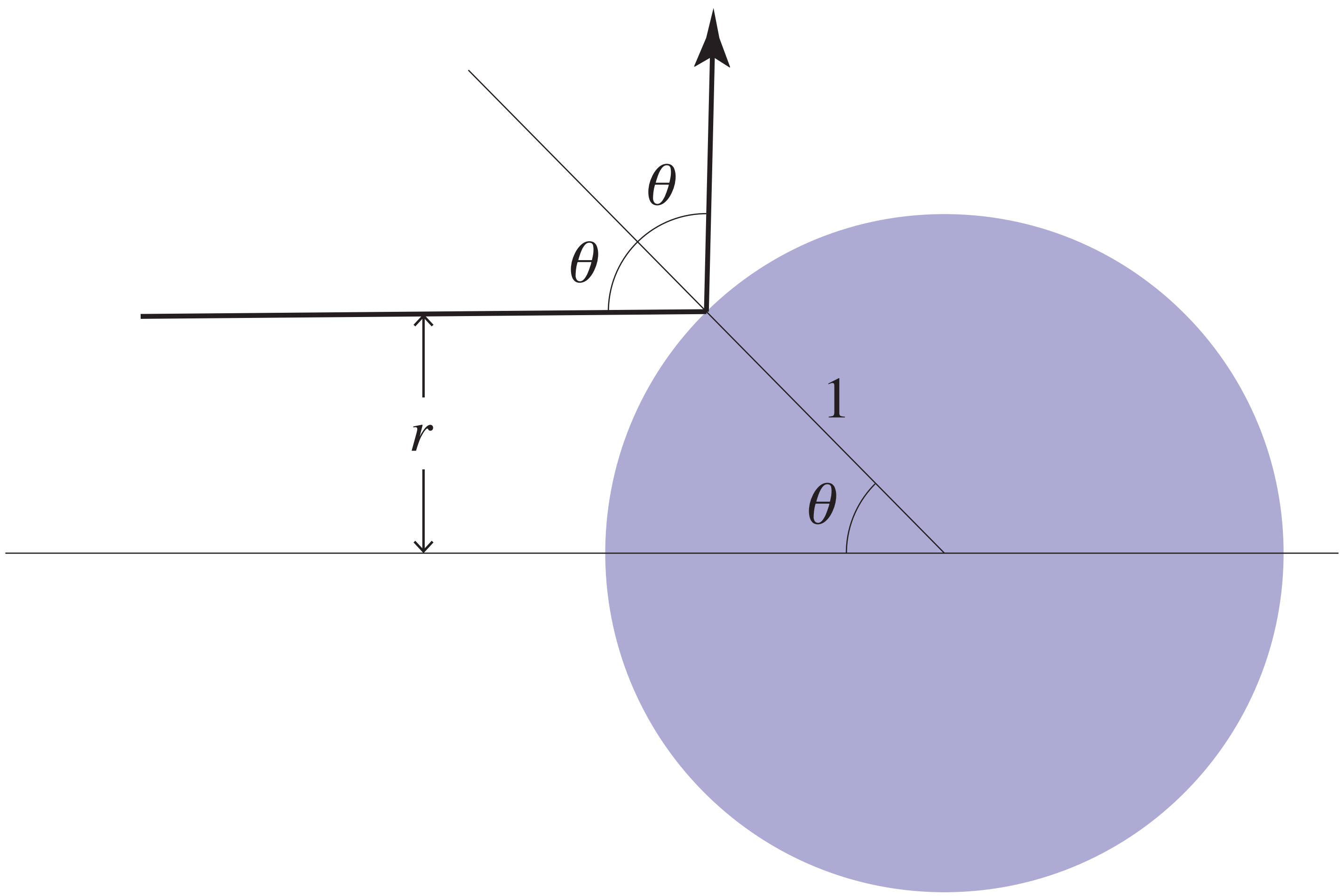
- **A function from the sphere to colors, stored as a texture.**

[Blinn & Newell 1976]



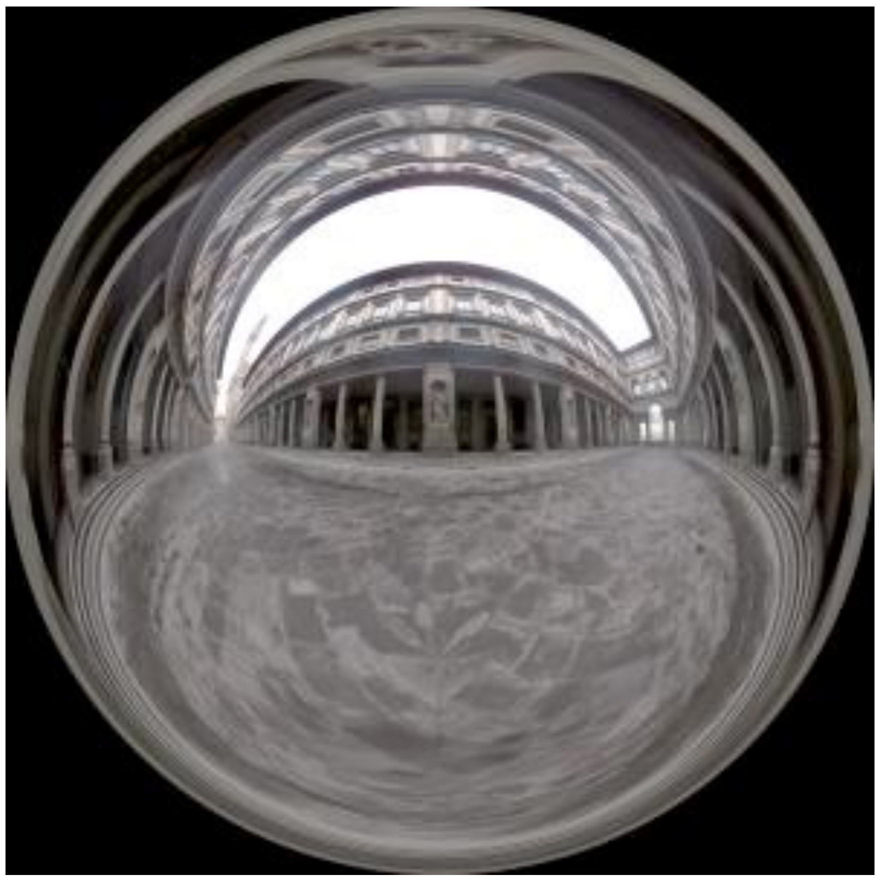
# Spherical environment map

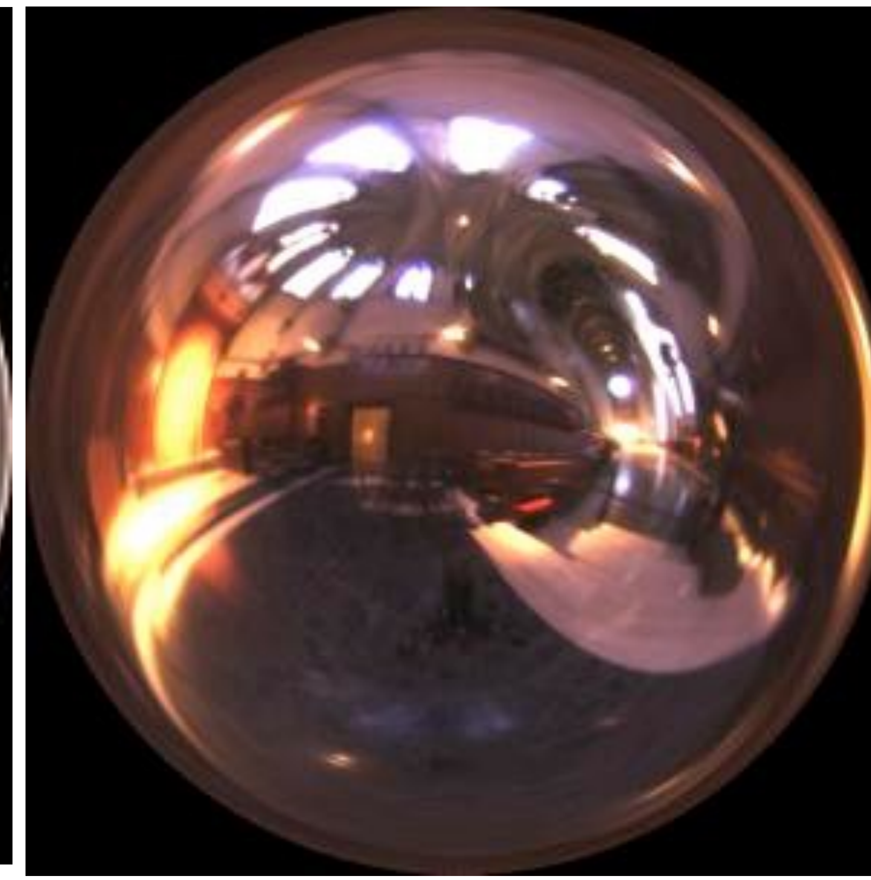




*Hand with Reflecting Sphere*. M. C. Escher, 1935. lithograph

# Environment Maps





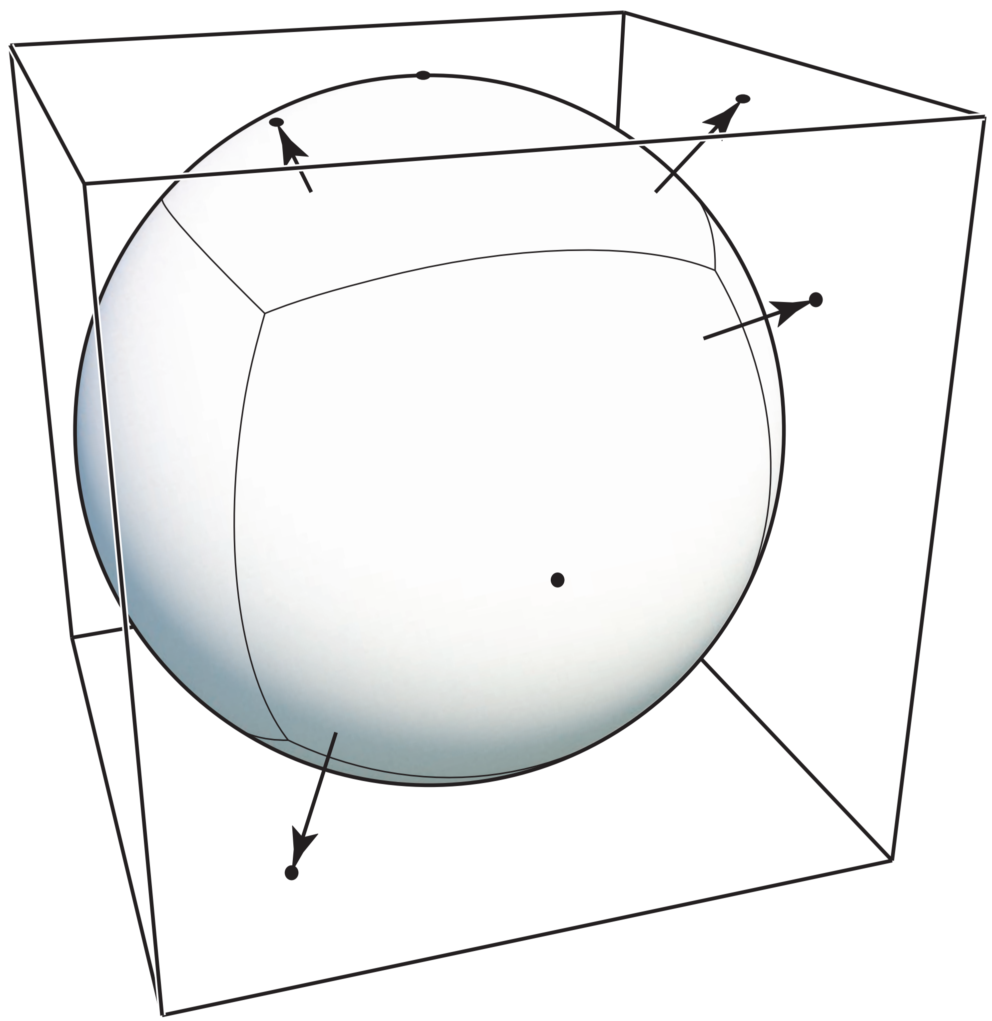
[Paul Debevec]

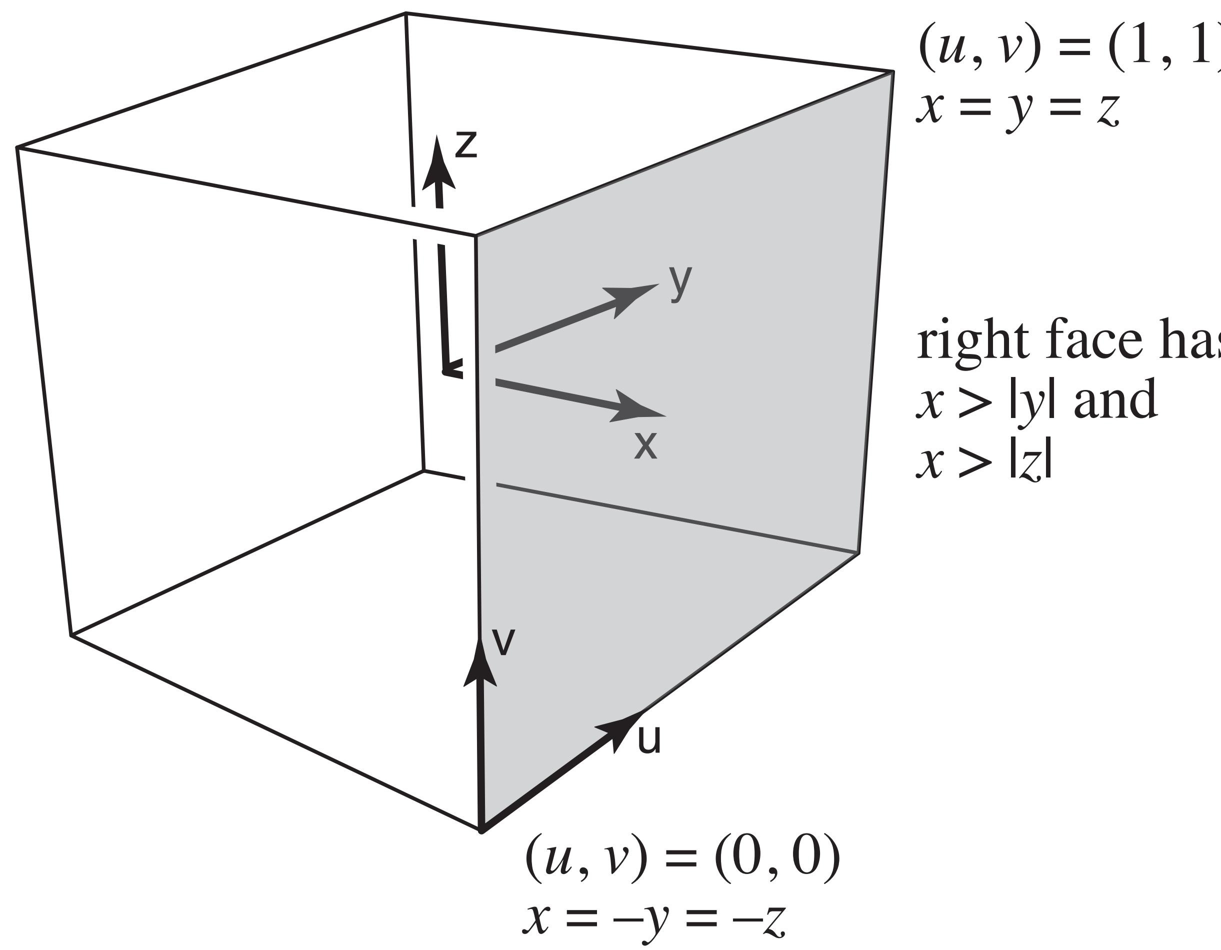


[CS467 slides]

# Cube map

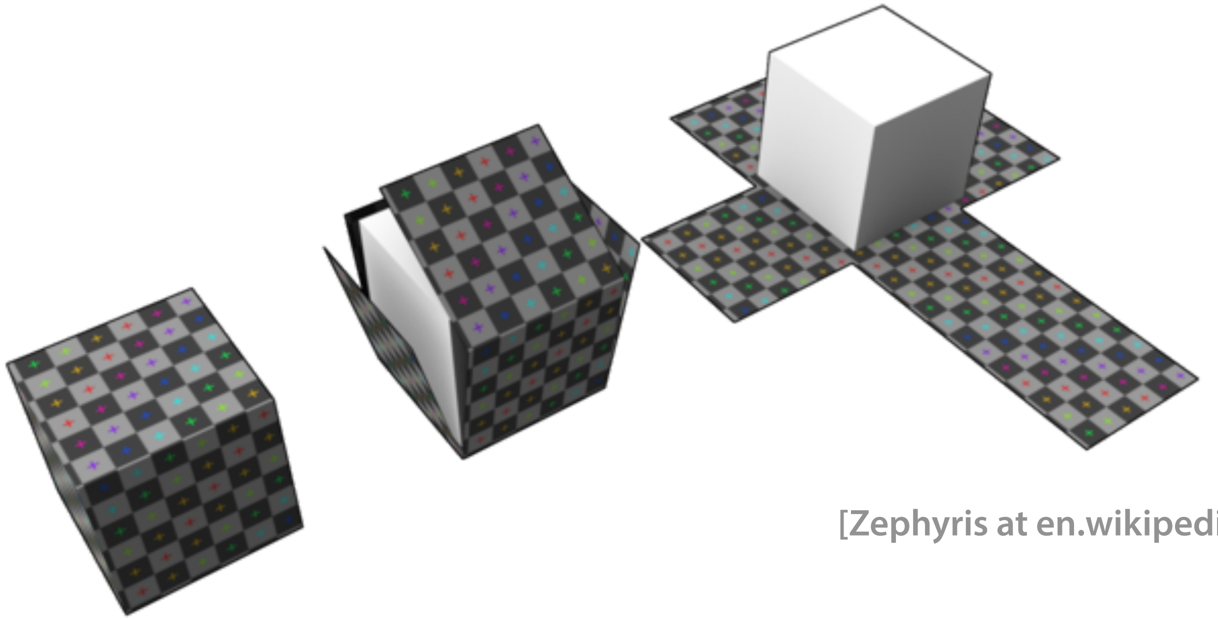





a direction vector maps to the point on the cube that is along that direction.
The cube is textured with 6 square texture maps.





[Zephyris at en.wikipedia]


[Emil Persson]

# Reflection mapping in GLSL

- **A fragment operation**
  - requires surface normal and a way to get the view direction
- **GLSL handles cubemaps by itself**
  - you just give it the reflection vector and it figures out where to sample and on which face
  - sample using `textureCube()`
- **Don't overlook built-in functions**
  - e.g. `reflect()`

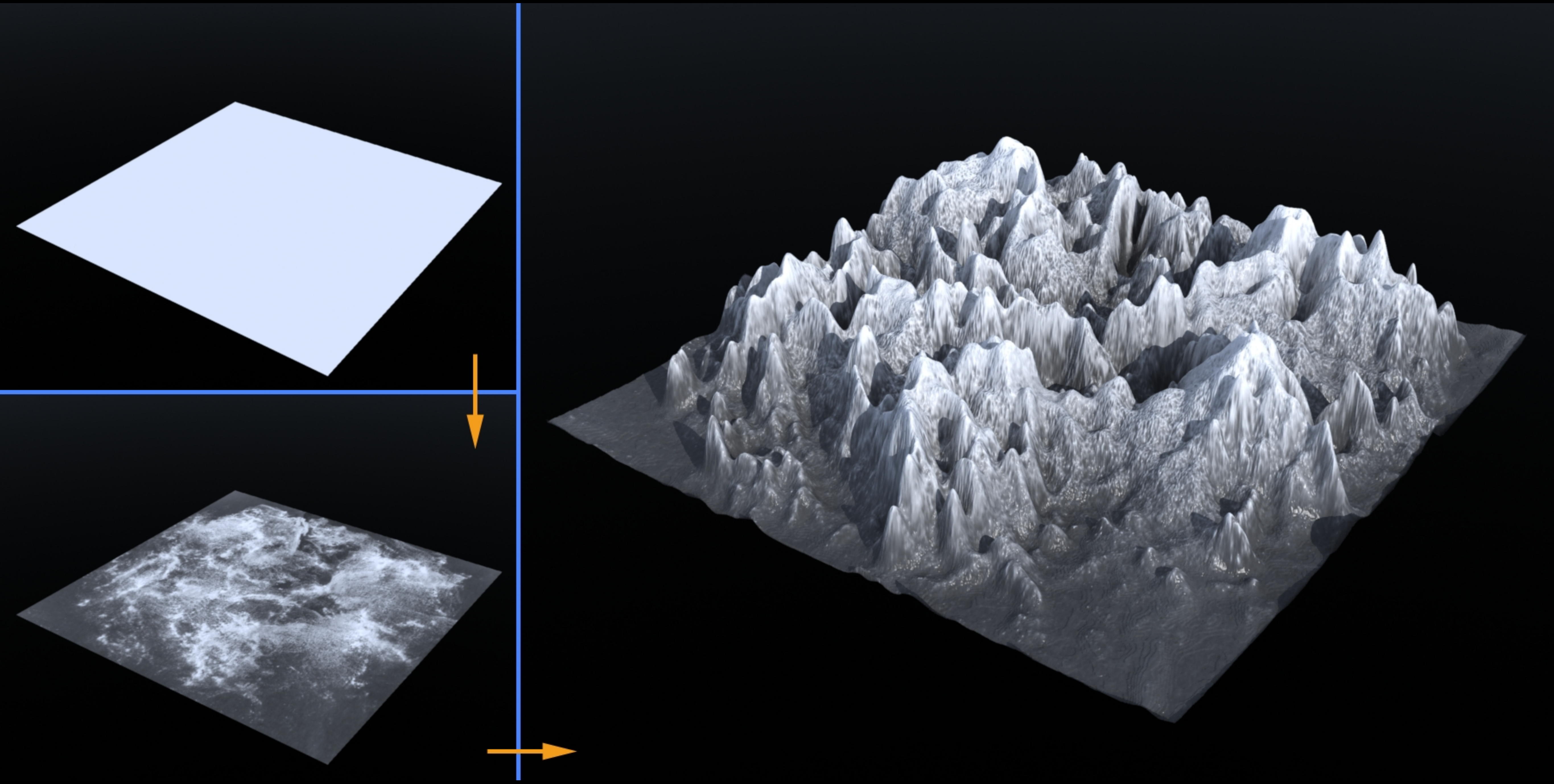


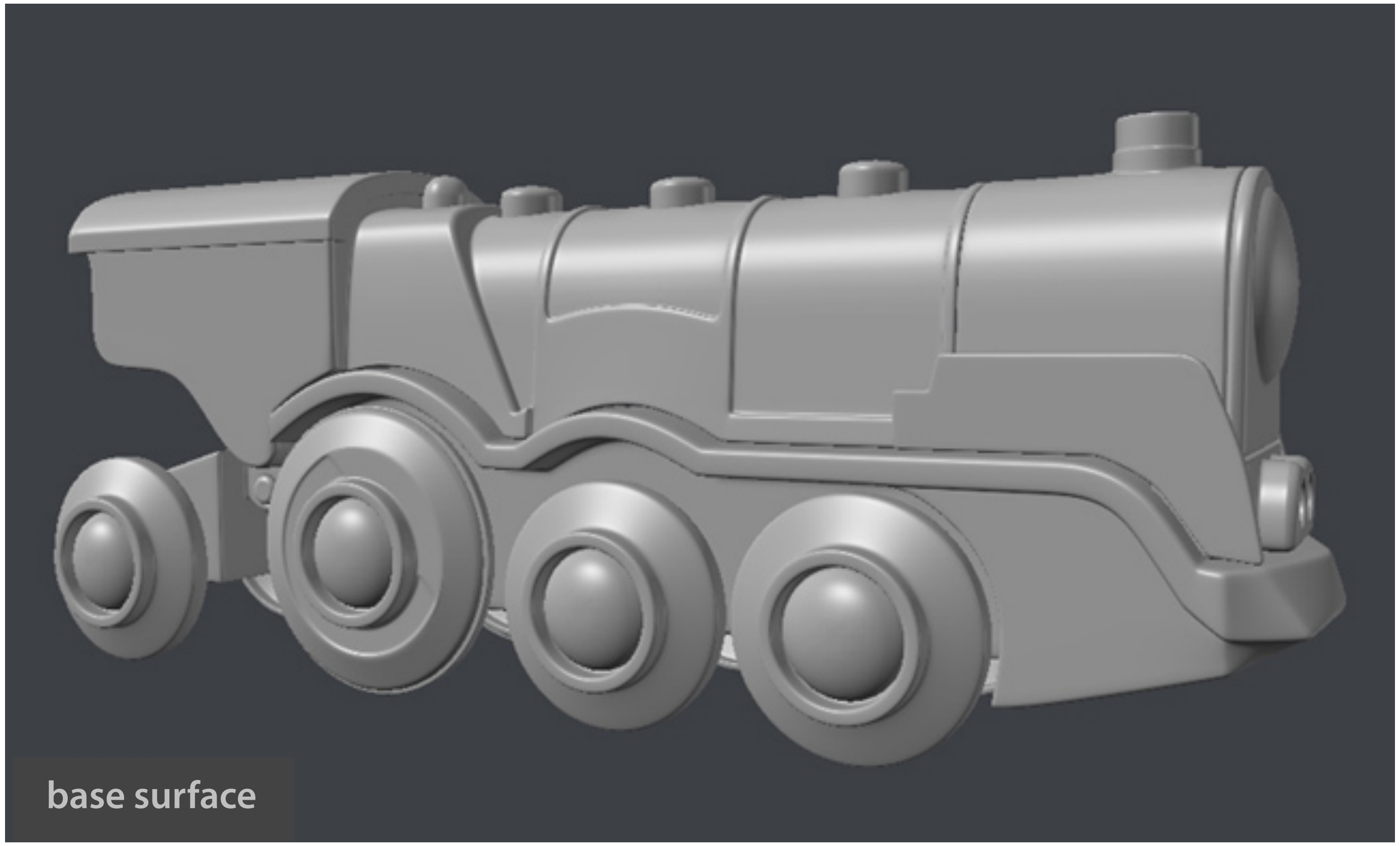
Geometry

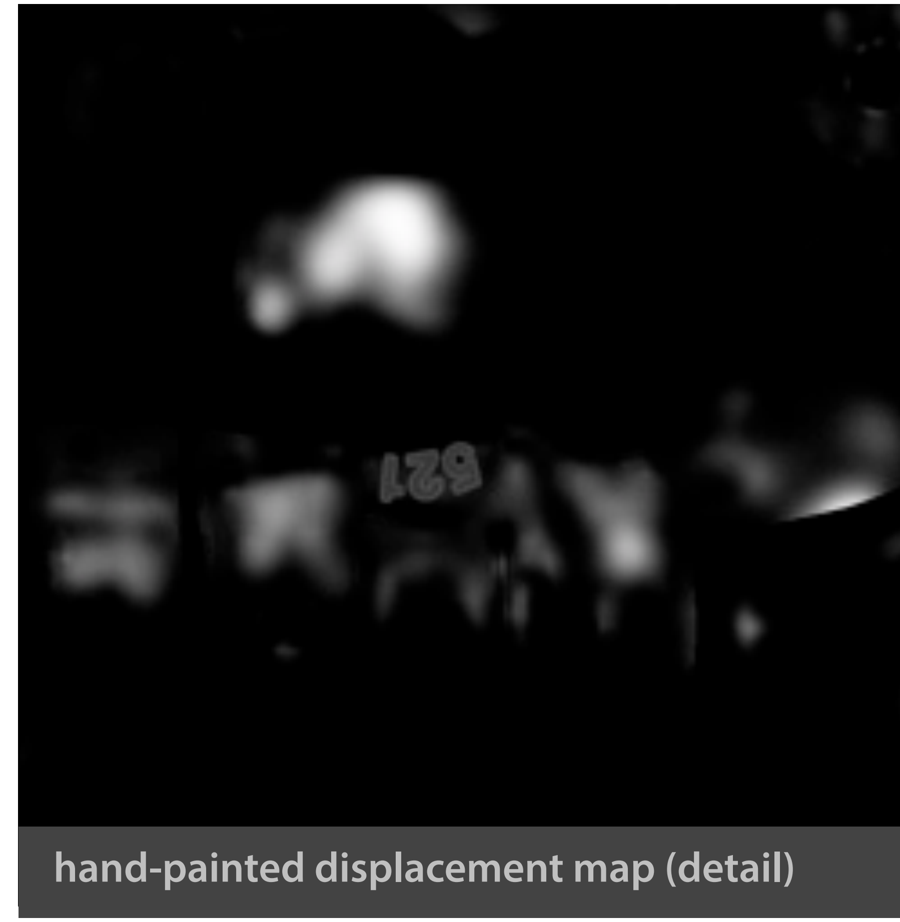
Displacement mapping

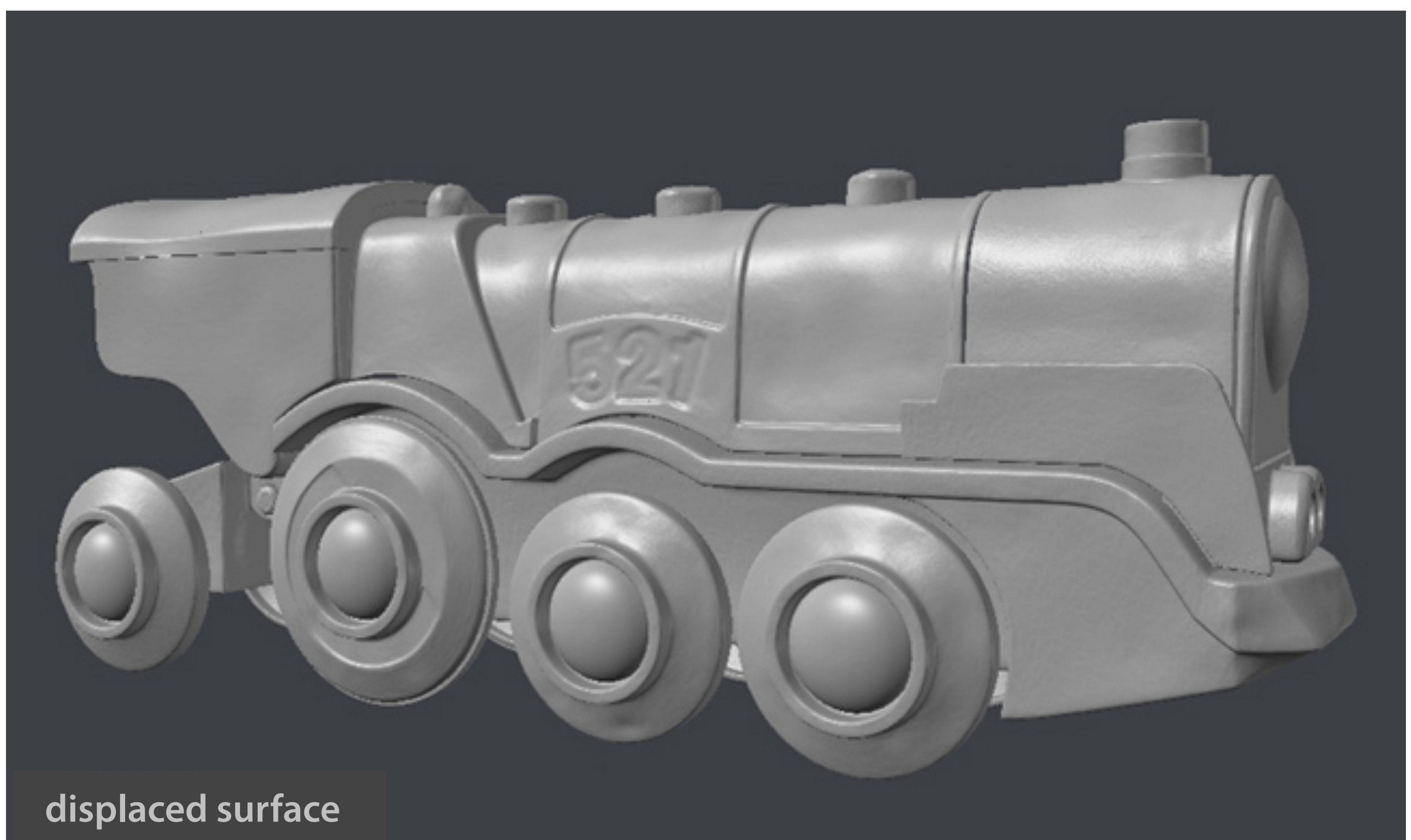
[CS467 slides]



[wikiwand]




**Paweł Filip**
**tolas.wordpress.com**


521
fryrender
physically-based render engine
©2007 Paweł Filip

# Displacement mapping

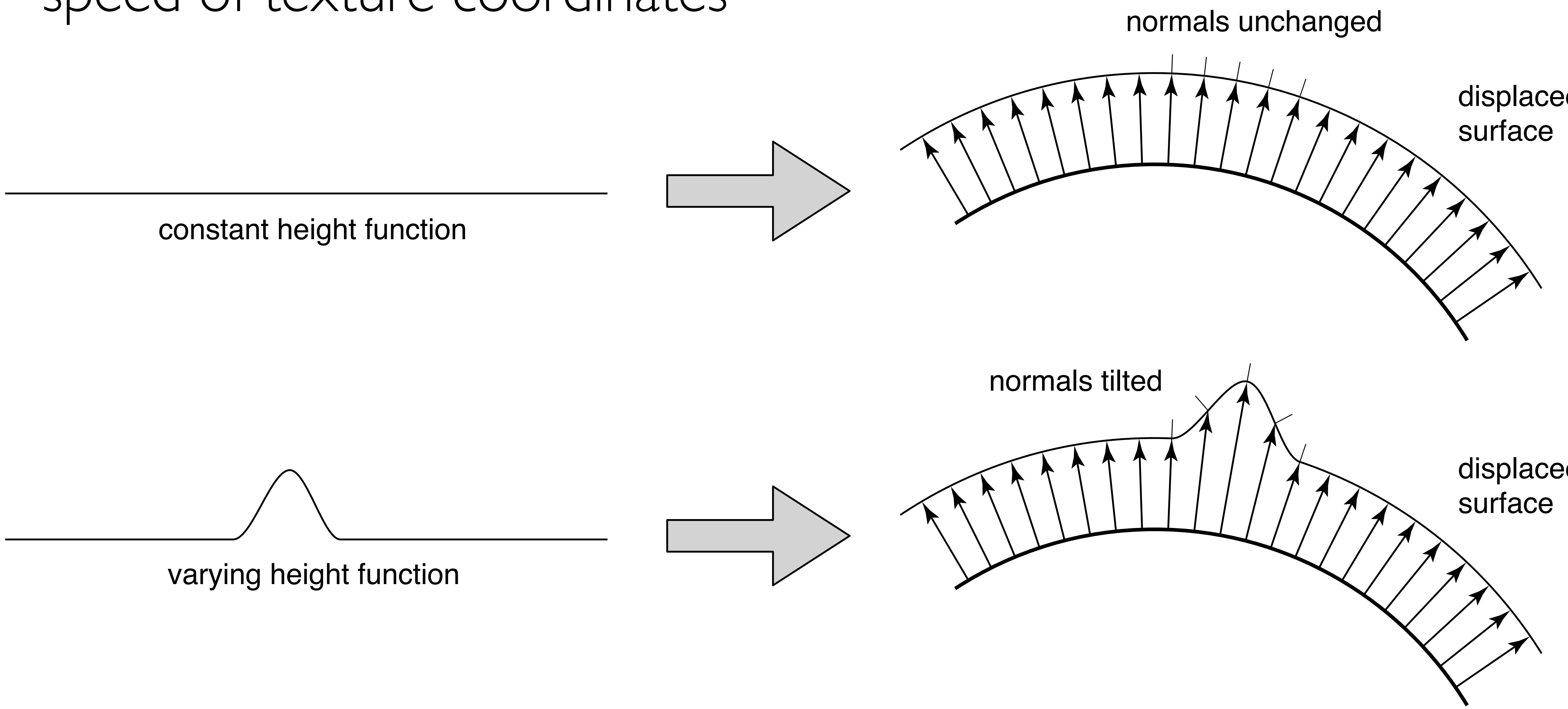
- **A powerful tool for modeling detail**
  - used heavily in film production
- **Geometric prerequisites**
  - texture map representing height field
  - smooth normals
  - texture coordinates
  - dense triangulation
- **In GLSL**
  - a vertex operation (because it moves geometry)
  - displace vertices along normal vectors by a distance proportional to texture map value
  - compute new normal to displaced surface



# Normals in displacement mapping

- **Displacement changes the surface normal, depending on:**
  – derivative of height function
  – orientation of texture coordinates
  – speed of texture coordinates

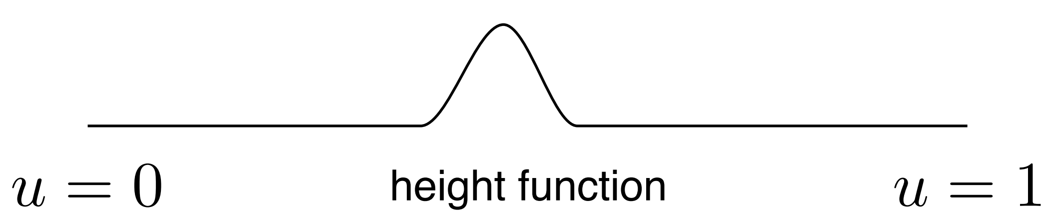


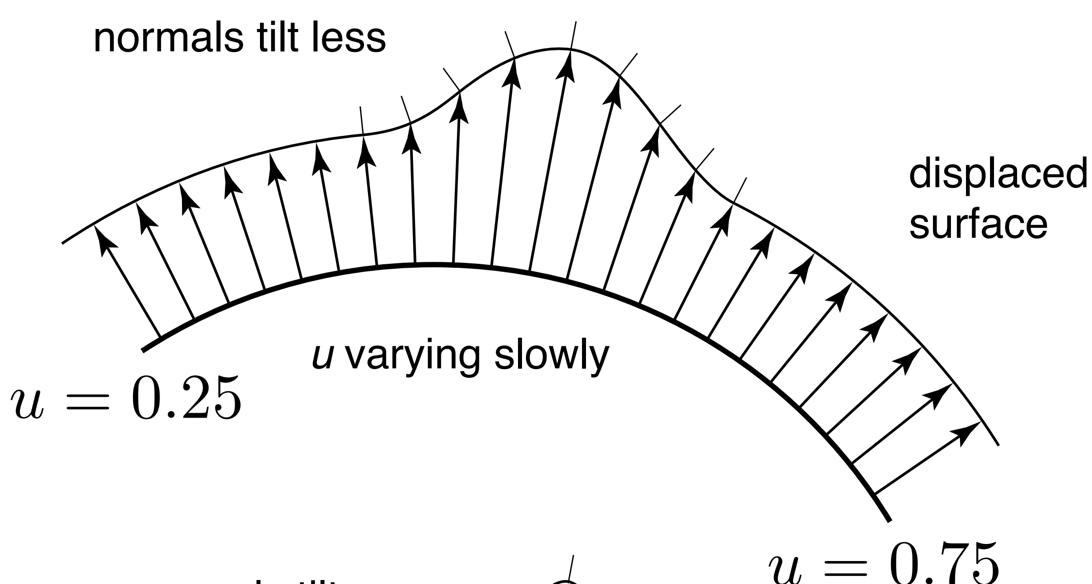


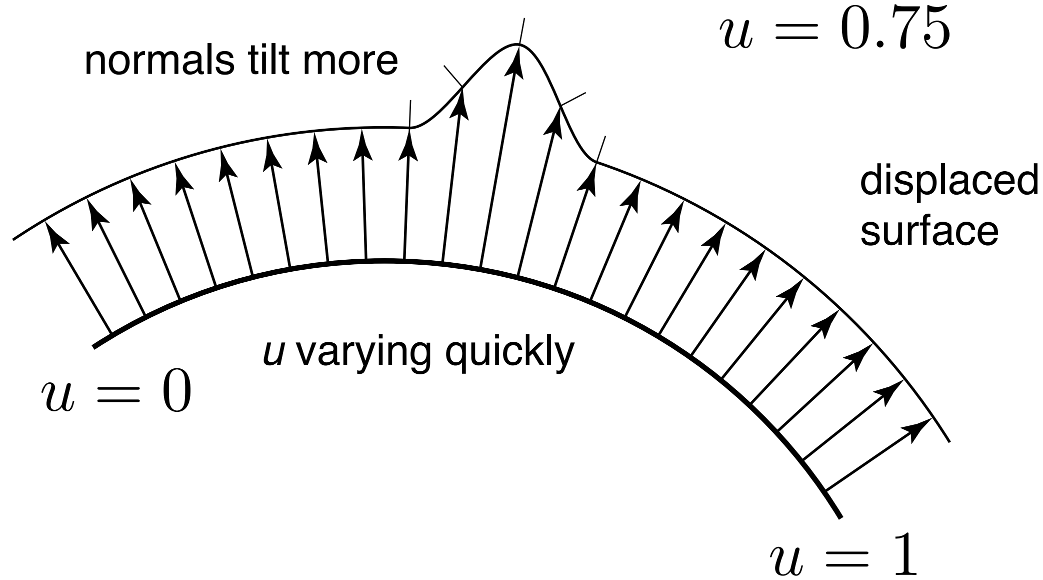
 

# Normals in displacement mapping

- **Displacement changes the surface normal, depending on:**
  - derivative of height function
  - orientation of texture coordinates
  - speed of texture coordinates





# Displacement mapping math

- **Start with a parametric surface and a height function**

$$\mathbf{p}(u,v) : \mathbb{R}^2 \to \mathbb{R}^3 \qquad h(u,v) : \mathbb{R}^2 \to \mathbb{R}$$

  - Recall the tangent vectors are the partial derivatives of p

$$\mathbf{t}_u(u,v) = \frac{\partial \mathbf{p}}{\partial u}(u,v) \qquad \mathbf{t}_v(u,v) = \frac{\partial \mathbf{p}}{\partial v}(u,v)$$

  - …and the normal vector is the cross product of the two tangents.

$$\mathbf{n}(u,v) = \mathbf{t}_u(u,v) \times \mathbf{t}_v(u,v)$$

  - We normalize to make unit tangents and normals, when needed

$$\hat{\mathbf{t}_u} = \frac{\mathbf{t}_u}{\|\mathbf{t}_u\|} \qquad \hat{\mathbf{t}_v} = \frac{\mathbf{t}_v}{\|\mathbf{t}_v\|} \qquad \hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|} = \hat{\mathbf{t}}_u \times \hat{\mathbf{t}}_v$$



# Displacement mapping math

- **Define displaced surface by adding an offset along the normal**

$$\mathbf{p}^d(u,v) = \mathbf{p}(u,v) + h(u,v)\hat{\mathbf{n}}(u,v)$$

(unit normal here because we want *h* to measure the displacement distance)

- **Tangents to the displaced surface**
  - start with tangent in the direction of the *u* texture coordinate

$$\frac{\partial \mathbf{p}^d}{\partial u}(u,v) = \frac{\partial \mathbf{p}}{\partial u}(u,v) + \frac{\partial h}{\partial u}(u,v)\hat{\mathbf{n}}(u,v) + h(u,v)\frac{\partial \hat{\mathbf{n}}}{\partial u}(u,v)$$

  - last term gets messy but only matters for large displacements relative to surface curvature; throw it out. Then the tangents are

$$\mathbf{t}_u^d(u,v) = \mathbf{t}_u(u,v) + \frac{\partial h}{\partial u}(u,v)\,\hat{\mathbf{n}}(u,v)$$

$$\mathbf{t}_v^d(u,v) = \mathbf{t}_v(u,v) + \frac{\partial h}{\partial v}(u,v)\,\hat{\mathbf{n}}(u,v)$$

(non-unit tangents here because the correct result depends on their length)



# Displacement mapping math

- **Last step is to compute the normal to the displaced surface**

$$\mathbf{n}^d = \mathbf{t}_u^d \times \mathbf{t}_v^d$$

$$\hat{\mathbf{n}}^d = \frac{\mathbf{n}^d}{\|\mathbf{n}^d\|}$$



# Geometry for displacement

- **geometric inputs**
  - *u* tangent (unnormalized) as vertex attribute
  - *v* tangent (unnormalized) as vertex attribute
  - height field as a texture
- **vertex stage**
  - compute displaced vertex position
    - look up displacement value from texture
  - compute normal to displaced surface
    - compute derivatives of height by finite differences
    - add offset to the base surface tangents
    - normalized cross product is the shading normal
- **fragment stage: just compute shading**

(or compute them ahead of time and store height and derivatives in a 3-channel texture)



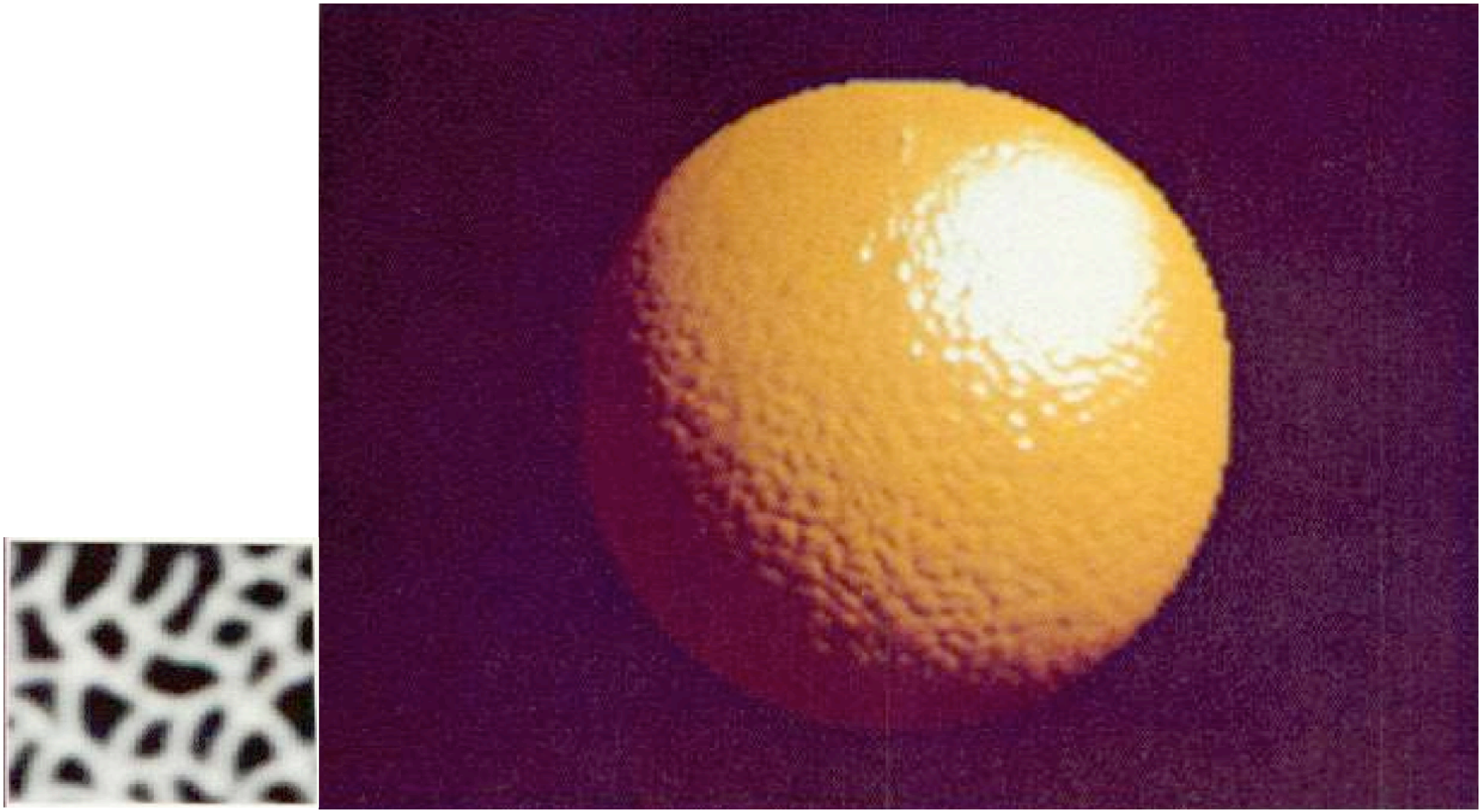
 

# Computing tangent vectors

- **How do we get these tangent vectors?**
  - they need to be stored at vertices on the mesh, like normals
- **For a triangle, there's a unique linear map from (u,v) to (x,y,z)**
  - the derivatives of that map are the (non-unit) tangents
  - can be computed by solving three 2x2 linear systems
  - math resembles triangle setup for rasterization; details here
- **For displacement mapping you want to leave the tangents unnormalized and non-orthogonal**
- **For other uses it's often handy to make the two tangents and the normal into an ONB**
  - use exactly the basis-from-two math that we have used for cameras and manipulators

# Bump mapping

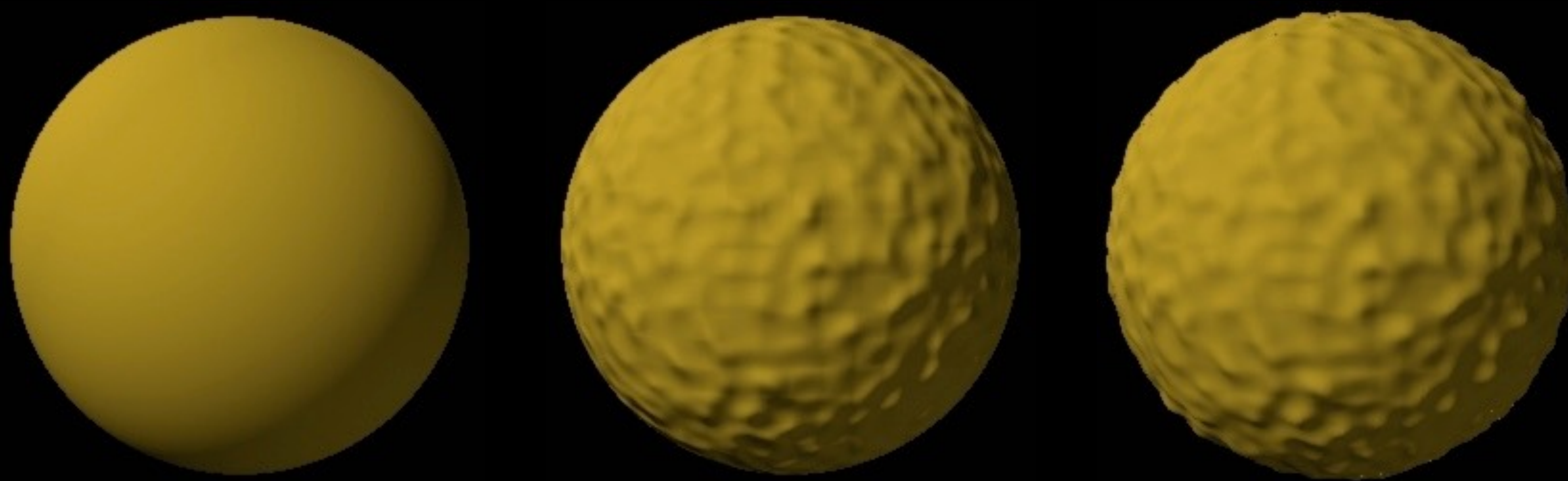


[Blinn 1978]

# Bump mapping

- **Displacement mapping is expensive**
  – requires densely tessellated geometry
  – many triangles to rasterize
- **For small displacements, the most important effect is on the normal**
  – so just do that part; don't displace the surface
- **Bump mapping is then a fragment operation**
  – doesn't require dense tessellation
  – doesn't actually displace the surface
  – gives shading that looks just like displaced surface



Geometry | Bump mapping | Displacement mapping

[CS467 slides]

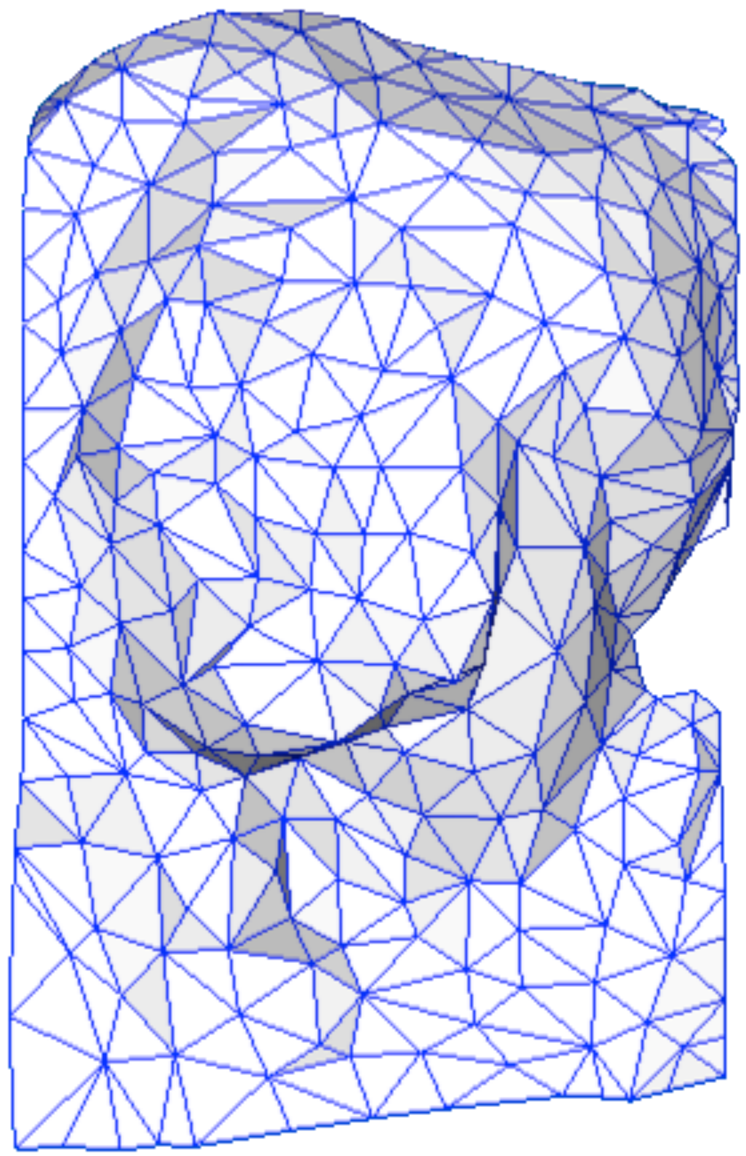
# Bump mapping

- **Geometric inputs**
  - tangent vectors (unnormalized) as vertex attributes
  - height field as a texture
  - no dense triangulation needed
- **Vertex phase**
  - simply transform and pass through the position and tangents
- **Fragment phase**
  - compute normal to displaced surface
    - compute derivatives of height by finite differences
    - add offset to the base surface tangents
    - normalized cross product is the shading normal
  - compute shading using displaced normal

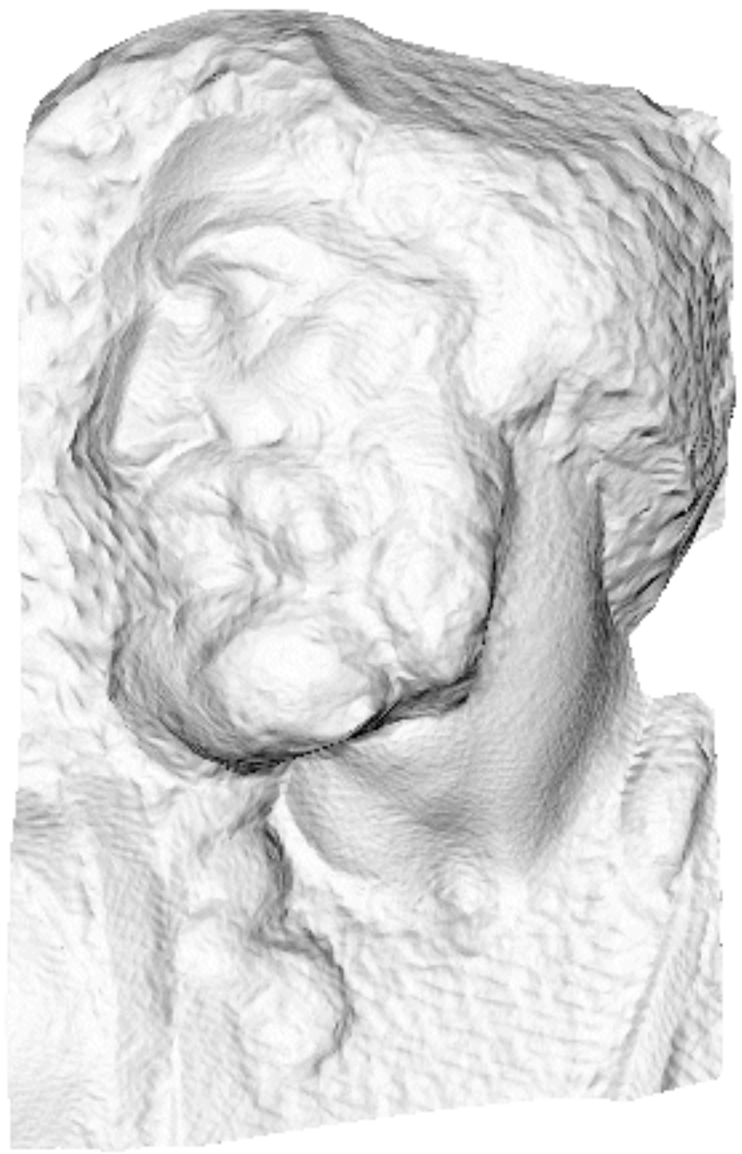
(or compute them ahead of time and store in a 2-channel texture)

# Normal mapping

original mesh
4M triangles



simplified mesh
500 triangles



simplified mesh
and normal mapping
500 triangles

[Paolo Cignoni]

# Normal mapping

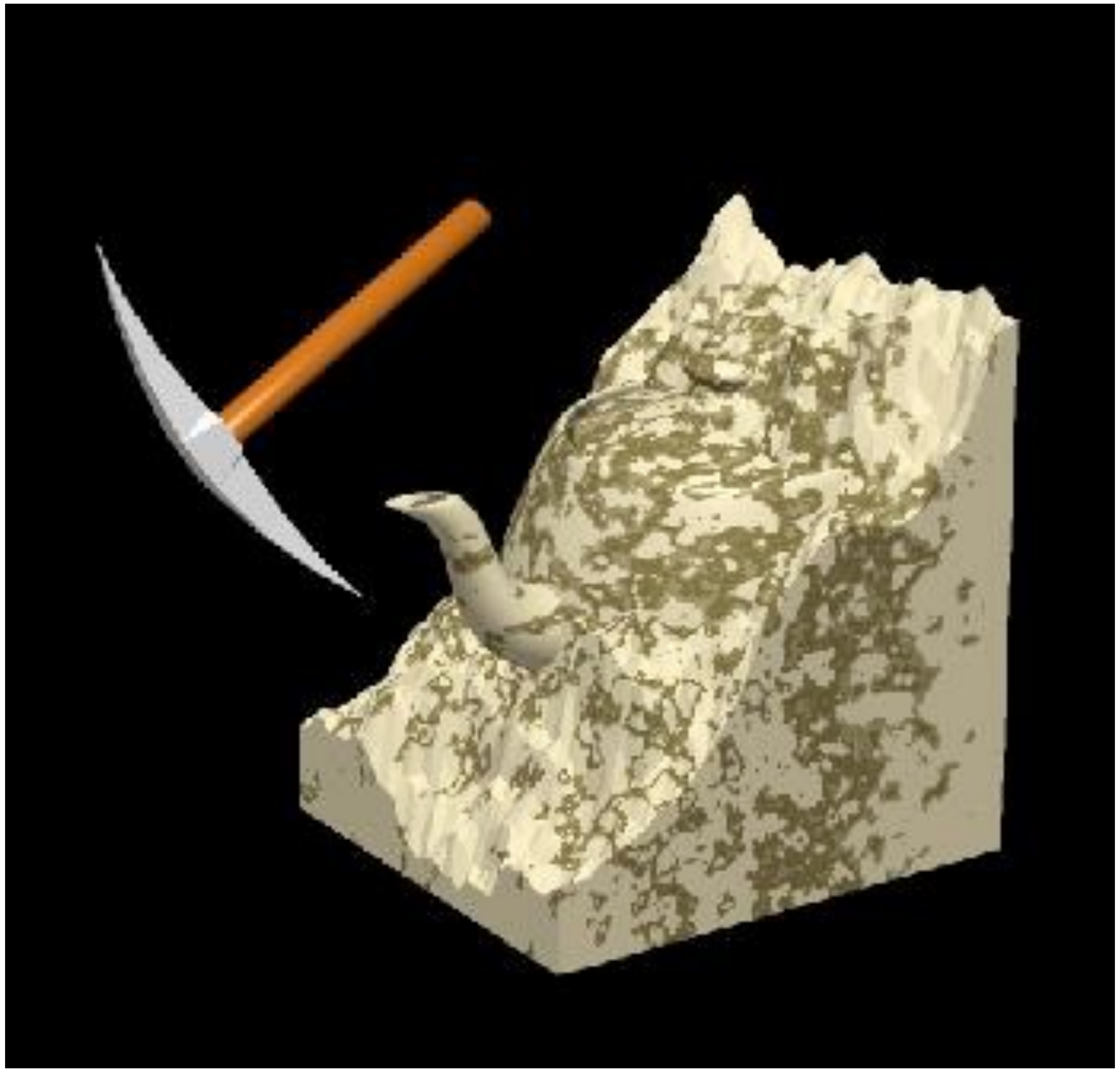
- **Geometric prerequisites**
  - Texture map (3 channels) representing normal field
    - single lookups into normal map required
  - Smooth normals
  - Unit tangent vectors
    - if you want to store normals in tangent space (and you do)
  - No dense triangulation needed
  - No finite differencing needed
- **Geometric logic**
  - look up normal from map
  - transform into (tangent-u, tangent-v, normal) space



# 3D textures

- **Texture is a function of (*u*, *v*, *w*)**
  - can just evaluate texture at 3D surface point
  - good for solid materials
  - often defined procedurally
  - see book for more!



[Wolfe / SG97 Slide set]

