# Texture Mapping

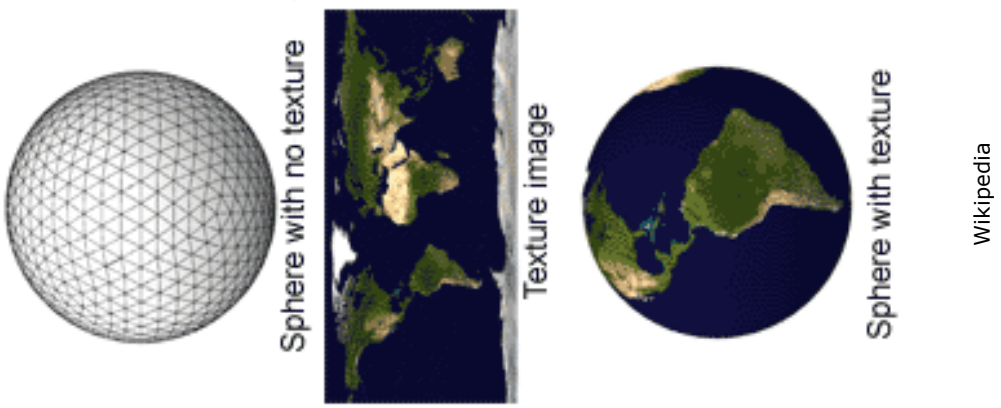
## aka Beautification of Surfaces

Microsoft Flight Simulator

# Texture Mapping

- Texture mapping:
  - Implemented in hardware on every GPU
  - Simplest surface detail hack, dating back to the '60s GE flight simulator and its terrain generator

- Technique:
  - "Paste" photograph or bitmap (the texture, for example: a brick pattern, a wood grain pattern, a sky with clouds) on a surface to add detail without adding more polygons.
  - Map texture onto the surface to get the surface color or alter the object's surface color
  - Think of texture map as stretchable contact paper



# Motivation

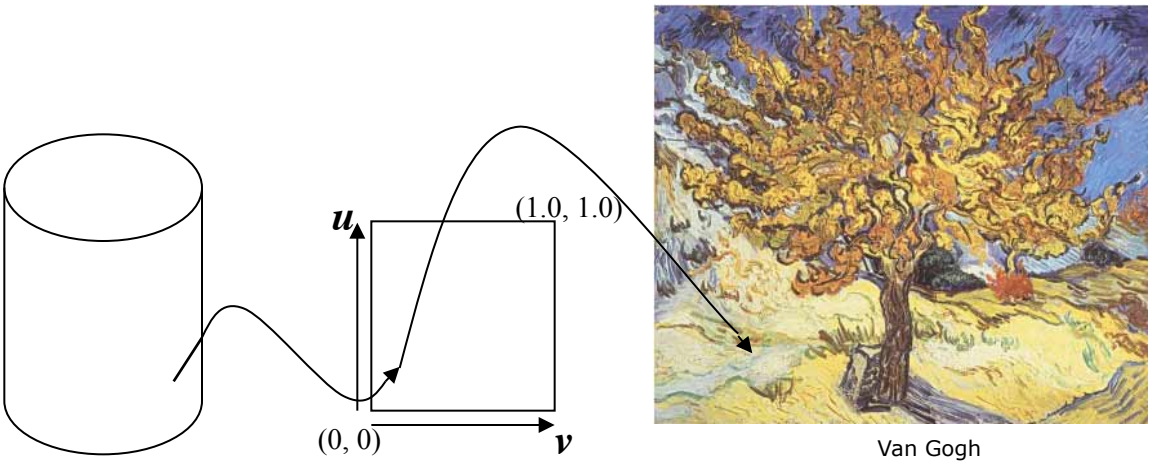
- Why texture map? To hack surface detail
- Expensive solution: add more geometric detail to model
  - + detail incorporated as a part of object
  - – model takes longer to render
  - – model takes up more space in memory
  - – complex detail cannot be reused
- Efficient solution: map a texture onto model
  - + texture maps can be reused
  - + texture maps take up space in memory, but can be shared, and compression and caching techniques can reduce overhead significantly compared to real detail
  - + texture mapping can be done quickly (we'll see how)
  - – texture maps do not affect the geometry of the object
- What kind of detail goes into these maps?
  - – diffuse, ambient and specular colors
  - – specular exponents (shininess)
  - – transparency, reflectivity
  - – fine detail surface normals (bumps)
  - – data to visualize (e.g., temperature, elevation)
  - – projected lighting and shadows
  - – games use "billboards" for distant detail. (sprites are effectively moving billboards)

# Mappings

- A function is a mapping
  - Takes any value in the domain as an input and outputs ("maps it to") one unique value in the co-domain.
- Mappings in "Intersecter":
  - Map screen space points (input) to camera space rays (output)
  - Map camera space rays into world space rays
  - Map world space rays into un-transformed object space for intersecting
- Mapping a texture:
  - Take points on the surface of an object (domain)
  - Return an entry in the texture (co-domain)

# Basic Idea

- Definition: texture mapping is the process of mapping a geometric point to a color in a texture map
- Our goal is to map arbitrary geometry to a texture of any dimension
- This is done in two steps:
  1. map a point on the geometry to a point on the unit square
  2. map the unit square to the texture

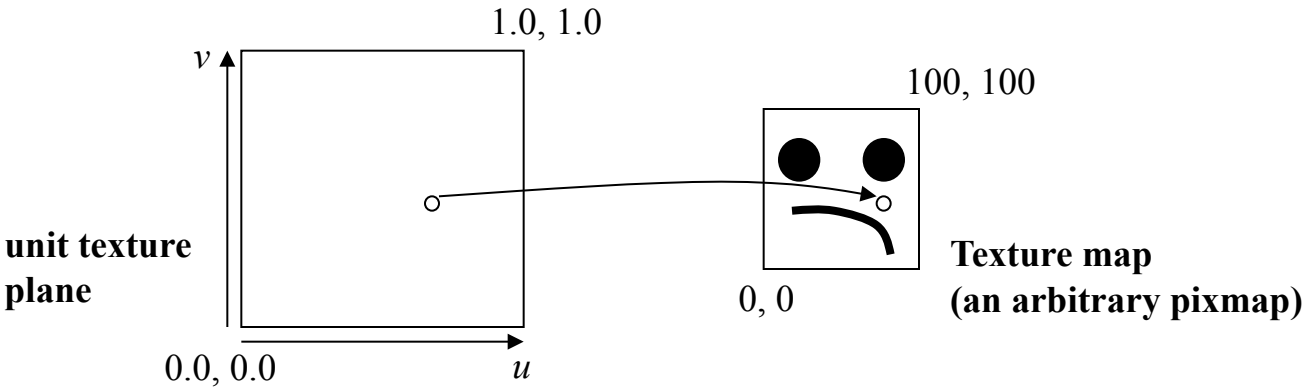



Van Gogh

- Second mapping is easier, so we present it first
- Note: *u,v* space is not related to film plane UV space (sorry!)

# Step 2: Mapping Unit Square to Texture

- A 2D example: mapping from unit *u,v* square to texture



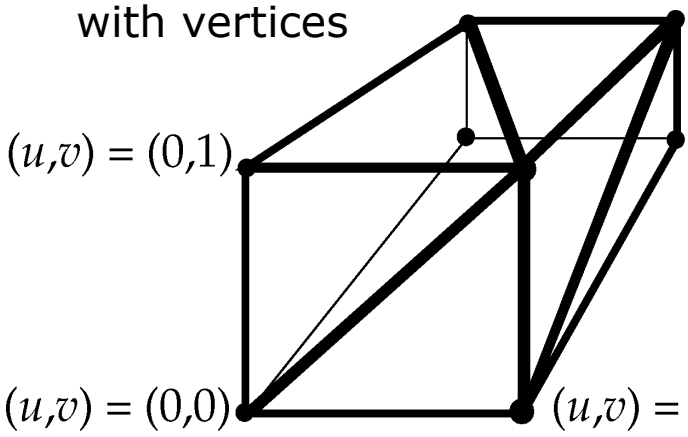

  - Above Example:
    (0.0, 0.0)->(0, 0); (1.0, 1.0)->(100, 100); (0.75, 0.45)->(75, 45)

- In general: for any point *(u, v)* on unit plane, corresponding point in texture space is:
  *(u* * texture width, *v* * texture height)

- Once we have the coordinates, we just need to look up the color of the texture at that location

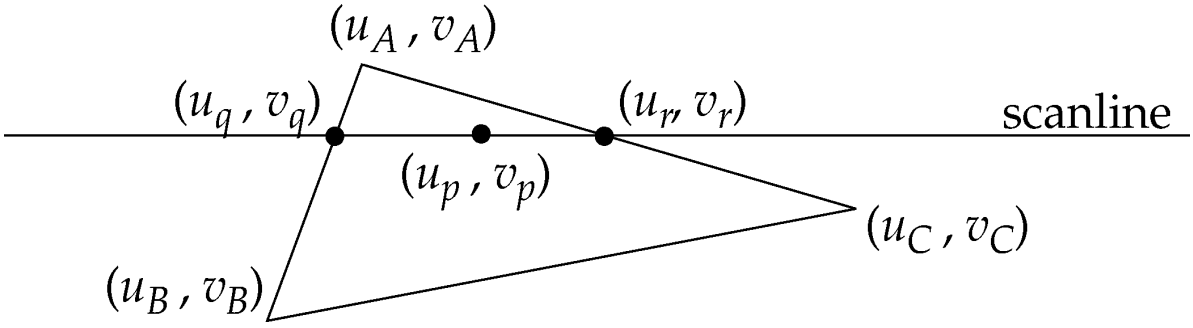
# Mapping Polygons to Texture
## (done in hw)

- Pre-calculate texture coordinates for each facet's vertices during tessellation - store them with vertices



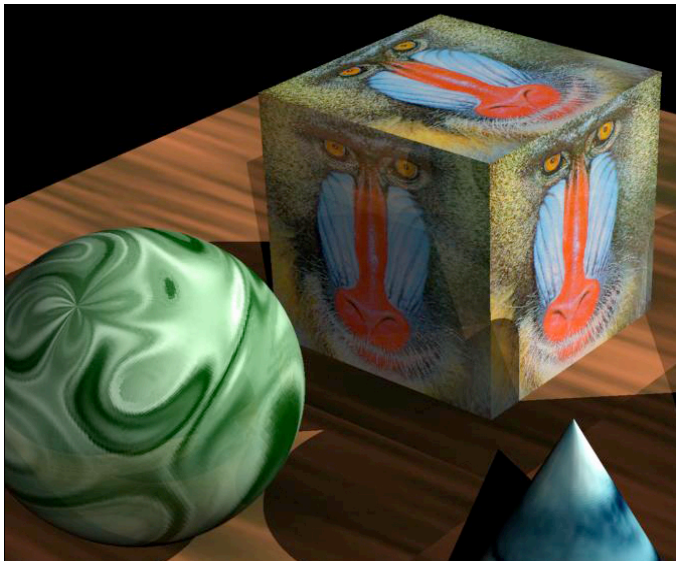

Note: vertices will have different tex coordinates on each face

- Interpolate *uv* coordinates linearly across triangles as part of Gouraud shading

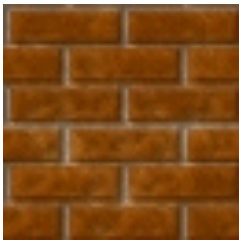


# Step 1: Map from point on object to (u, v) square

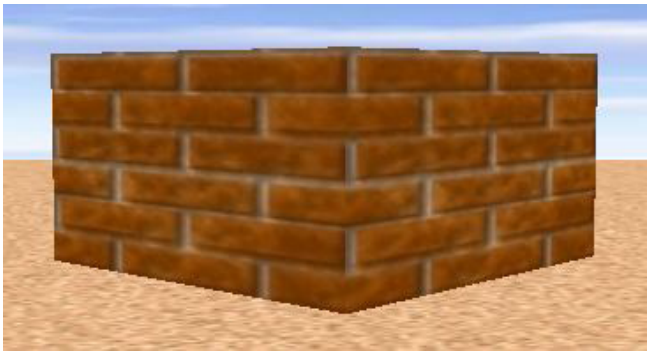
- Three easy cases: planes, cylinders, and spheres
  - Can cause unwanted texture scaling
  - Texture filtering is an option in most graphics libraries
  - OpenGL allows you to choose filtering method. (GL_NEAREST, GL_LINEAR, etc…)



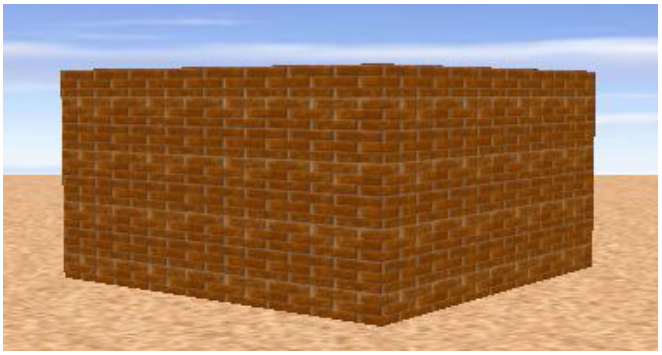
# Tiling: Motivation



•Texture



•Without Tiling



•With Tiling

- Tiling allows us to scale repetitive textures to make texture elements just the right size.

# Tiling Infinite Planes

- How to map from a point on an infinite plane to a point on the unit plane?
- Tiling: use decimal portion of *x* and *z* coordinates to compute 2D *(u, v)* coordinates

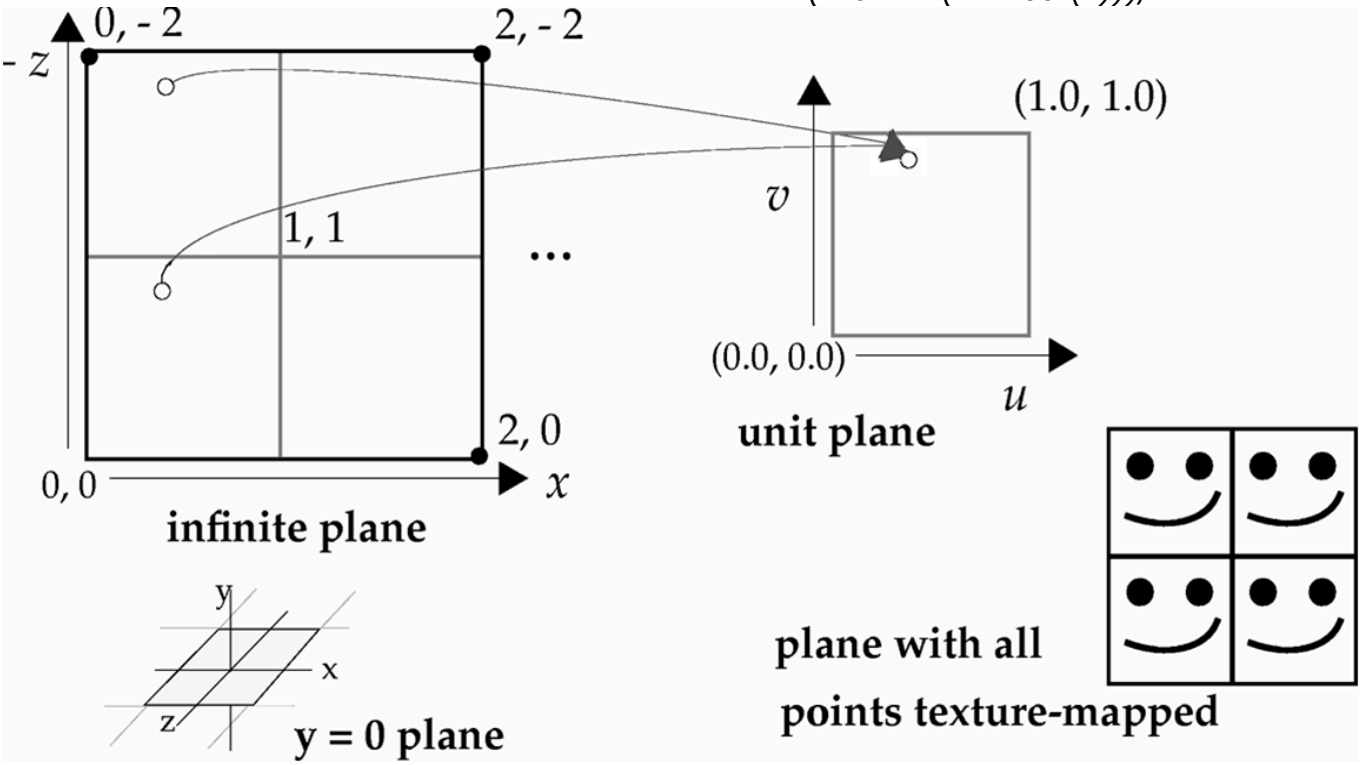
  $u = x - \text{floor}(x)$

  $v = z - \text{floor}(z)$

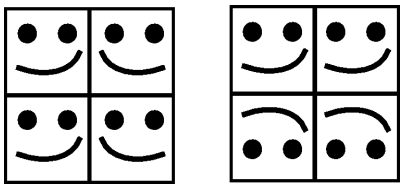
To map to tiles of size nxm:

$u = (x\%n + (x - floor(x)))/n$

$v = (z\%m + (z - floor(z)))/m$

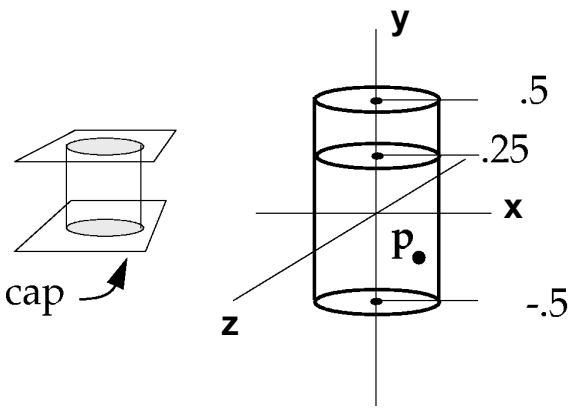



- Mirroring:



# Mapping Cylinders and Cones

- Imagine a standard cylinder or cone as a stack of circles
  - use position of point on perimeter to determine *u*
  - use height of point in stack to determine *v*
  - map top and bottom caps separately, as onto a plane
- The easy part: calculating *v*
  - height of point in object space, which ranges between
    [-.5, .5], gets mapped to *v* range of [0, 1]

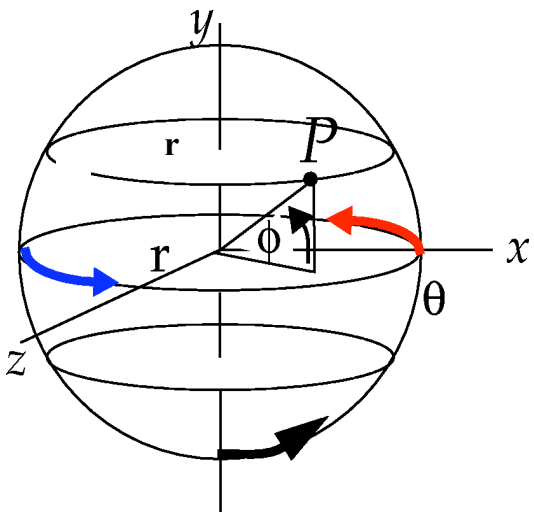


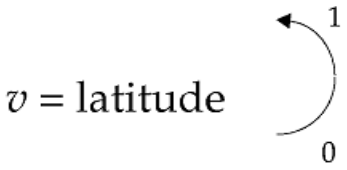

$$-.5 < P_y < .5$$

- Calculating *u*: map points on circular perimeter to *u* values between 0 and 1; 0 radians is 0, 2π radians is 1
- Then $u = \theta/2\pi$
- These mappings are arbitrary: any function mapping the angle around the cylinder (θ) to the range 0 to 1 will work

# Mapping Spheres

- Imagine a sphere as a stack of circles



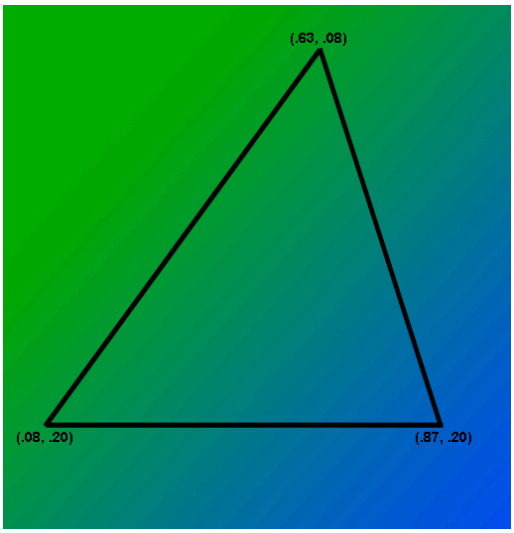

- *P* is a point on the surface of the unit sphere
- Defining *u* – use mapping for circle
  - if *v*=0 or 1, there is a singularity and *u* should equal some specific value (i.e., *u* = 0.5)
  - never really code for these cases because rarely see these values within floating point precision
- Defining *v*

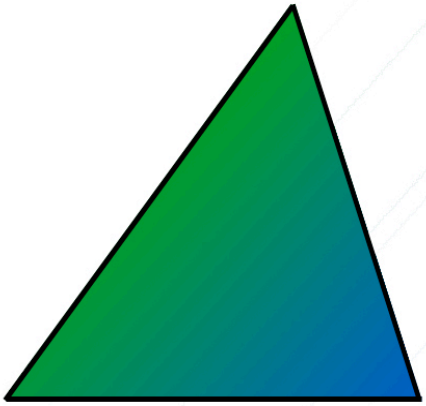
$$\varphi = \sin^{-1}\left(P_y / r\right) \qquad -\frac{\pi}{2} \le \phi < \frac{\pi}{2} \quad , \; r = \text{radius}$$

$$v = \frac{\phi}{\pi} + 0.5$$

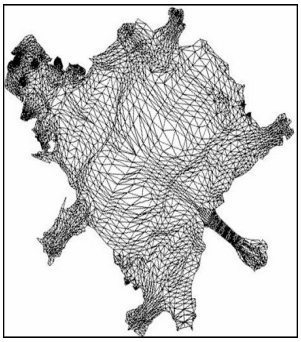
# Mapping Meshes

- Texture Mapping a Triangle:
  - –specify texture coordinates for each vertex based on position within the unit square
  - –pass texture coordinate along with vertex, normal etc., while drawing

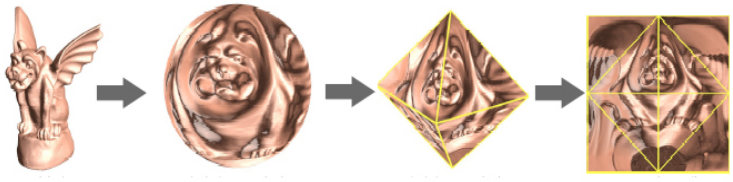


- How do you map an entire mesh?
  No standard method.



- Current Research:
  - – e.g., Spherical Parameterization (E. Praun and H. Hoppe)
    - •No wasted space
    - •Can cause distortions
    - •Hard to work with

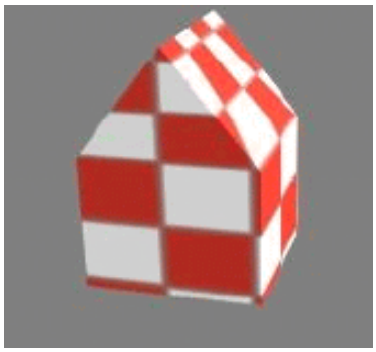


# Texture Mapping Complex Geometries

- Say we have a relatively complex shape: a house
- How should we texture map it?
- We could texture map each polygonal face of the house (we know how to do this already as they are planar).

- This causes discontinuities at edges of polygons

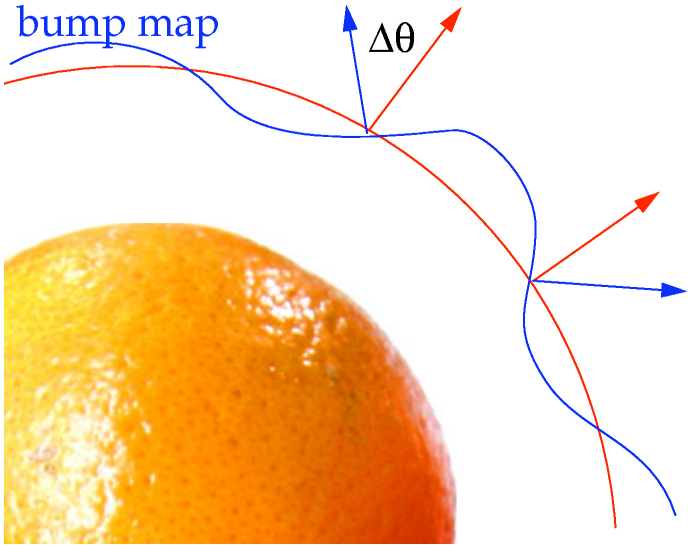
- Intuitive approach: reduce to a solved problem. Pretend the house is a sphere for texture-mapping purposes



# Variation 1: Bump Mapping

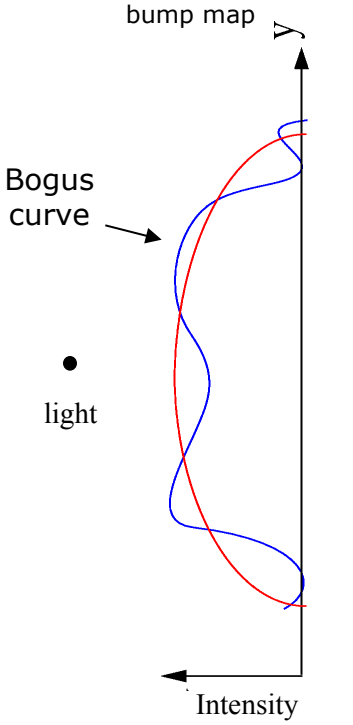
- Texture mapping a rough surface onto an object doesn't look right - illumination is all wrong
- Blinn's hack: use an array of values to perturb surface normals (calculate gradient and add it to the normal)
- Evaluate illumination equation with perturbed normals
- Effect is convincing across surface, but silhouette edges still appear unperturbed
- Consider an orange:

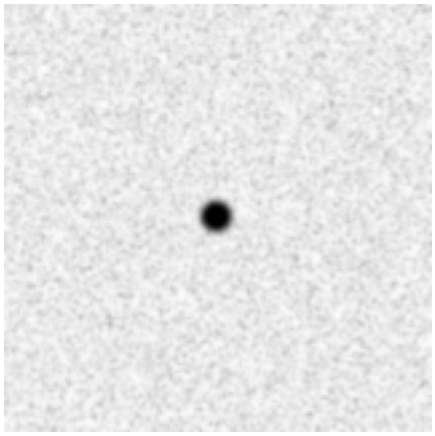
nVidia




Not actual bump map for orange, just an abstraction

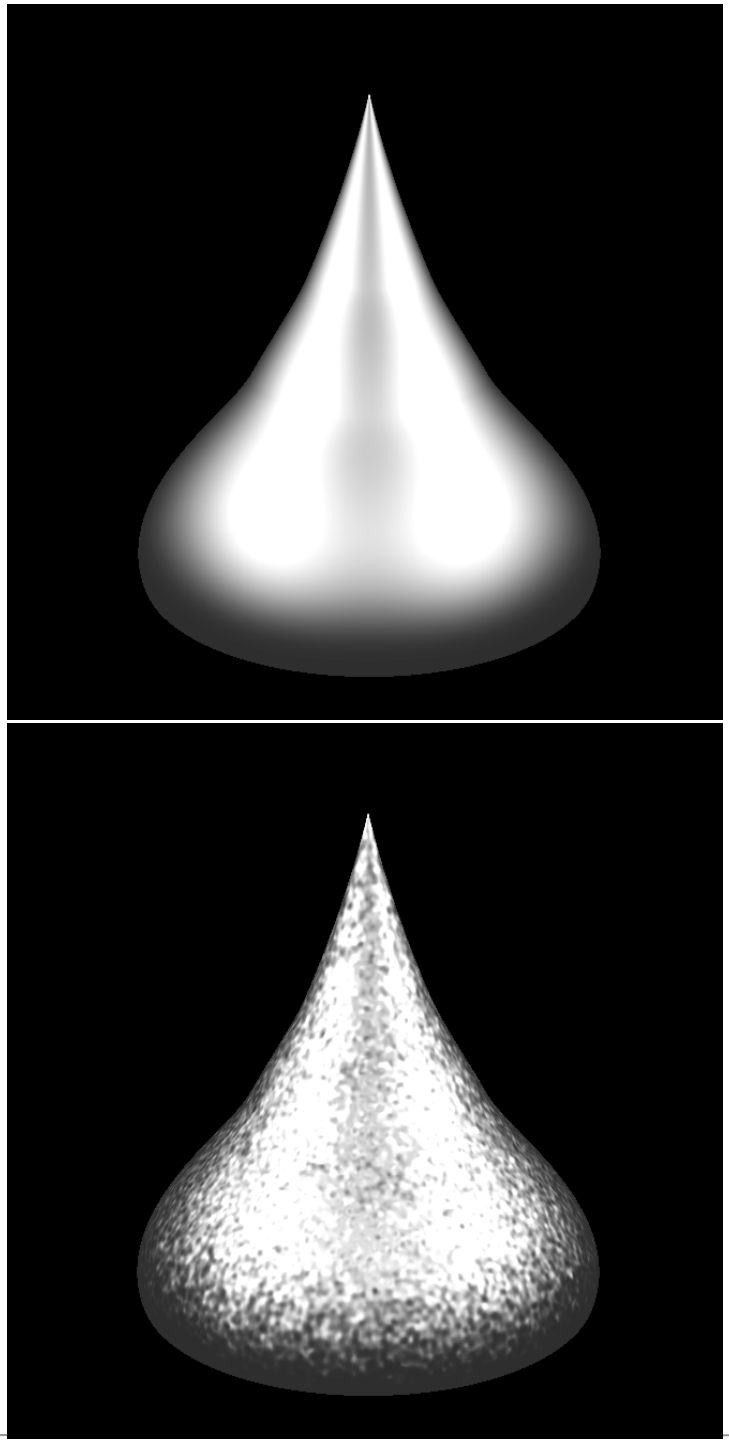
Red line shows standard diffuse lighting, blue line shows theoretical effect of bump map





Wikipedia.org

# Bump Mapping Example



# Variation 2: Normal Mapping

- Recently bump mapping is being replaced by normal mapping for many applications
  - -similar idea with much better results
  - -normal mapping handles drastic variation in normals better than bump mapping
  - -lighting looks better but silhouetting is still a problem

- *Bump mapping* uses a single-channel (grayscale) map to perturb existing normals on the surface; *normal mapping* replaces a surface's existing normals altogether with normal values stored in a map
  - -normal maps give complete control of object normals

- Normal mapping uses a multichannel map (image) with the red, green and blue values of each pixel corresponding to the x,y and z components of the normal vector
  - -x,y,z components of the mapped normals are usually in object-space but other spaces are sometimes used
  - -level of detail of output renders is limited by resolution of normal maps instead of by the number of polygons in rendered meshes!
  - -runtime surface normal determination is trivial—it's just a look-up—making this technique particularly useful for real-time applications

- Limitations:
  - -silhouettes do not reflect added detail (as in bump mapping)
  - -though runtime is easy, initial creation of normal maps is not straight-forward (this is a big deal--more later)

# Normal Mapping

- Normal maps are most useful for adding detail to simple meshes, allowing low-res polygon models to *appear* to be much higher-resolution

Image courtesy of www.anticz.com

render showing actual, simple underlying geometry

when a normal map is applied, there appears to be much more to the model than there is

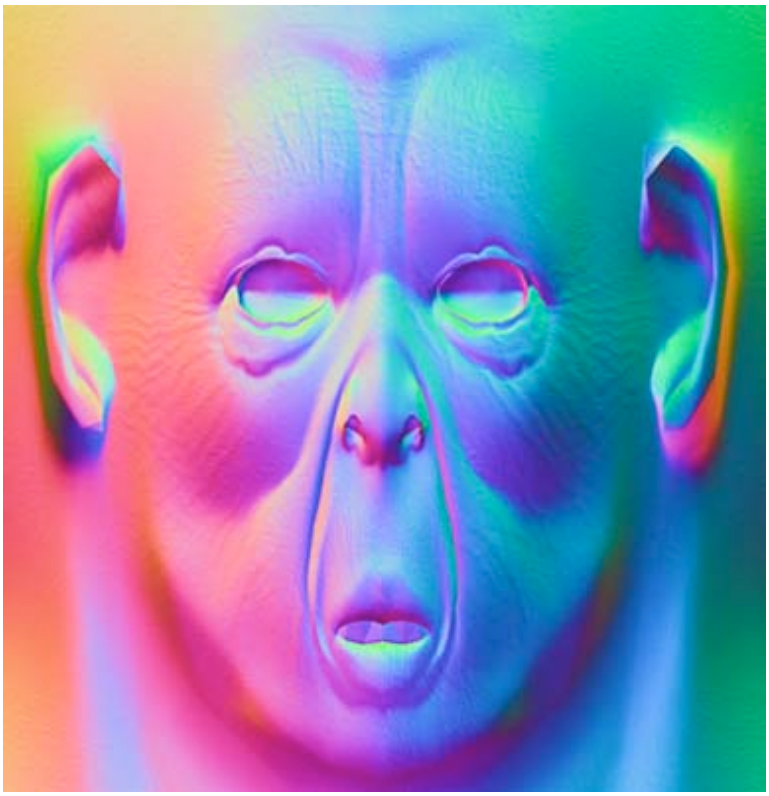
- This approach can be extended to almost completely alter the perceived geometry of a model


*Left half* – shading

*Right half* – normal map, environment map

# Normal Map Creation

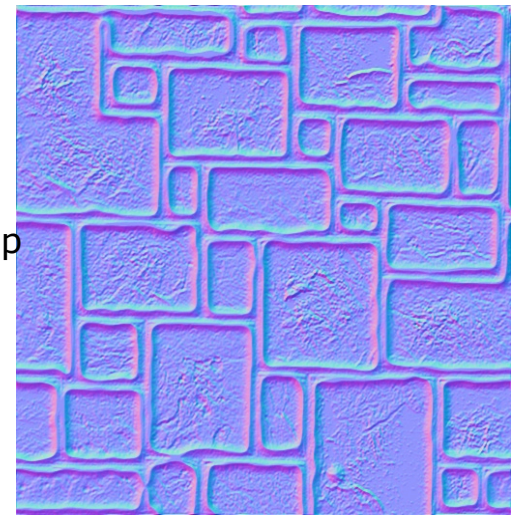
- As mentioned, the creation of meaningful normal maps is not simple
  -unlike bump maps, normal maps can not simply be painted by hand—color in a normal map is constantly varied and r,g,b values are spatially meaningful



www3.telus.net

Normal map for a human face. Try painting that in Photoshop! Note how red, green and blue correspond to the direction a normal at each pixel would face

Normal map for brick street



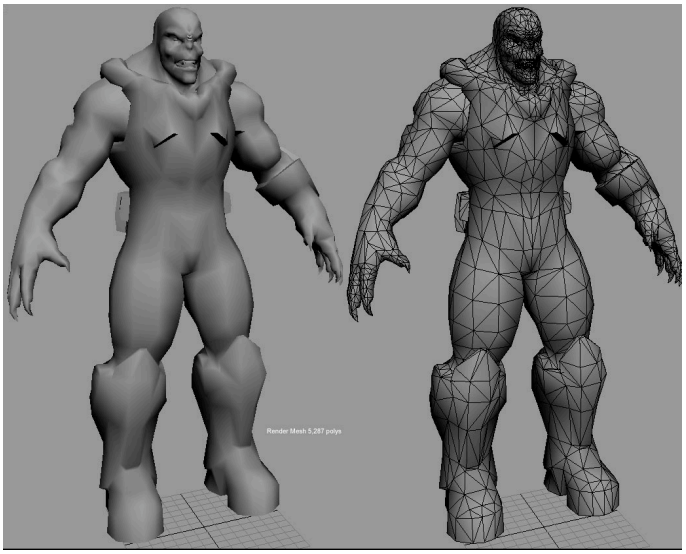
www.bitmanagement.com

- So how are normal maps created?

# Normal Map Creation 2

- Normal maps must be generated by specialized software, such as Pixologic's Zbrush (check out www.pixologic.com)
  - -a high resolution model is created by the artist and its normals are used to generate maps for lower-res versions of the model

Low-res mesh (~5,000 polygons)

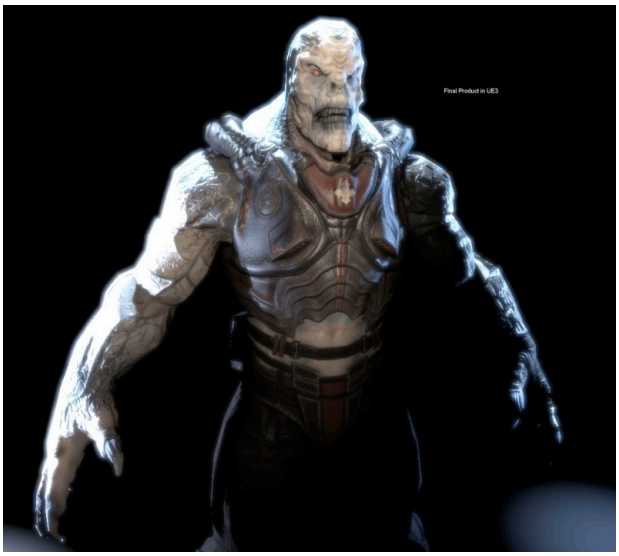


High-resolution model (2million polygons!) —this is the source of the normal map

*3D geometry*

*normal map*

So many polygons that you can't even see the edges of the triangles!



Final rendering with normal map applied to low-res mesh (and lighting and color maps). This is only 5,000 triangles!

Images courtesy of www.unrealtechnology.com

# Summary

- Texture mapping: definition and motivation

- Mapping mechanism: two steps

- Step 2: Map unit square to texture
  - interpolation in hw during scan

- Step 1: Map object to unit square
  - tiling infinite planes
  - mapping cylinders and cones
  - mapping spheres
  - mapping meshes
  - mapping complex geometries

- Bump Mapping
- Normal Mapping