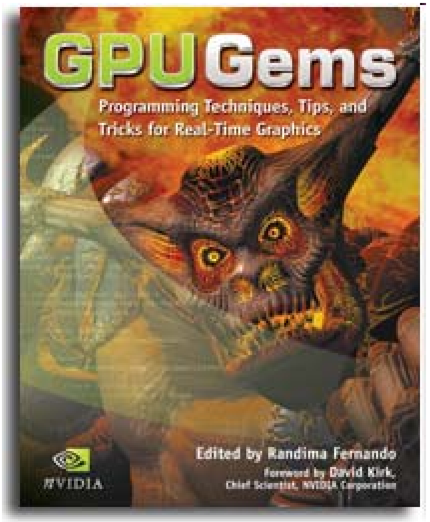
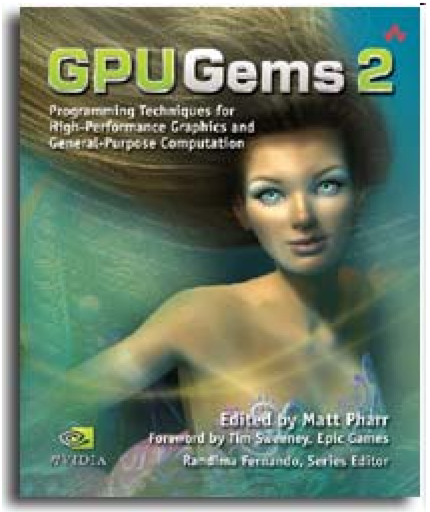
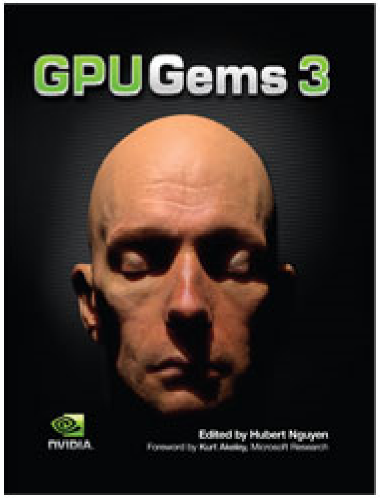
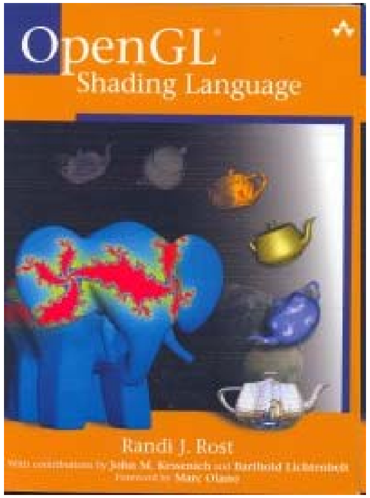
Introduction to Programming Mapping Techniques On The GPU









Cliff Lindsay

Ph.D. Student, C.S. WPI

http://users.wpi.edu/~clindsay

[images courtesy of Nvidia and Addision-Wesley]

Motivation

Why do we need and want mapping?

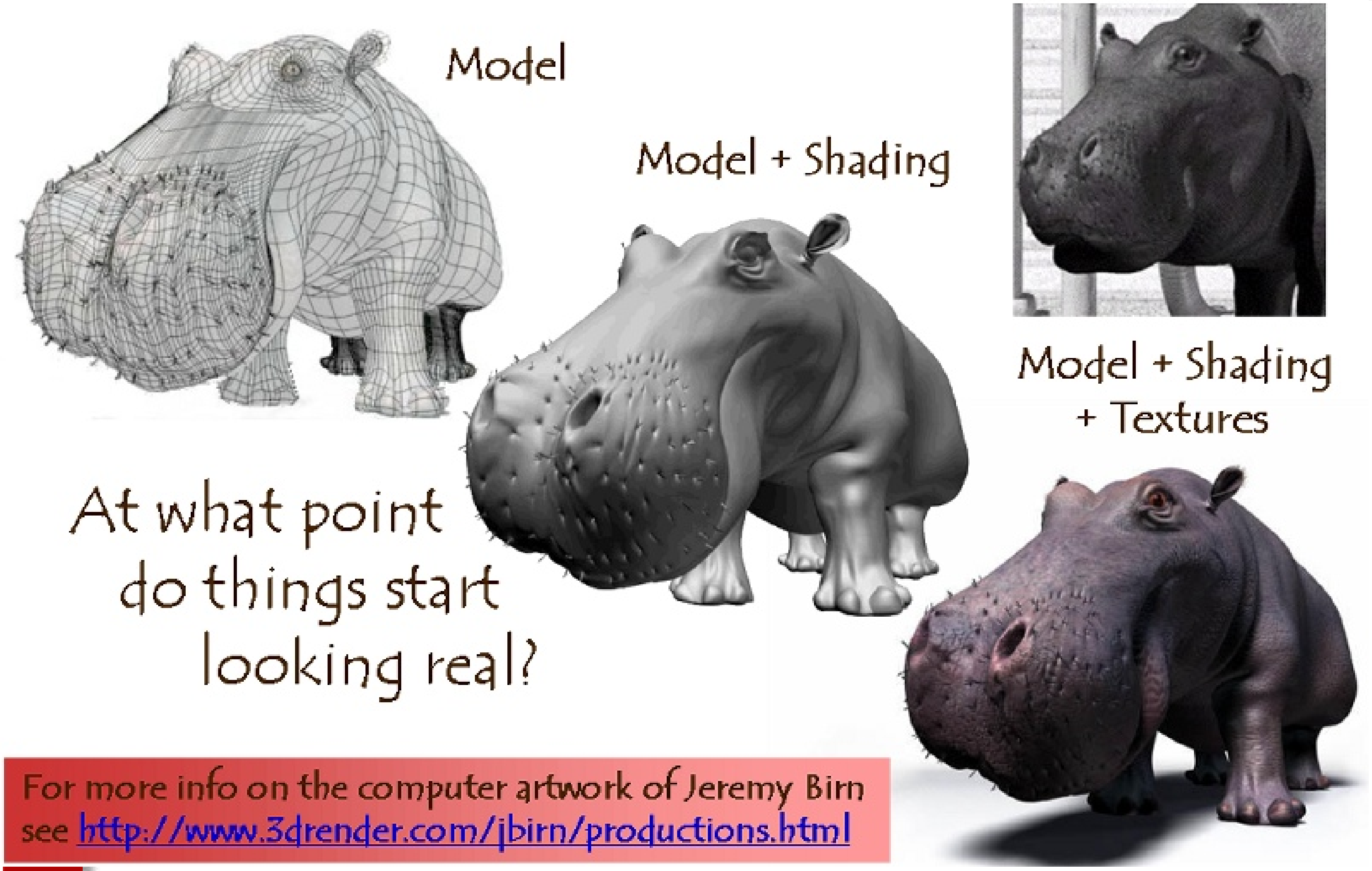
- Realism
- Ease of Capture vs. Manual Creation
- GPUs are Texture Optimized (Texture = Efficienct Storage)

Solid Color Metal

Metal Using Mapping Techniques

[Images from Pixar]

Quest for Visual Realism



At what point
do things start
looking real?

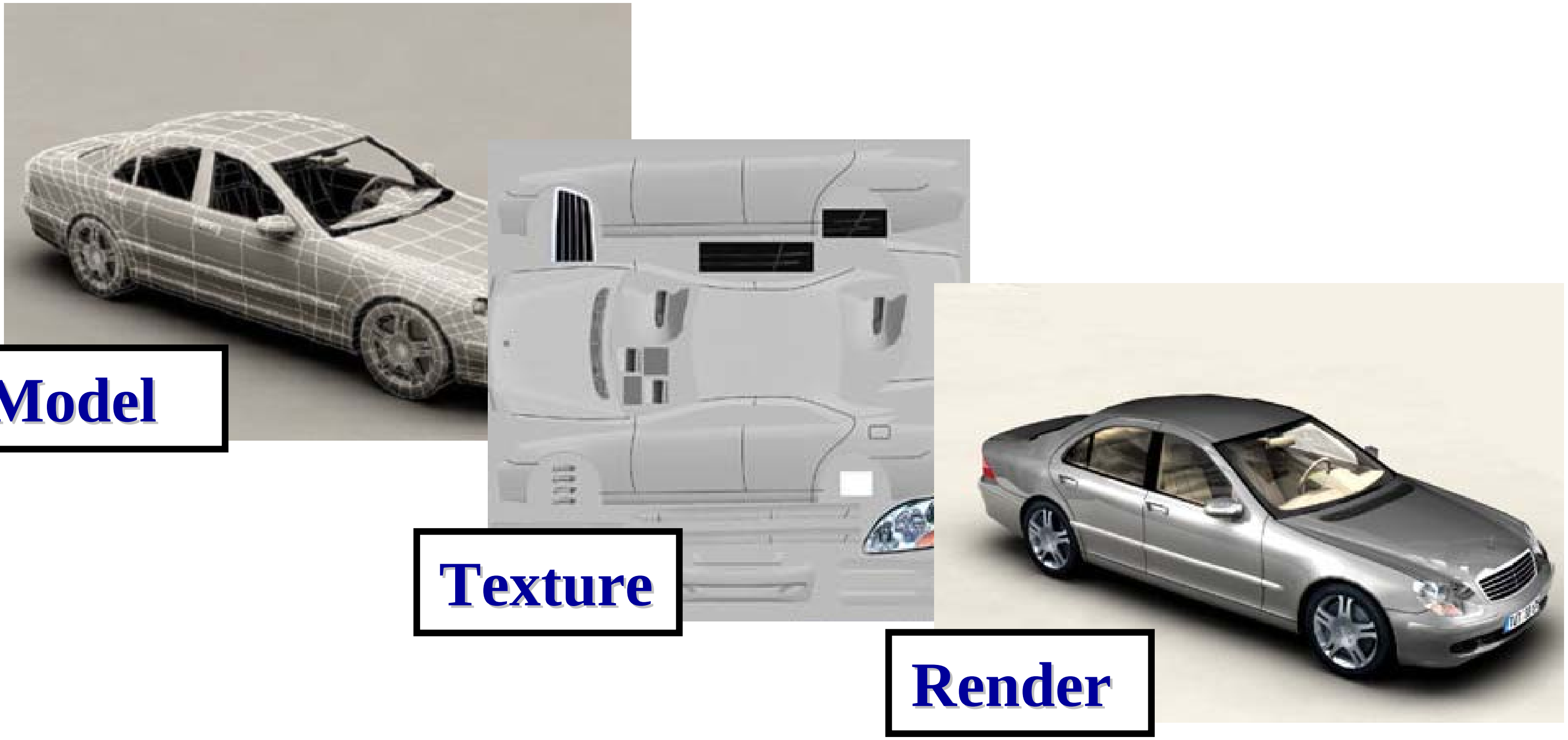
For more info on the computer artwork of Jeremy Birn
see http://www.3drender.com/jbirn/productions.html

Talk Overview

- Review Basic Texturing
- Environment Mapping
- Bump Mapping
- Displacement Mapping

Texture Mapping

Main Idea: **Use an image to apply color to the pixels Produce by geometry of an object.**[Catmull 74]



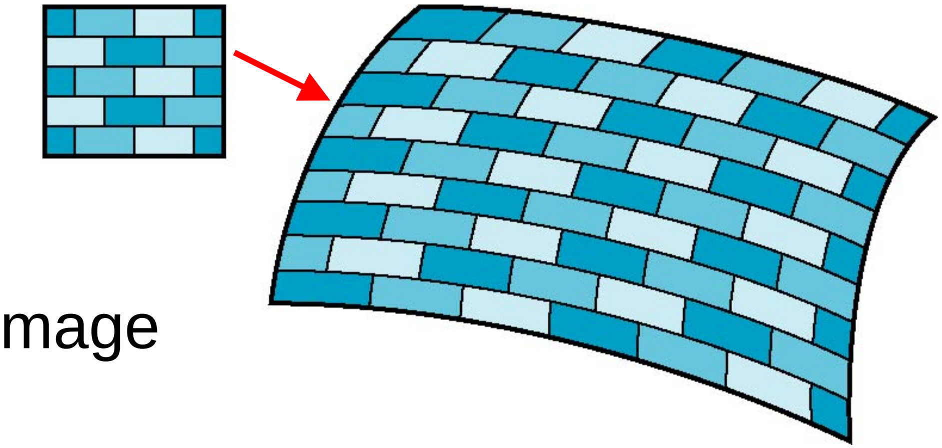
Model

Texture

Render

Is it simple?

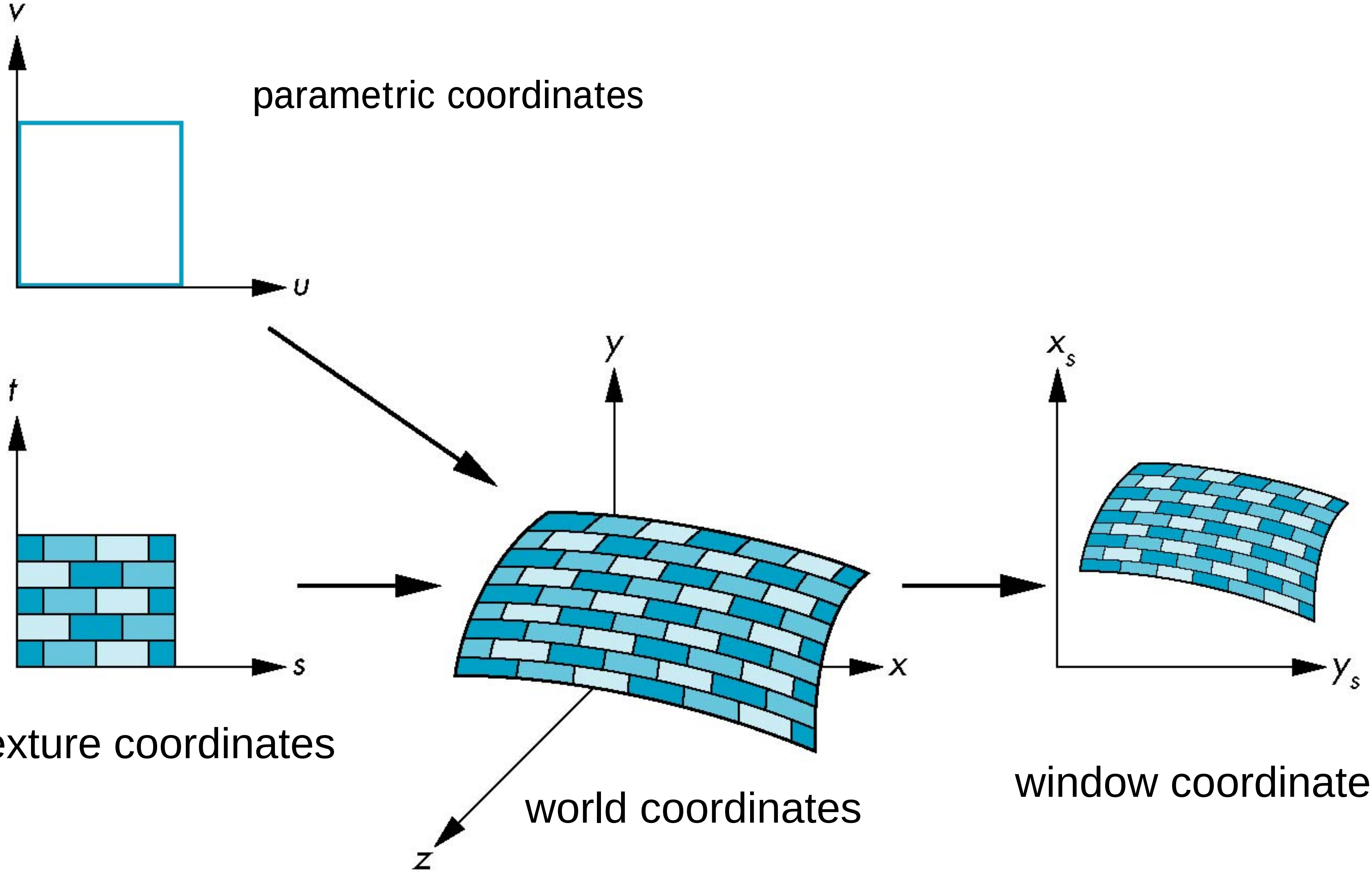
- Idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved



2D image

3D surface

Texture Mapping



parametric coordinates

texture coordinates

world coordinates

window coordinates

Mapping Functions

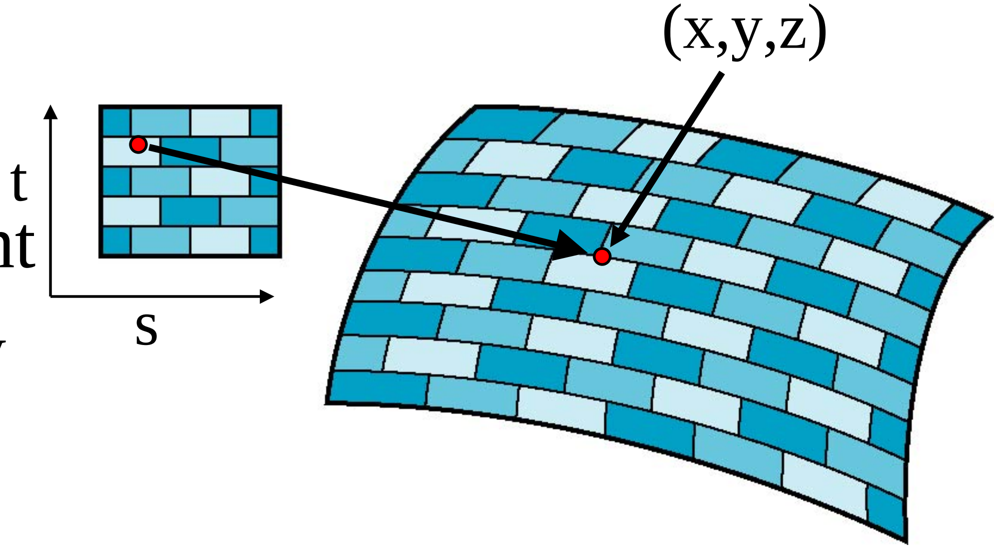
- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

$$x = x(s,t)$$

$$y = y(s,t)$$

$$z = z(s,t)$$

- But we really want to go the other way



Backward Mapping

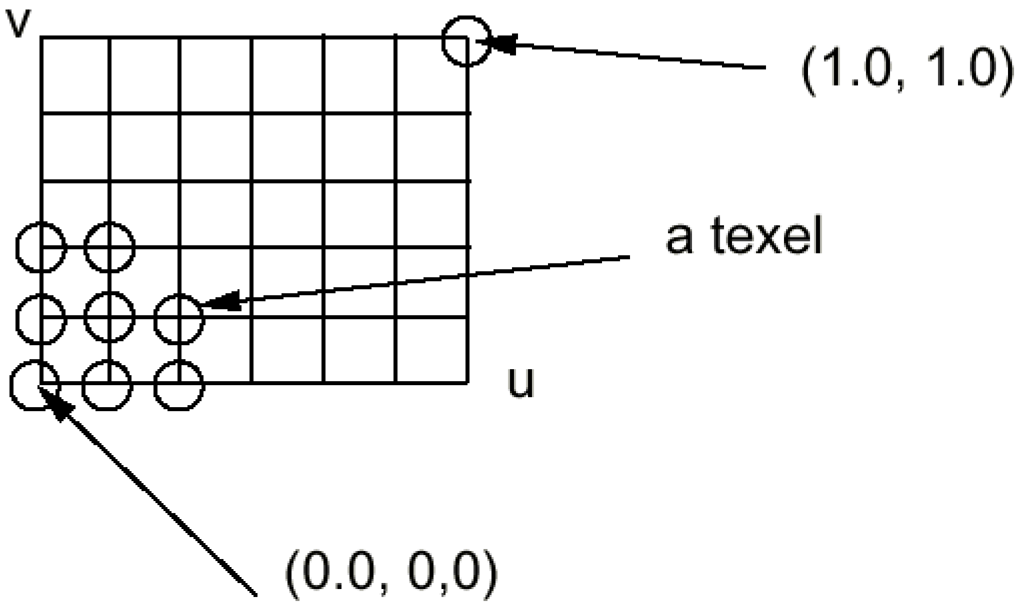
- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form

  $s = s(x,y,z)$

  $t = t(x,y,z)$
- Such functions are difficult to find in general

Texture and Texel

- Each Pixel in a Texture map = Texel
- Each Texel has (u,v) 2D Texture Coordinate
- Range of (u,v) is [0.0,1.0] (normalized)



Are there Issues?

2 Problems:

- •Which Texel should we use?
- •Where Do We Put Texel?

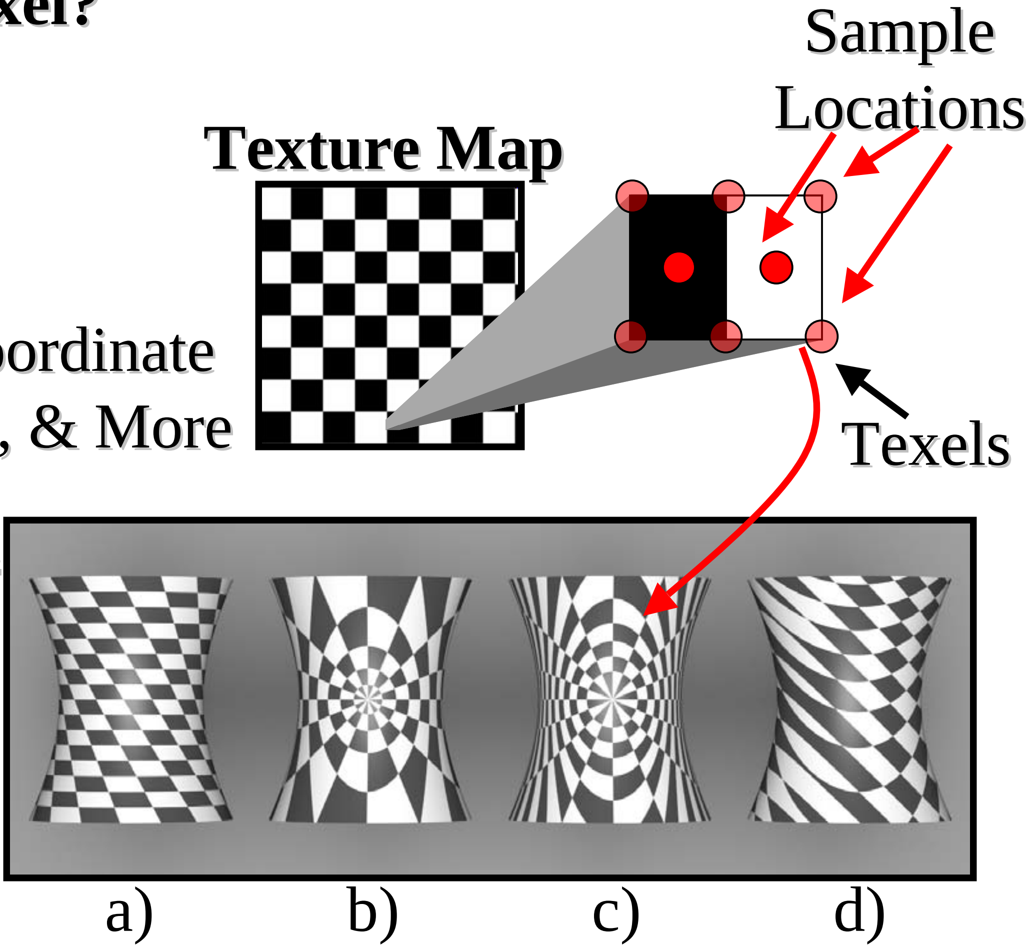
2 Solutions:

Sampling & Filtering

- •Map >1 Texel to 1 Coordinate
- •Nearest, Interpolation, & More

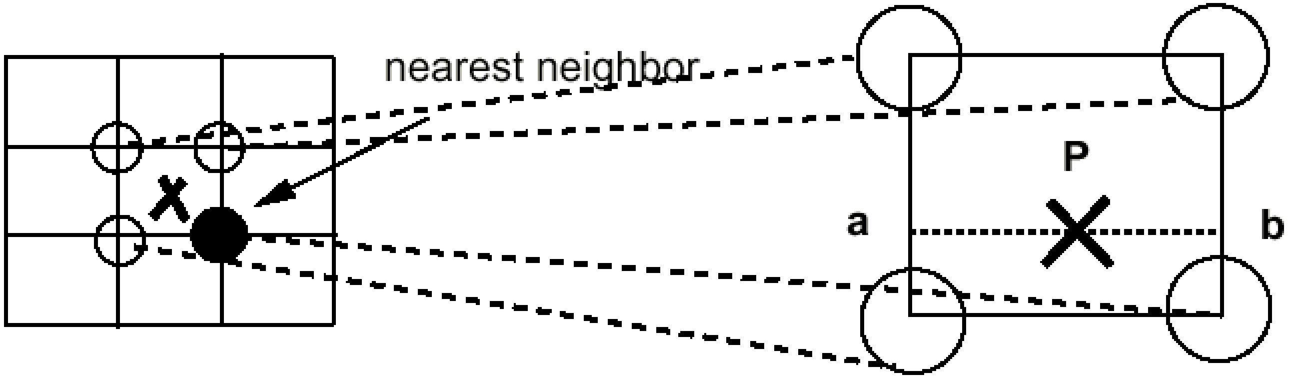
Coordinate Generation

a) UV (most common)
b) Spherical
c) Cylindrical
d) Planar



(u,v) tuple

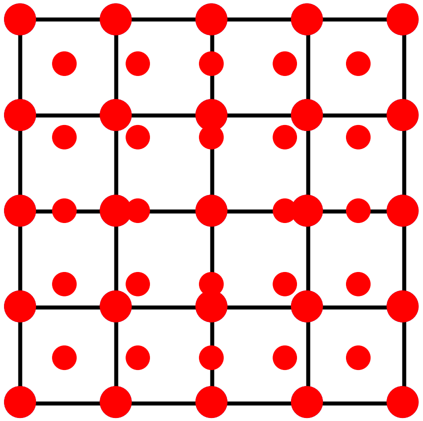
- For any (u,v) in the range of (0-1, 0-1) *multiplied by texture image width and height*, we can find the corresponding value in the texture map



How to get F(u,v)?

- We are given a discrete set of values:
  - **F**[i,j] for i=0,...,N, j=0,...,M
- Nearest neighbor:
  - **F**(u,v) = **F**[round(N*u), round(M*v)]
- Linear Interpolation:
  - i = floor(N*u), j = floor(M*v)
  - interpolate from **F**[i,j], **F**[i+1,j], **F**[i,j+1], **F**[i+1,j]
- Filtering in general !

Interpolation



Nearest neighbor

Linear Interpolation

Applying Our Mapping knowledge

Further Realism Improvements:

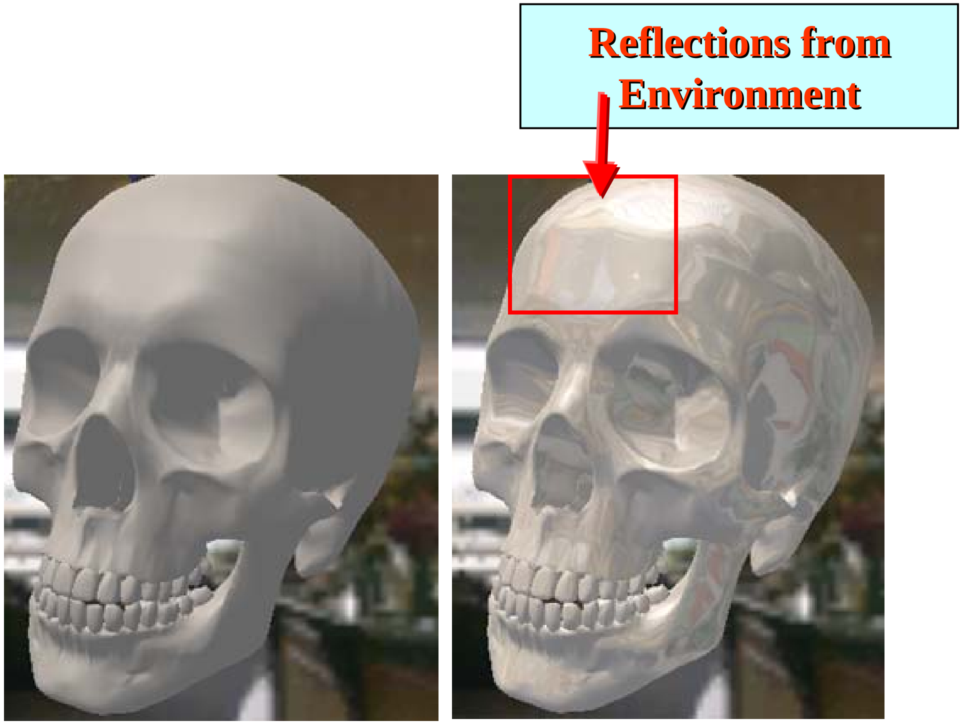
- Environment Mapping
- Bump Mapping
- Displacement Mapping
- Illumination Mapping & Others?

Environment Mapping

<u>Main idea</u>: "Environment Maps are textures that describe, for all directions, the incoming or out going light at a point in space."[Real Time Shading, pg. 49]"

<u>Three main types</u>:

- Cube Mapping
- Sphere mapping
- Paraboloid Mapping



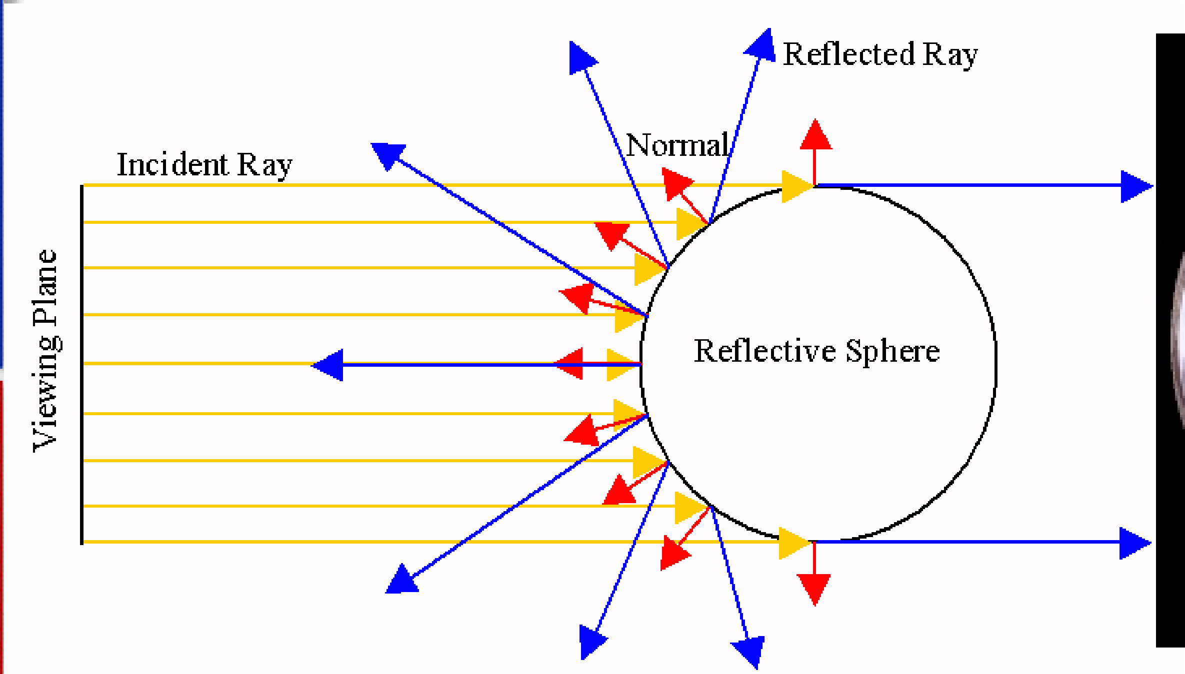
No Map applied **Map Applied**

[Images courtesy of Microsoft, msdn.microsoft.com]

Environment Mapping

Sphere Mapping

- Generated from photographing a reflective sphere
- Captures whole environment



Sphere Texture Map

[Diagram and Sphere Map image of a Cafe in Palo Alto, CA, Heidrich]

Environment Mapping

Cons :

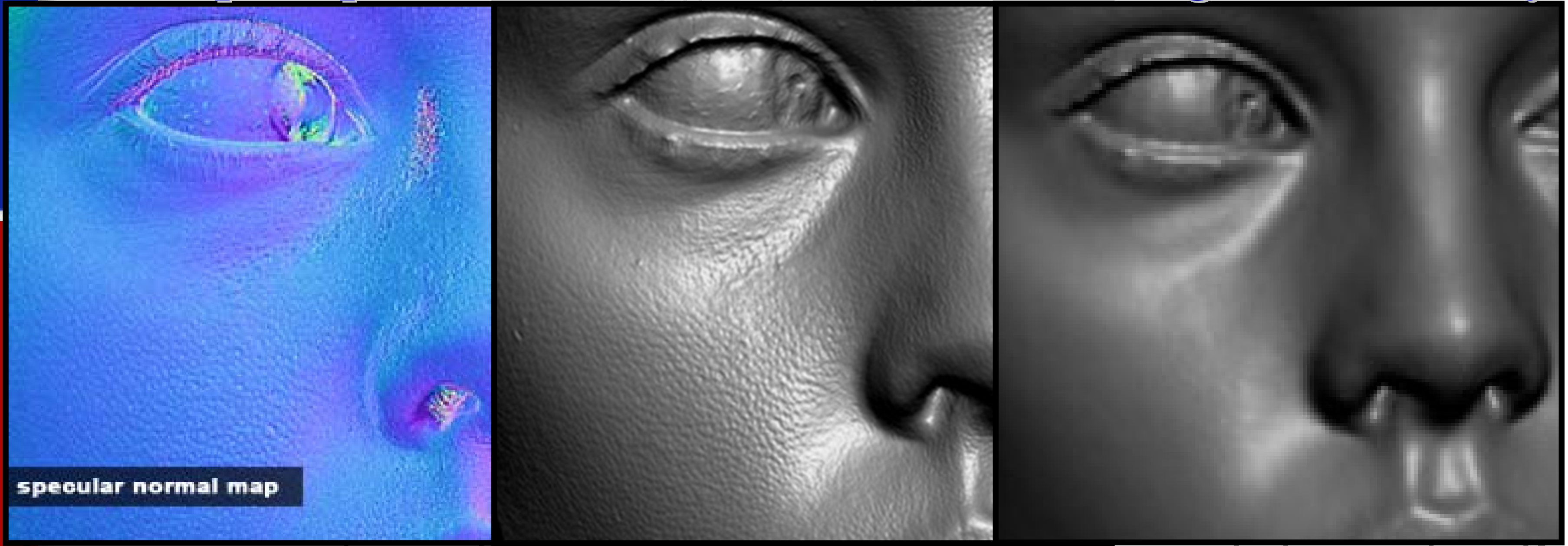
- Sphere maps have a singularity of the parameterization of this method, we must fix viewing direction, view-dependent (meaning if you want to change the viewers direction you have to regenerate the Sphere map).
- Paraboloid maps requires 2 passes

Pros:

- Better sampling of the texture environment for Paraboloid mapping, view-independent,
- Cube maps can be fast if implemented in hardware (real-time generation), view independent,

Bump Mapping

Main idea: "Combines per-fragment lighting with surface normal perturbations supplied by a texture, in order to simulate light interactions on a bumpy surface."[Cg Tutorial, pg 199]



Bump Map

Geometry W/ New Normals

Original Geometry

[Hi-Res. Face Scanning for "Digital Emily", *Image Metrics & USC Institute for Creative TechnologiesGraphics Lab*]

Bump Mapping

$$P'(u,v) = P(u,v) + \vec{N}(u,v)F(u,v)\,*$$

- P = original Surface location/height
- N = Surface Normal
- F = Displacement Function
- P' = New Surface location/height

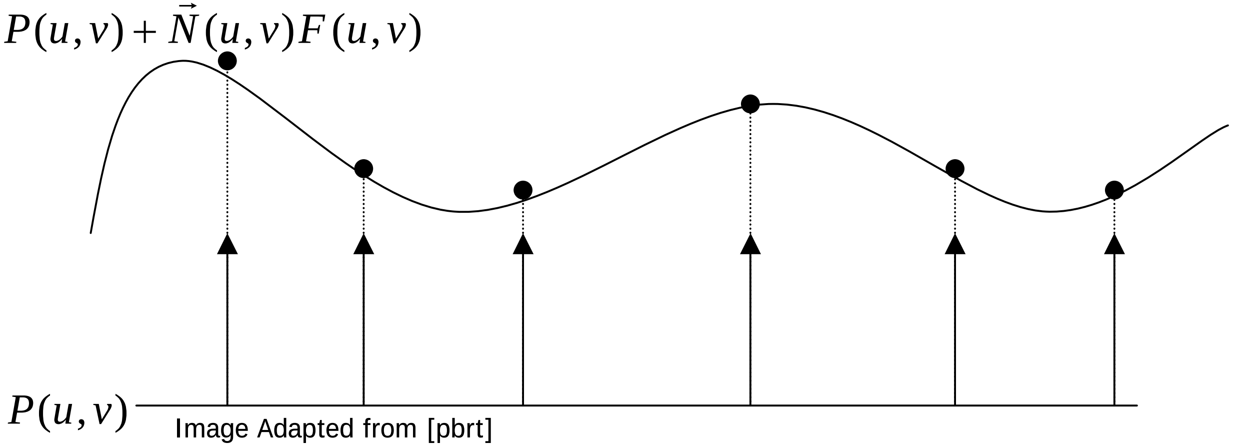


Image Adapted from [pbrt]

* Assumes $\vec{N}$ is normalized.

Bump Mapping

Bump Map

- The new Normal N’ for P’ can be calculated from the cross product of it’s partial derivatives[Blinn 78].

Differential Math!!!

$$\vec{N}' = \frac{\partial P'}{\partial u}\times\frac{\partial P'}{\partial v} \approx \vec{N} + \frac{\partial F}{\partial u}\left(\vec{N}\times\frac{\partial P}{\partial u}\right) + \frac{\partial F}{\partial v}\left(\vec{N}\times\frac{\partial P}{\partial v}\right)$$

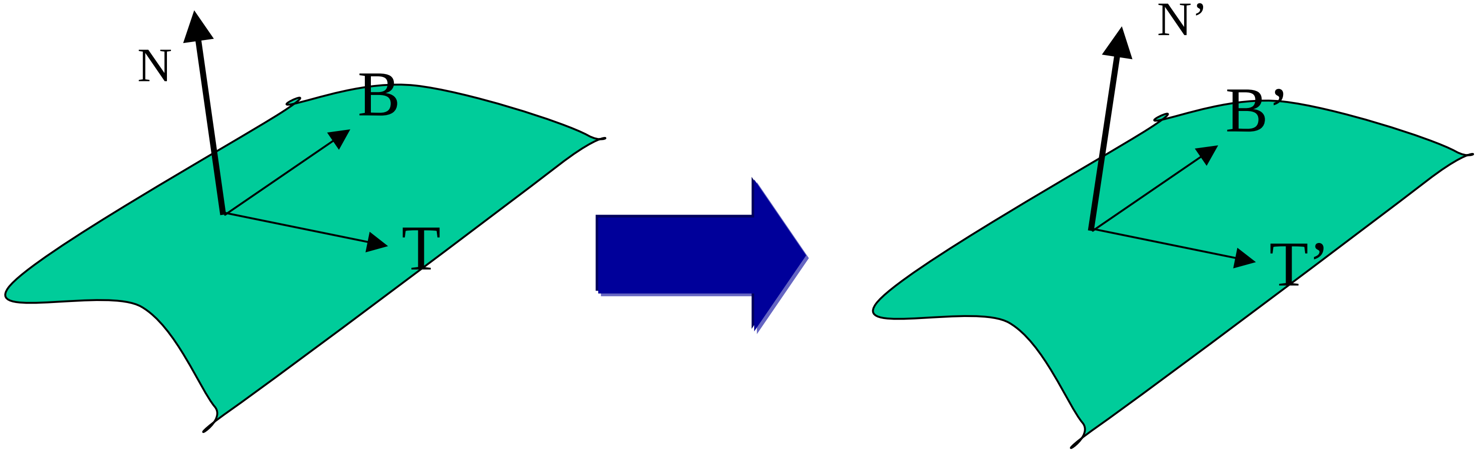
Tangent Space

Calculate Derivatives on the fly is complicated!

Solution:

- We know That our Normal **N = B x T**
- We Want a Normal N’

Determine B’ & T’ for P’ to Get N’



Tangent Space

$$N' = P'_u \; x \; P'_v$$

$$= N + \underbrace{B(NxP_v) - T(NxP_u)}_{D}$$

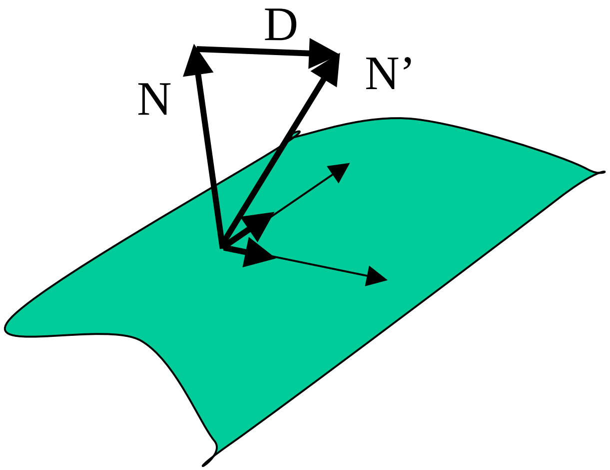
•

•

•

$$= N + D$$

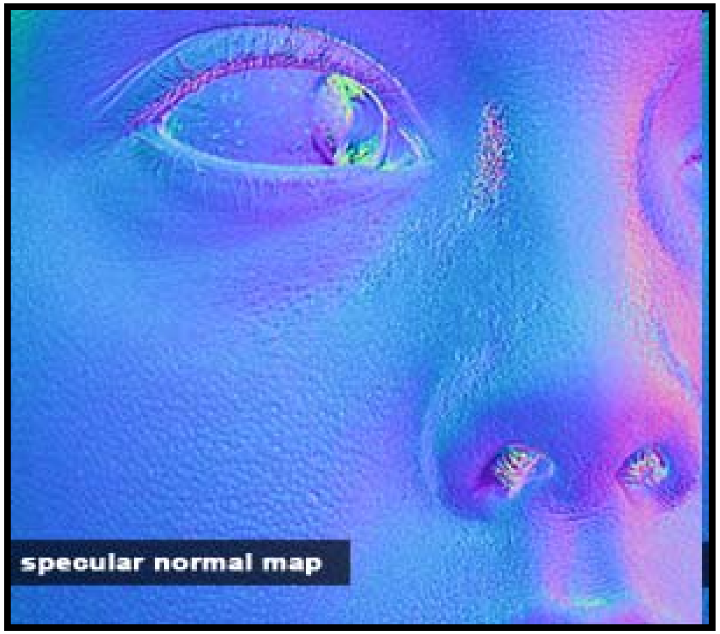
D is just the distance N has to move to be N'



Bump Mapping

Optimizations:

- Info Is Known In Advance
- Pre-process & Lookup At Run-time



Normal Mapping

- Use Texture Map To Store N’
- Look up At Run-time
- Translate & Rotate

Used in Games!

- Hardware Texture Optimized
- Most Work Processed Offline

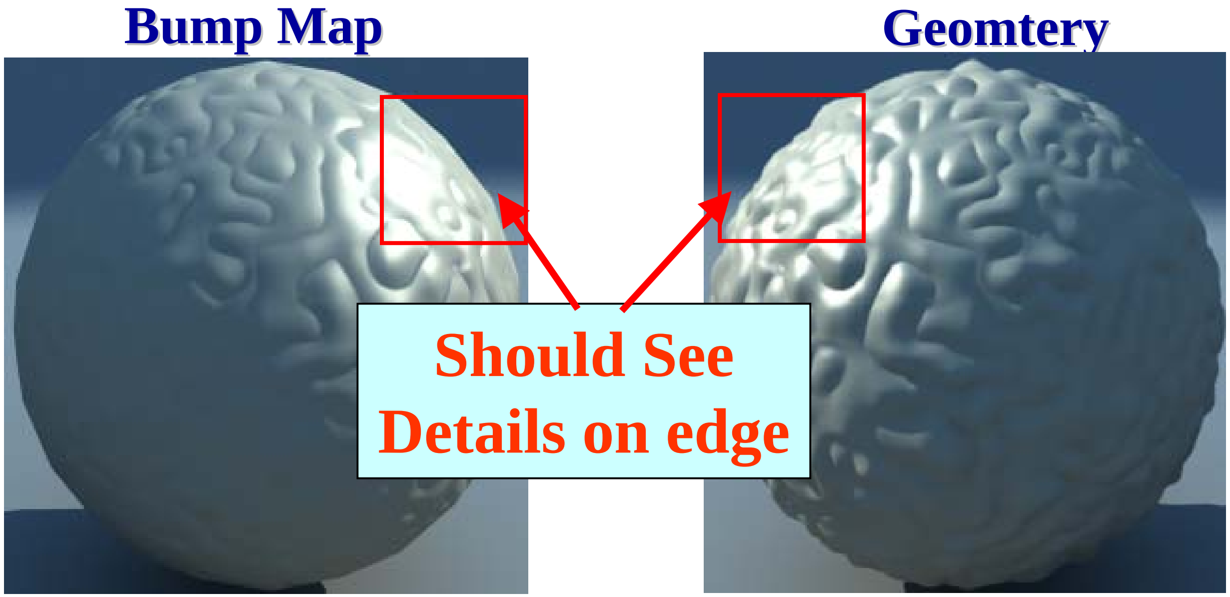
Bump Mapping

Pros:

- Produces the appearance of high detail w/ out cost
- Can be done in hardware

Cons:

- No self shadowing (natively)
- Artifacts on the silhouettes



Bump Map

Geomtery

Should See Details on edge

Displacement Mapping

Main Idea: Use height map texture to displace vertices

- Realistic Perturbations Impossible to Model by Hand
- Actually Displacing Geometry, Not Normals
- No Bump Map Artifacts On Edges

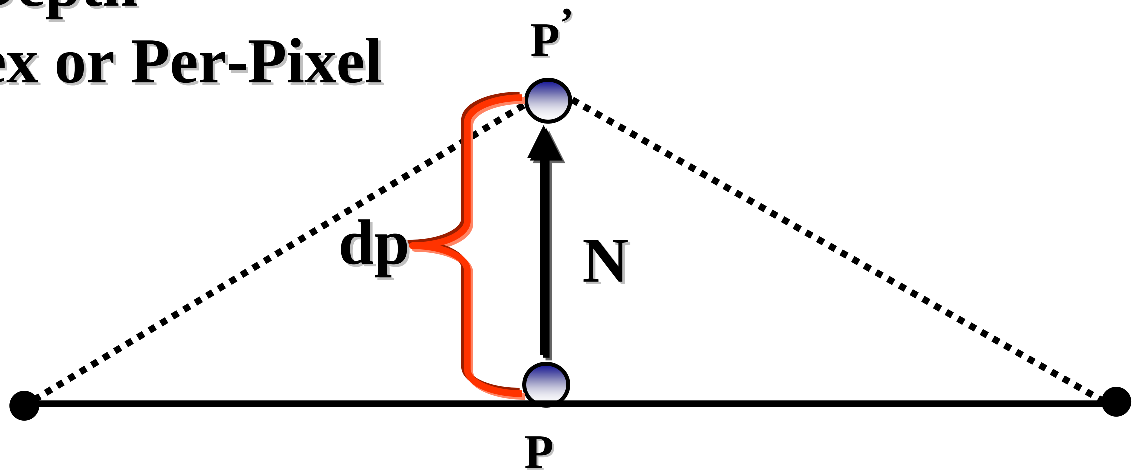
With Displacement

Without Displacement

GPU Gems 2: Ch 18, Using Vertex Texture Displacement for Realistic Water Rendering, Screen Captures of *Pacific Fighter* by Ubisoft

Displacement Mapping

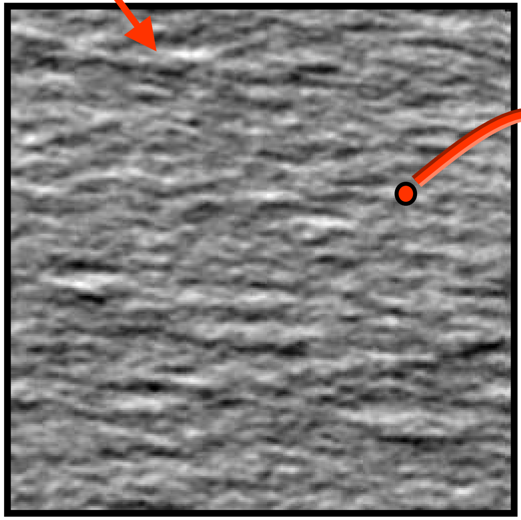
- Gives Geometry Depth
- Can Do Per-Vertex or Per-Pixel



P' = P + (N * dp)

dp = 0.30*R + 0.59*G + 0.11*B

Could be Heightfield



[Diagram Modified From Ozone3d.net]

Displacement Mapping Variant

Parallax Mapping:

- Perturb Texture Coordinates
- Based On Viewer Location
- As If Geometry Was Displaced

With Parallax Mapping **Without Parallax Mapping**

[Comparison from the Irrlicht Engine]

Displacement Mapping

Pros:

- Efficient To Implement On GPU
- Good Results With Little Effort

Cons:

- Valid For Smoothly Varying Height fields
- Doesn’t Account For Occlusions If Done Per-Pixel

Questions?

Thanks to all who's slides were borrowed and/or modified:

- David Lubke, Nvidia
- Ed Angel, University of New Mexico
- Durand & Cutler, MIT
- Juraj Obert, UCF