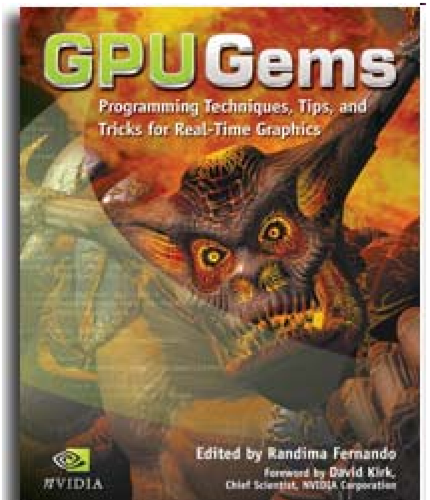
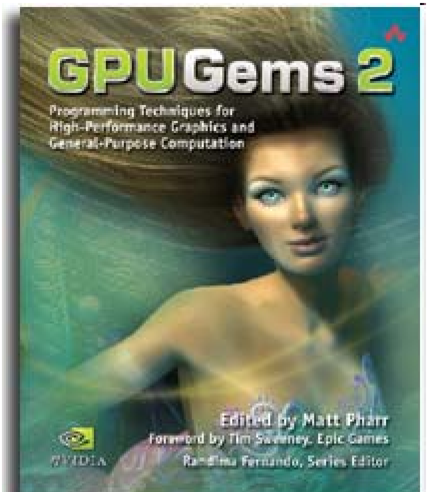
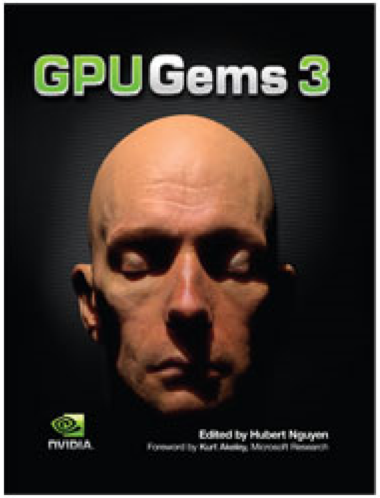
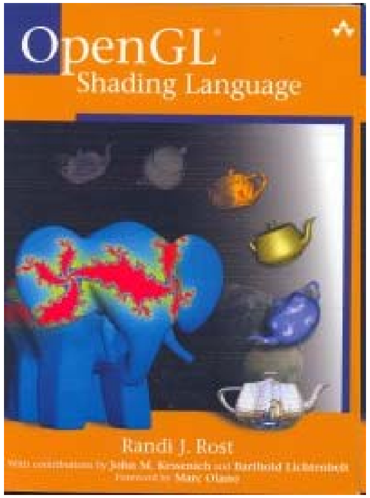
Introduction to Programming Mapping Techniques On The GPU









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?

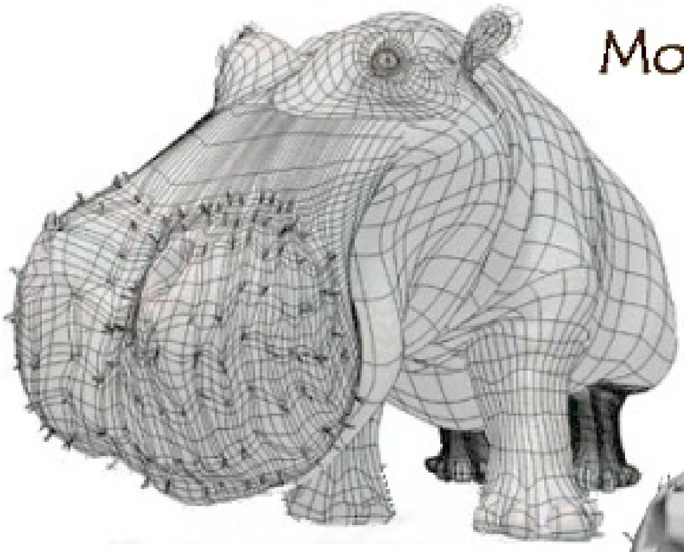
- Realism
- Ease of Capture vs. Manual Creation
- GPUs are Texture Optimized (Texture = Efficienct Storage)

Solid Color Metal

Metal Using
Mapping Techniques

[Images from Pixar]

Quest for Visual Realism



Model

Model + Shading

Model + Shading + Textures

At what point do things start looking real?

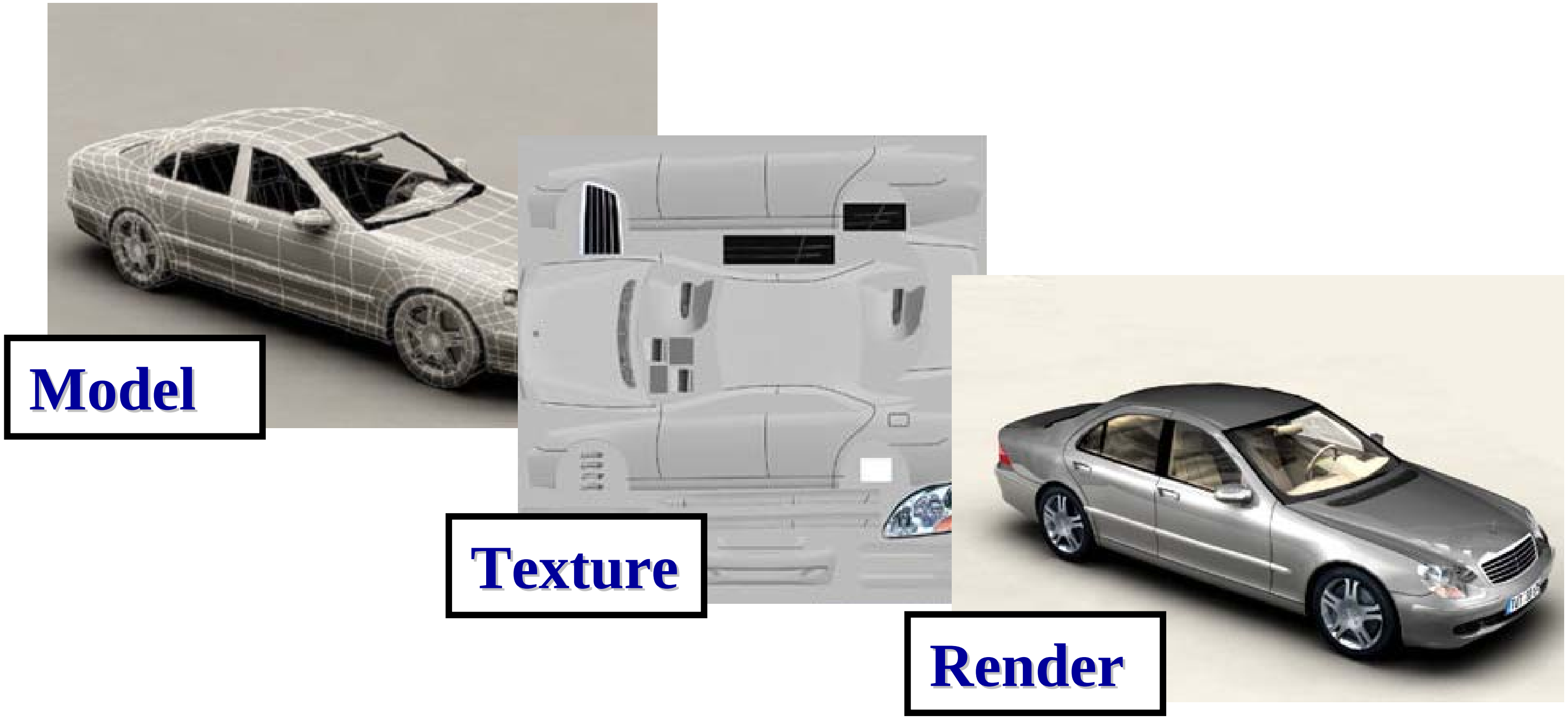
For more info on the computer artwork of Jeremy Birn see http://www.3drender.com/jbirn/productions.html

Talk Overview

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

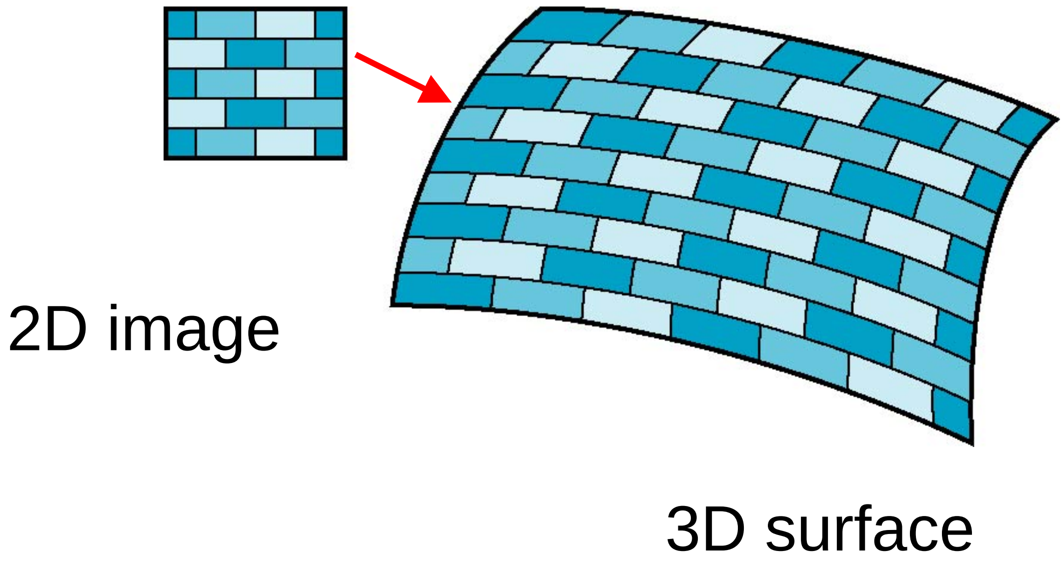
Texture Mapping

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

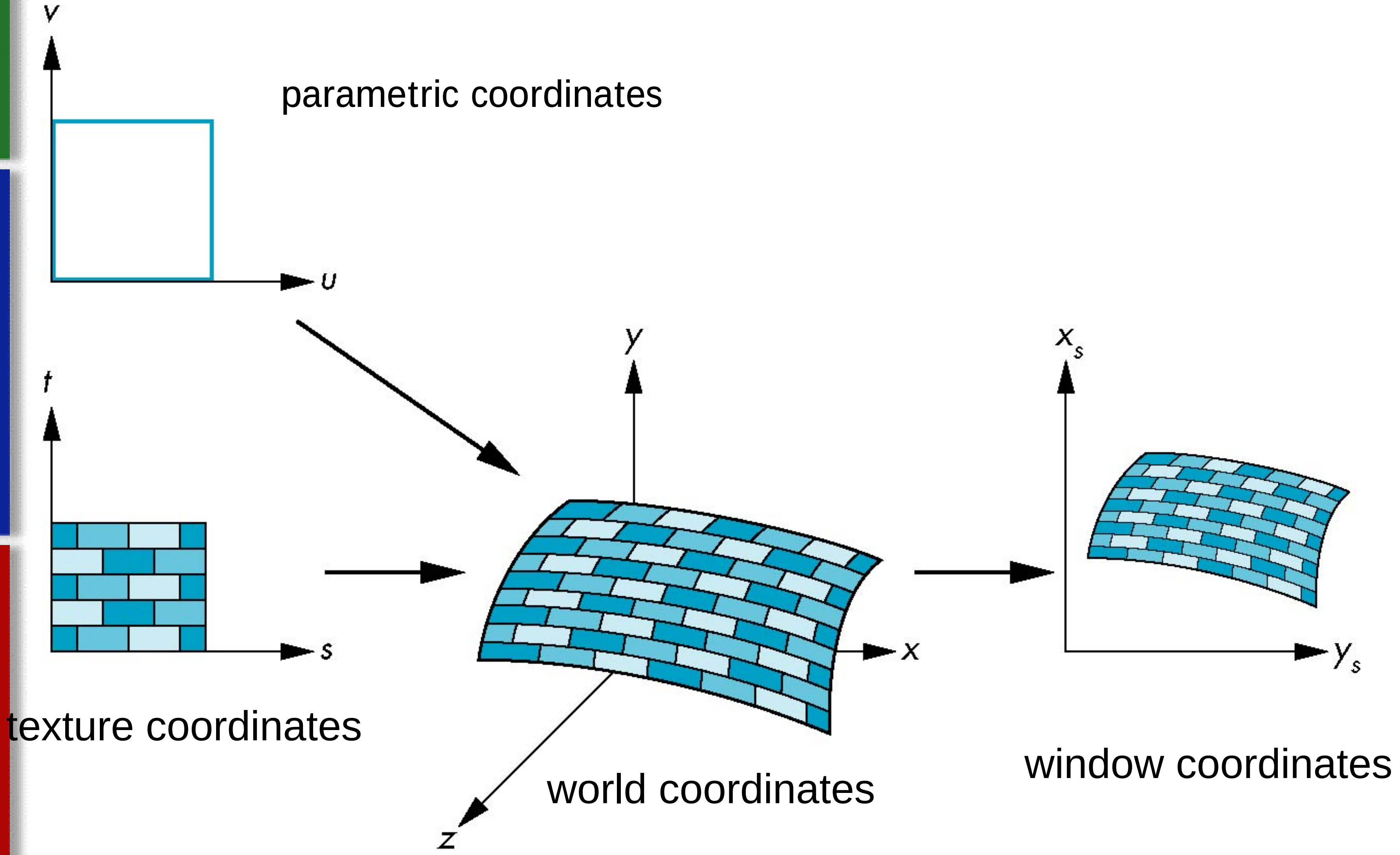


Is it simple?

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



Texture Mapping



Mapping Functions

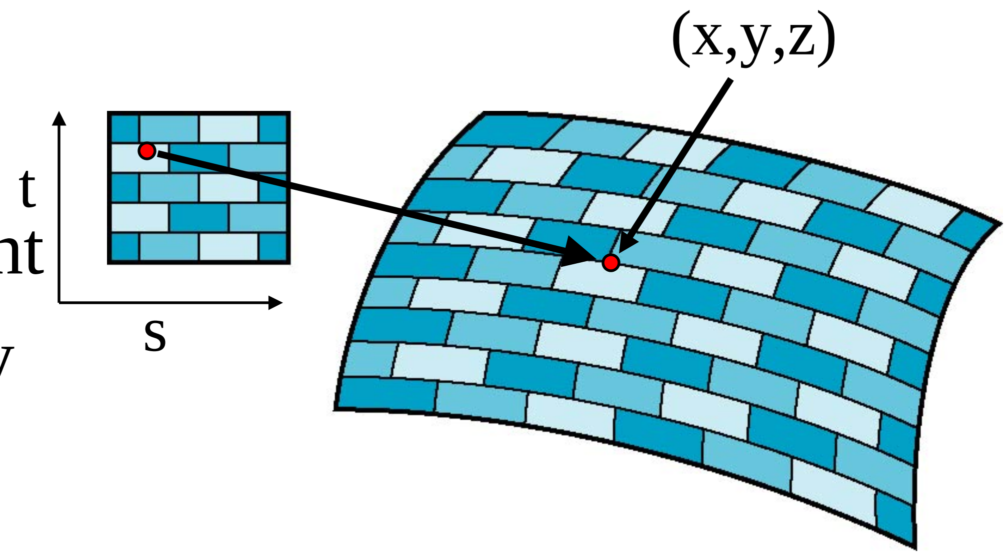
- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions

  x = x(s,t)

  y = y(s,t)

  z = z(s,t)

- But we really want to go the other way



Backward Mapping

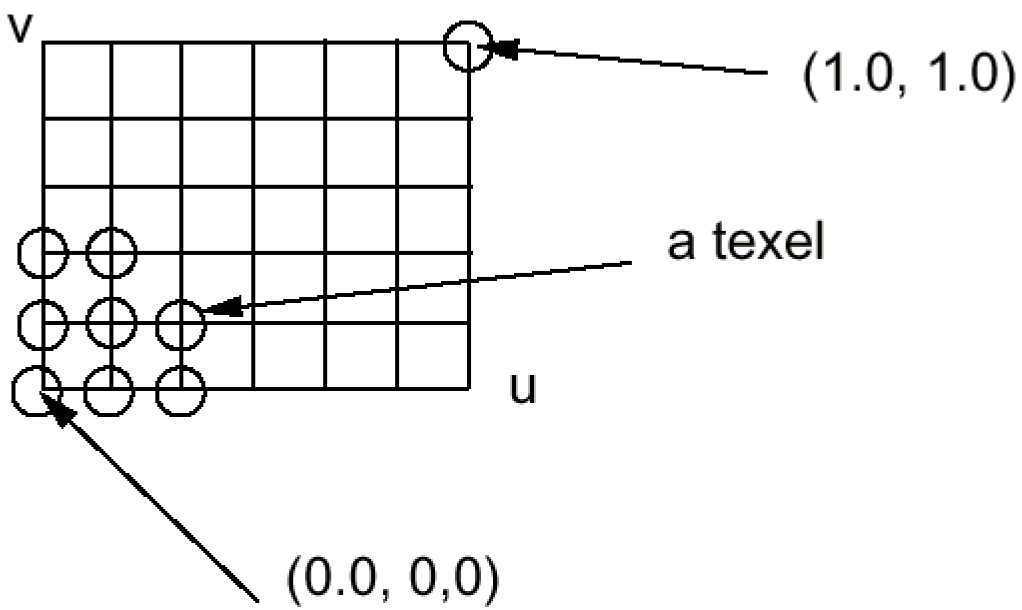
- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form

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

  $t = t(x,y,z)$
- 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?

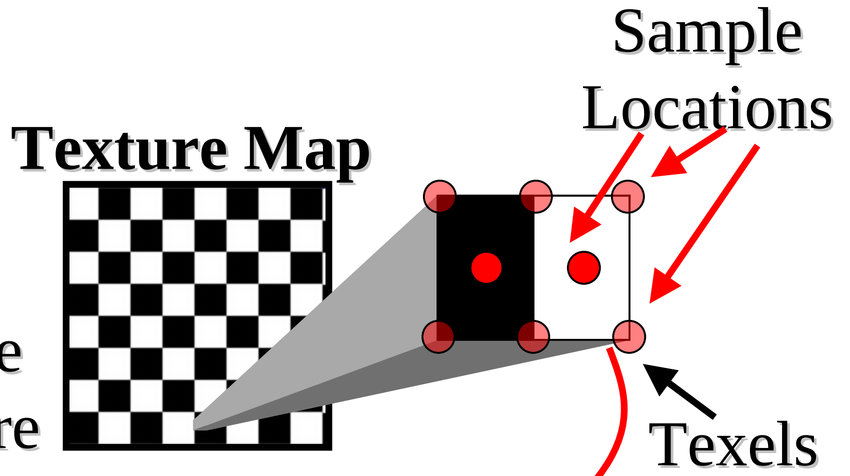
2 Problems:

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

2 Solutions:

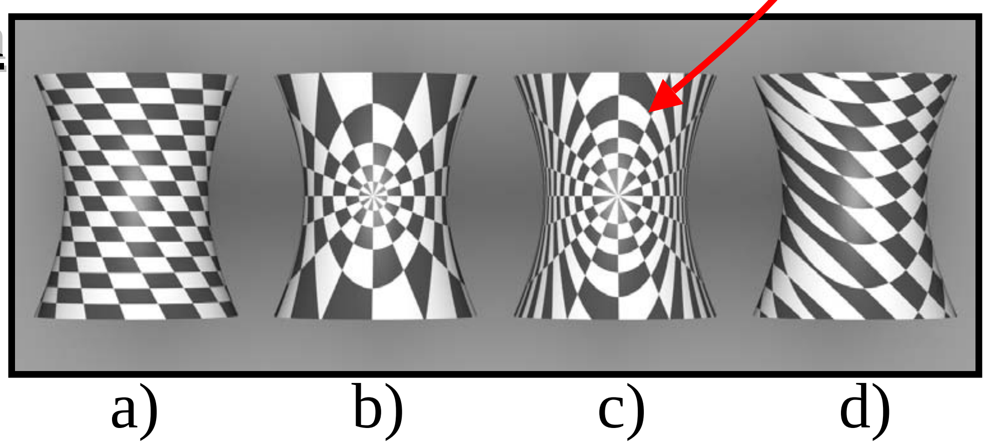
Sampling & Filtering

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



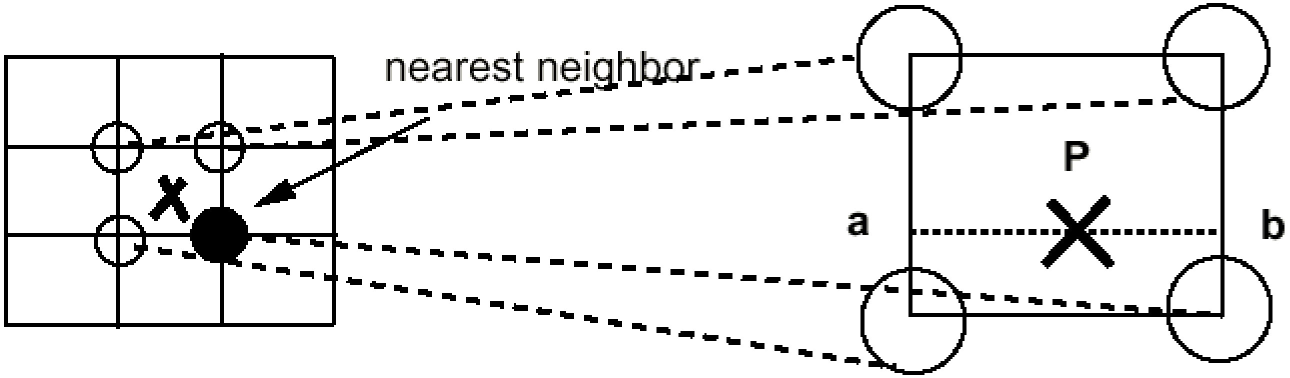
Coordinate Generation

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



(u,v) tuple

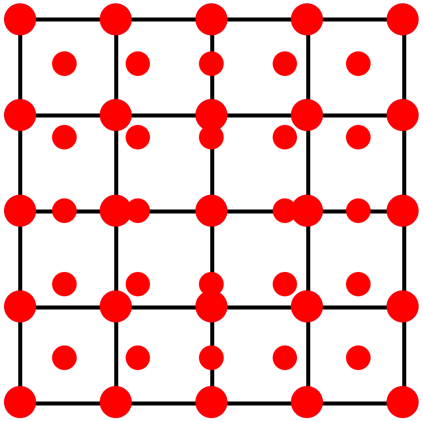
- For any (u,v) in the range of (0-1, 0-1) *multiplied by texture image width and height*, we can find the corresponding value in the texture map



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

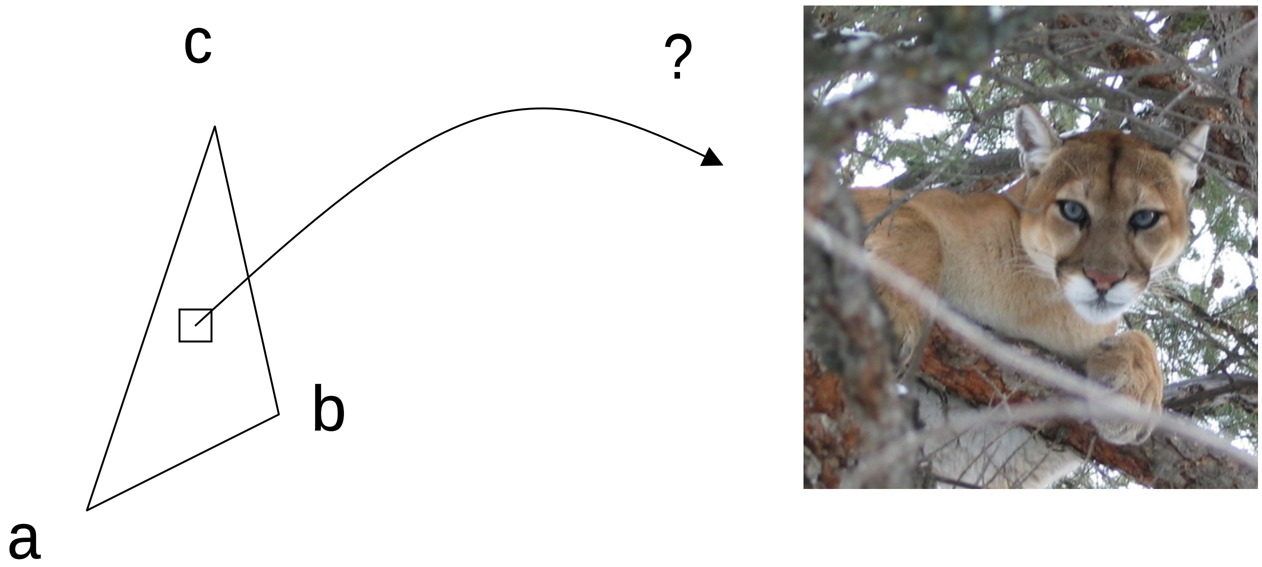


Nearest neighbor

Linear Interpolation

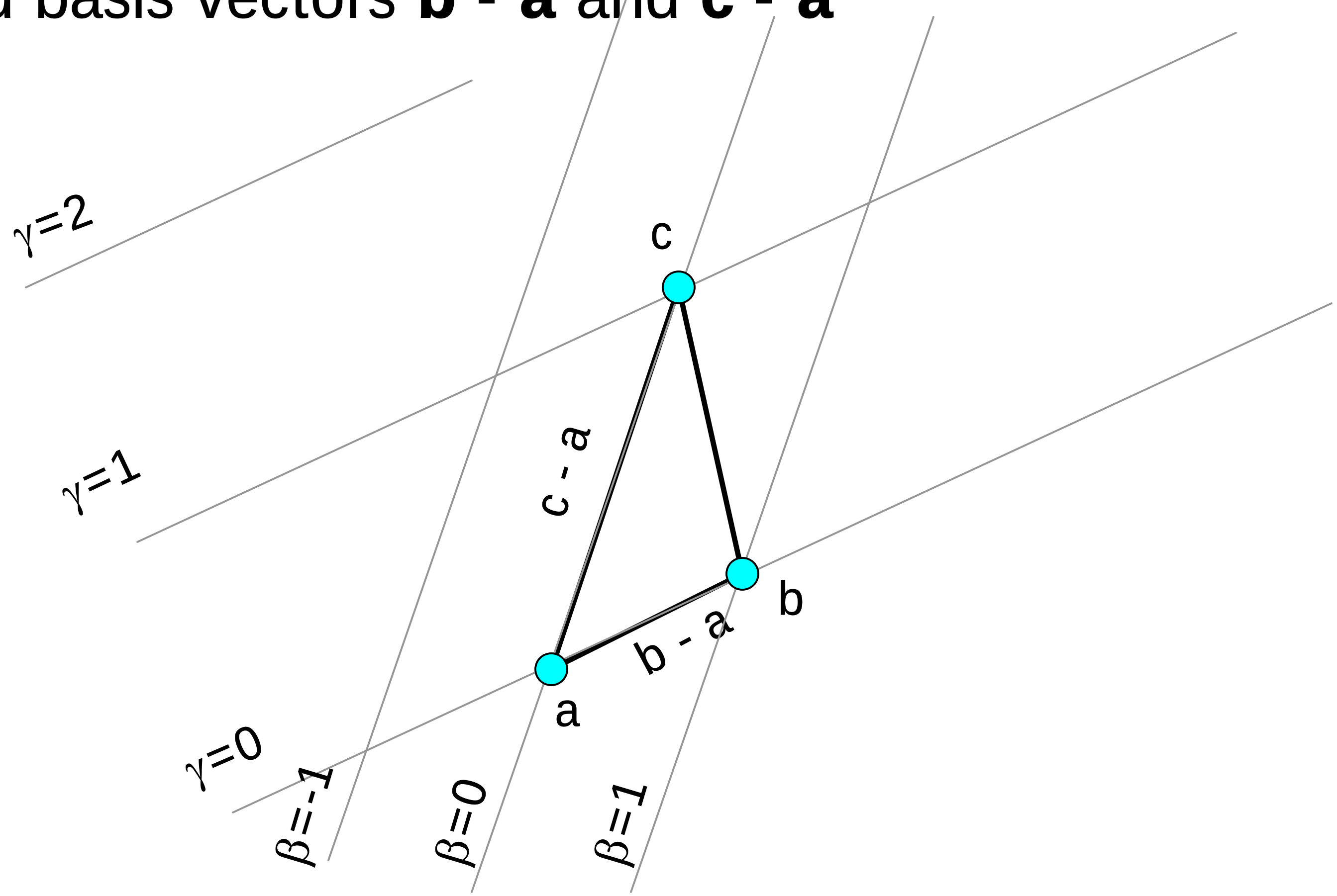
UV Coordinates For Triangles

- Given a triangle defined by three points (**a, b, c**), how do we associate a texture color with a point on the triangle?



Computing the Point

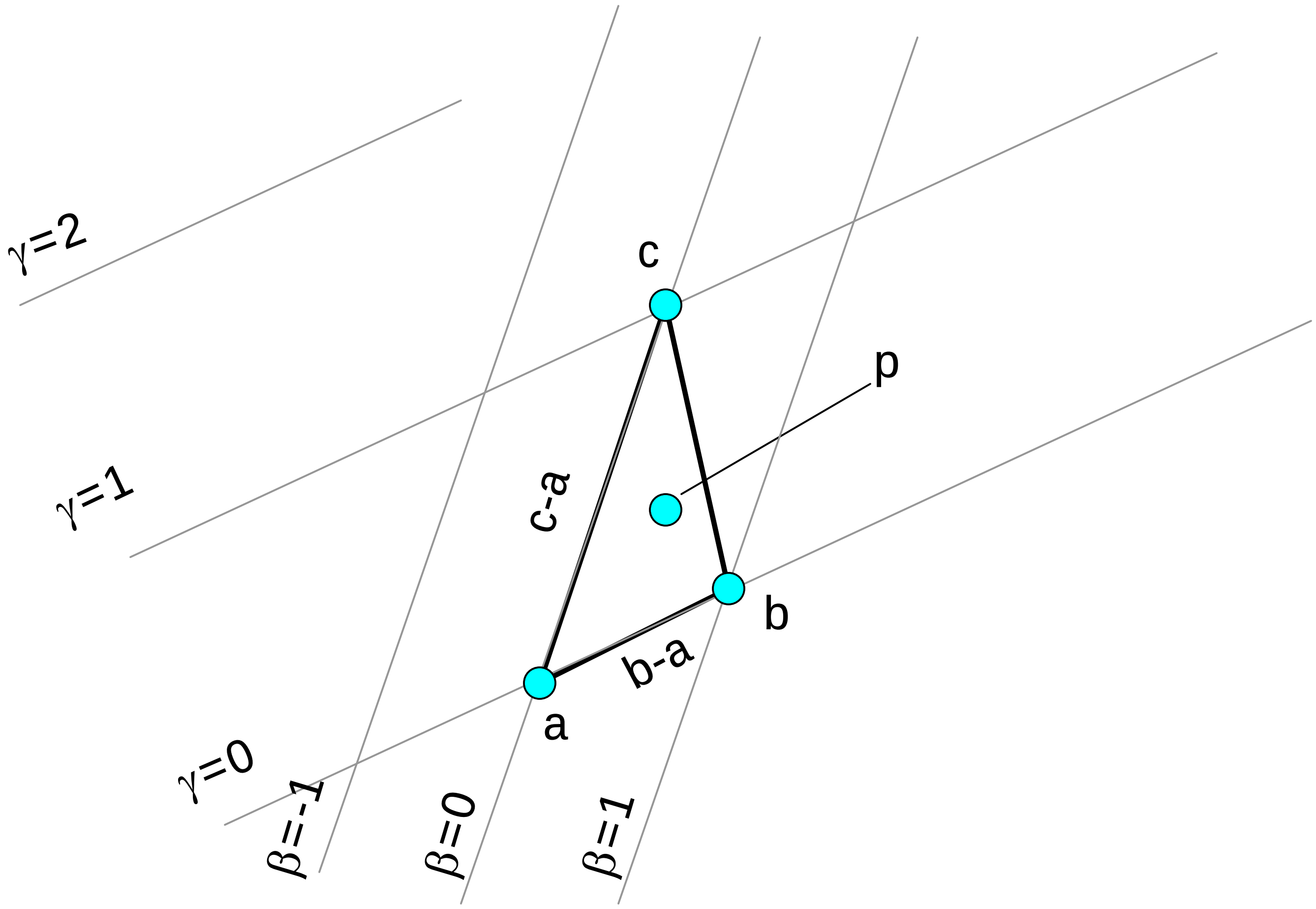
- Given the (x,y) point in the triangle, how do we transform that to a (u,v) point in the image?
- Set up a non-orthogonal coordinate system with origin **a** and basis vectors **b** - **a** and **c** - **a**



Barycentric coordinates

- Any point on the triangle can be defined by the barycentric coordinate

$$p = a + \beta(b-a) + \gamma(c-a)$$



Barycentric coordinates

- Once we have computed the (β,γ) barycentric coordinate for the triangle, we can determine the corresponding (**u, v**) point.
- First, establish the (**u, v**) system:

(0, 1) (1, 1)

(0, 0) (1, 0)

Computing the (u, v) coordinate

- $u(\beta, \gamma) = u_a + \beta(u_b - u_a) + \gamma(u_c - u_a)$
- $v(\beta, \gamma) = v_a + \beta(v_b - v_a) + \gamma(v_c - v_a)$

Example: Texture Mapping On GPU

Assumptions:

- We Have Existing Geometry
- Texture Coordinates Pre-generated
- Texture map

We Can Write 2 Shaders:

- **Vertex** – Set Geometry & Pass Through Coordinates
- **Fragment** – Sample Texture & Apply Pixel to Shading

Example: Texture Mapping On GPU

Vertex Shader

```
struct Vert_Output {
  float4 position   : POSITION;
  float3 color          : COLOR;
  float2 texCoord : TEXCOORD0;
};

Vert_Output vert_shader(
           float2 position : POSITION,
           float3 color    : COLOR,
           float2 texCoord : TEXCOORD0)
{
 Vert_Output OUT;

  OUT.position   = float4(position,0,1);
  OUT.color      = color;
  OUT.texCoord   = texCoord;

  return OUT;
}
```

Example: Texture Mapping On GPU

Fragment Shader

```
struct frag_Output {
  float4 color : COLOR;
};

frag_Output frag_shader(
                float2 texCoord : TEXCOORD0,
                uniform sampler2D decal : TEX0)
{
  frag_Output OUT;
  OUT.color = tex2D(decal, texCoord);
  return OUT;
}
```

Applying Our Mapping knowledge

Further Realism Improvements:

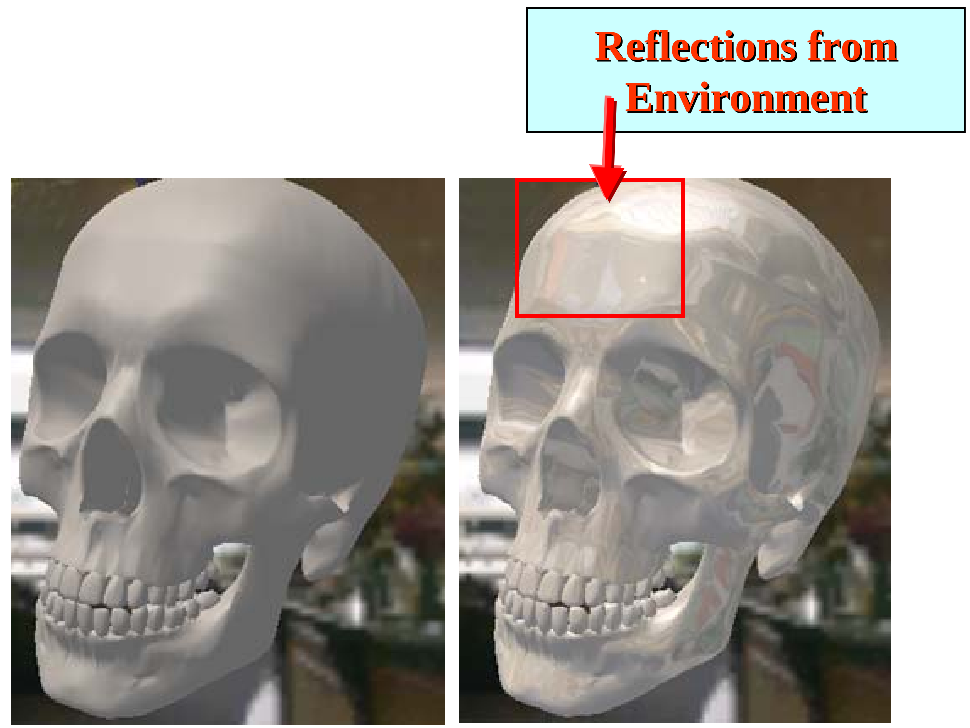
- Environment Mapping
- Bump Mapping
- Displacement Mapping
- Illumination Mapping & Others?

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

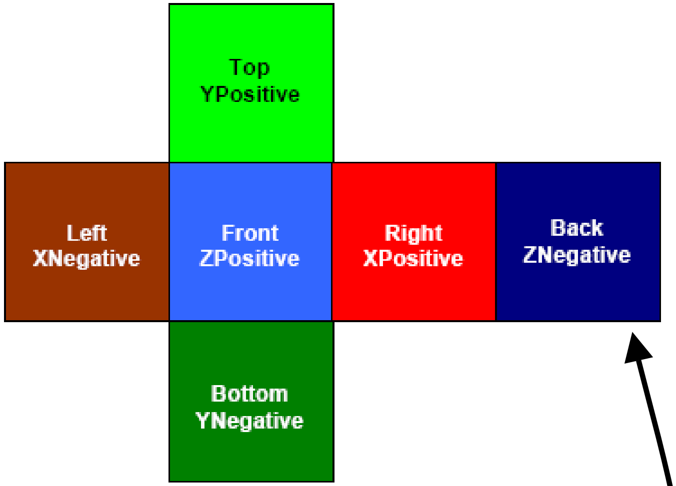
Cubic Mapping

- Camera takes orthographic pictures in six axis
- (-X, X, Y, -Y, Z, -Z)
- Map Look Up **=** Calculating a **reflection** vector

X, Y, Z

I.E.: R = (3.14, .21, -8.7)

Z is largest & negative



Cube Texture Map

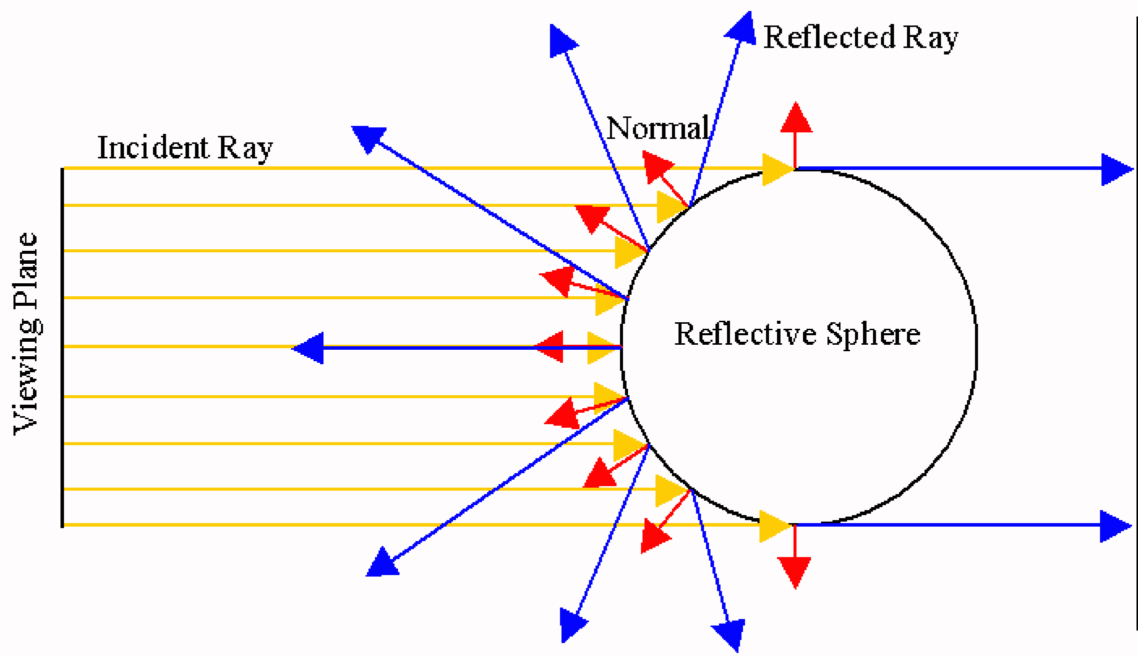
[image courtesy of NVidia.com]

* Index into the Negative Z region (dark blue)

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

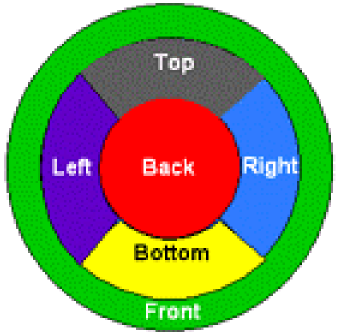
Sphere Mapping

- Obtain the reflection vector: $\vec{R} = \vec{I} - 2.0 \cdot \vec{N} \cdot (\vec{N} \bullet \vec{I})$

Index into the Sphere map:

$$s = \frac{R_x}{m} + \frac{1}{2}, \quad t = \frac{R_y}{m} + \frac{1}{2}$$

$$m = 2\sqrt{\left(R_x^2 + R_y^2 + \left(R_z + 1\right)^2\right)}$$



[image courtesy of nVidia.com]

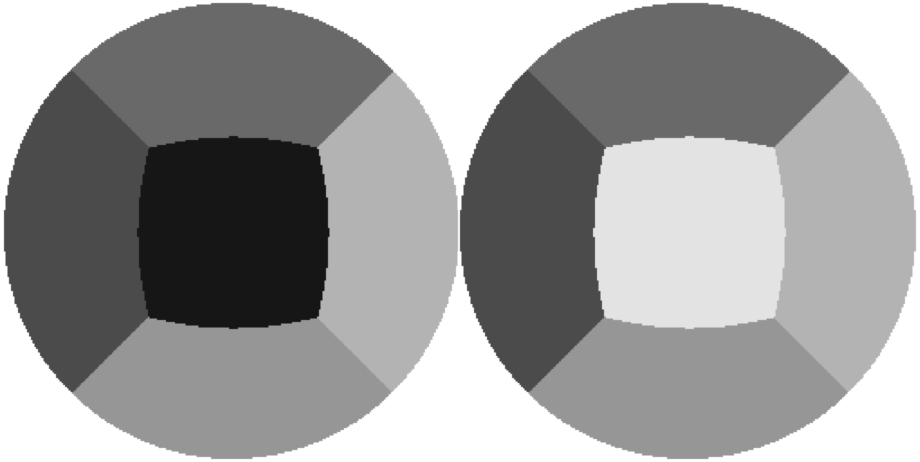
Environment Mapping

Paraboloid Mapping

$$f(x,y)=\frac{1}{2}-\frac{1}{2}\left(x^2+y^2\right)$$, where $x^2+y^2\leq 1$

High Lights:

- 2 textures, 1 per hemisphere
- No artifacts at poles
- Requires 2 passes or 2 texture fetches to render



[**Shaded areas of Paraboloid Map**, image adapted from [phd]]

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,

Vertex Shader

```
//
// Vertex shader for environment mapping with an
// equirectangular 2D texture
//
// Authors: John Kessenich, Randi Rost
//
// Copyright (c) 2002-2006 3Dlabs Inc. Ltd.
//
// See 3Dlabs-License.txt for license information
//

varying vec3  Normal;
varying vec3  EyeDir;
varying float LightIntensity;

uniform vec3  LightPos;

void main(void)
{
    gl_Position    = ftransform();
    Normal         = normalize(gl_NormalMatrix * gl_Normal);
    vec4 pos       = gl_ModelViewMatrix * gl_Vertex;
    EyeDir         = pos.xyz;
    LightIntensity = max(dot(normalize(LightPos - EyeDir), Normal), 0.
```

Environment Mapping on GPU

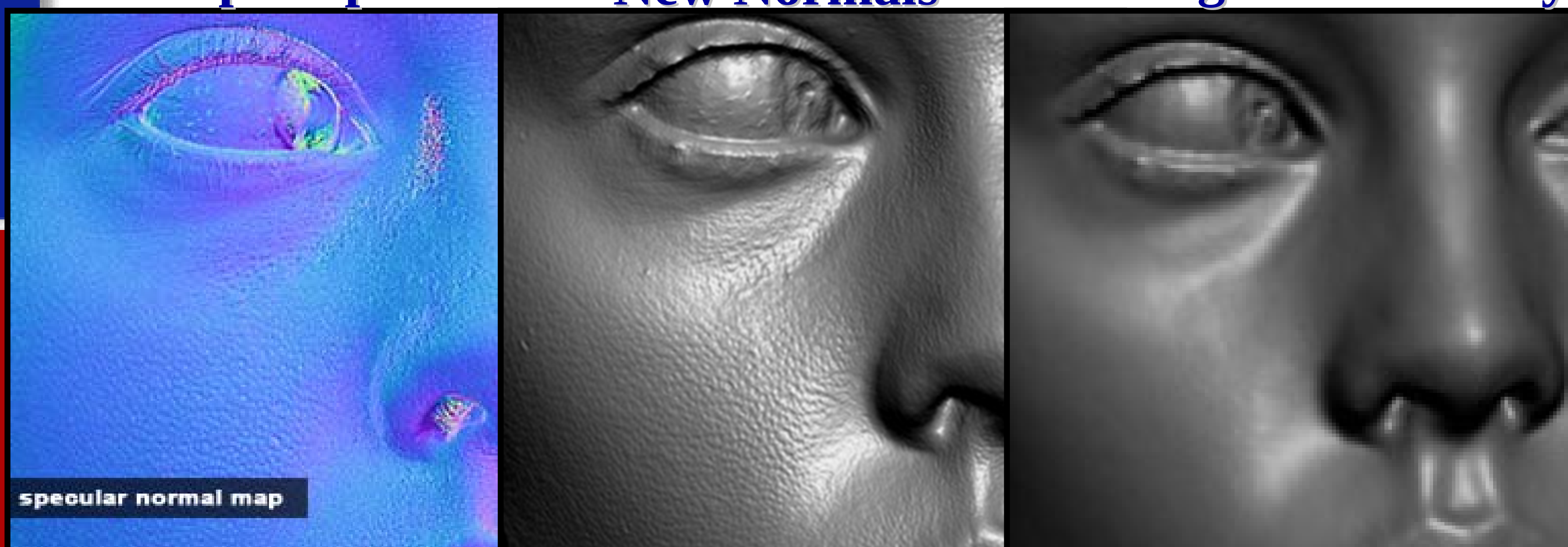
Fragment Shader

```
//
// Fragment shader for environment mapping with an
// equirectangular 2D texture
//
// Authors: John Kessenich, Randi Rost
//
// Copyright (c) 2002-2006 3Dlabs Inc. Ltd.
//
// See 3Dlabs-License.txt for license information
//

const vec3 Xunitvec = vec3 (1.0, 0.0, 0.0);
const vec3 Yunitvec = vec3 (0.0, 1.0, 0.0);

uniform vec3  BaseColor;
uniform float MixRatio;

uniform sampler2D EnvMap;

varying vec3  Normal;
varying vec3  EyeDir;
varying float LightIntensity;

void main (void)
```

Bump Mapping

<u>Main idea:</u> “Combines per-fragment lighting with surface normal perturbations supplied by a texture, in order to <u>simulate</u> 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

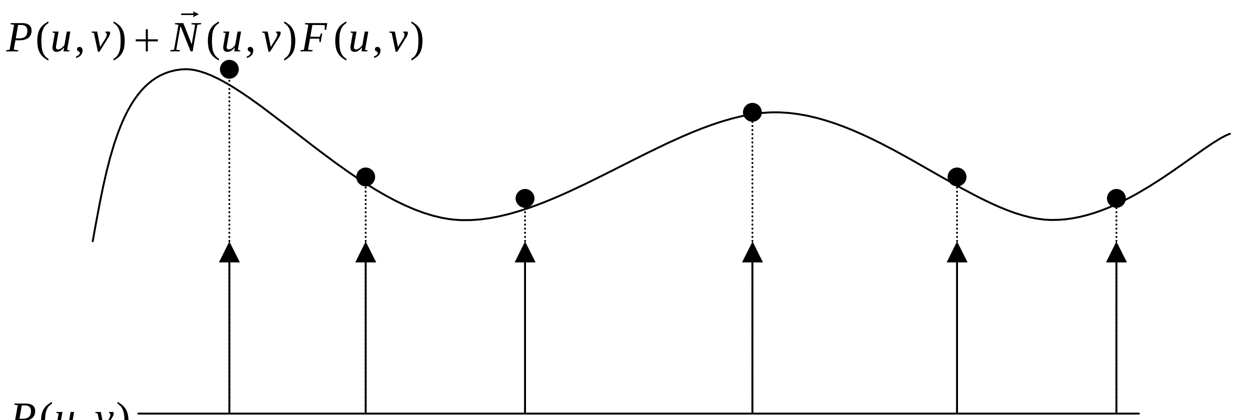


Image Adapted from [pbrt]

* 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*!!!**

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

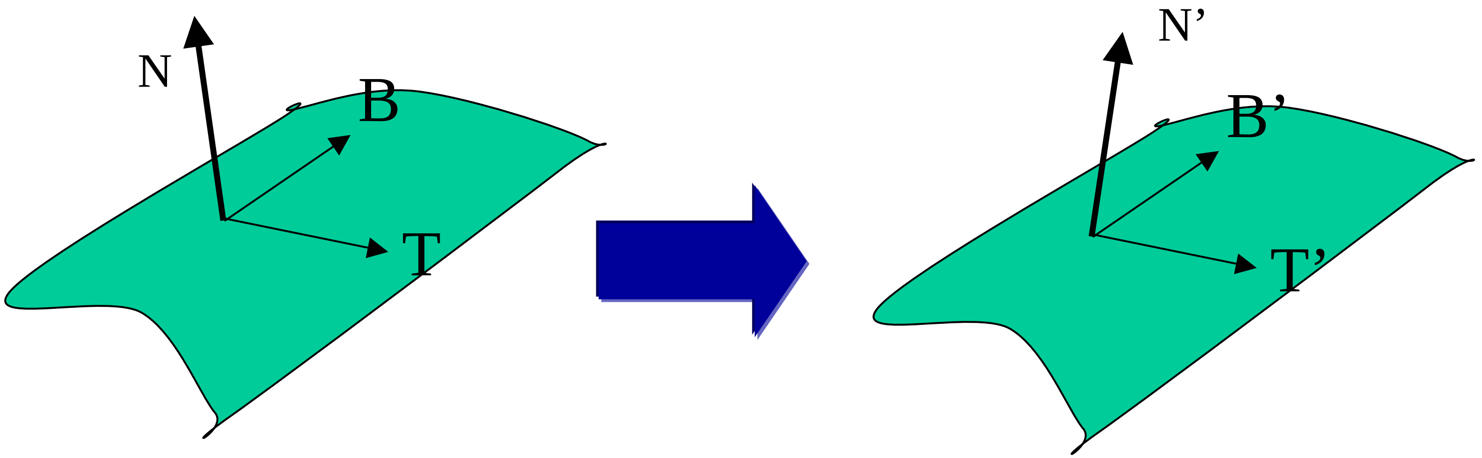
Tangent Space

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

$$N' = P'_u \times P'_v$$

$$= N + B(N \times P_v) - T(N \times P_u)$$

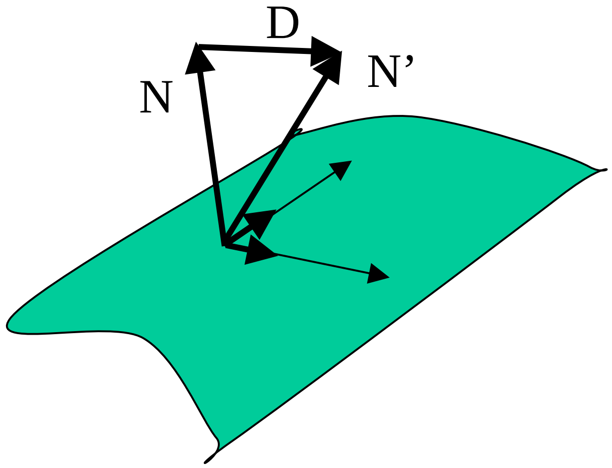
•

•

•

$$= N \quad + \quad D$$

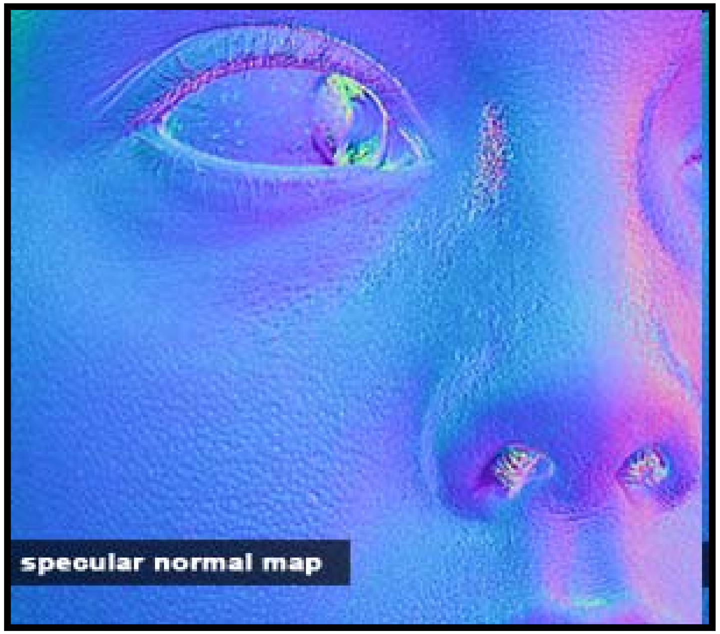
D is just the distance N has to move to be N'



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

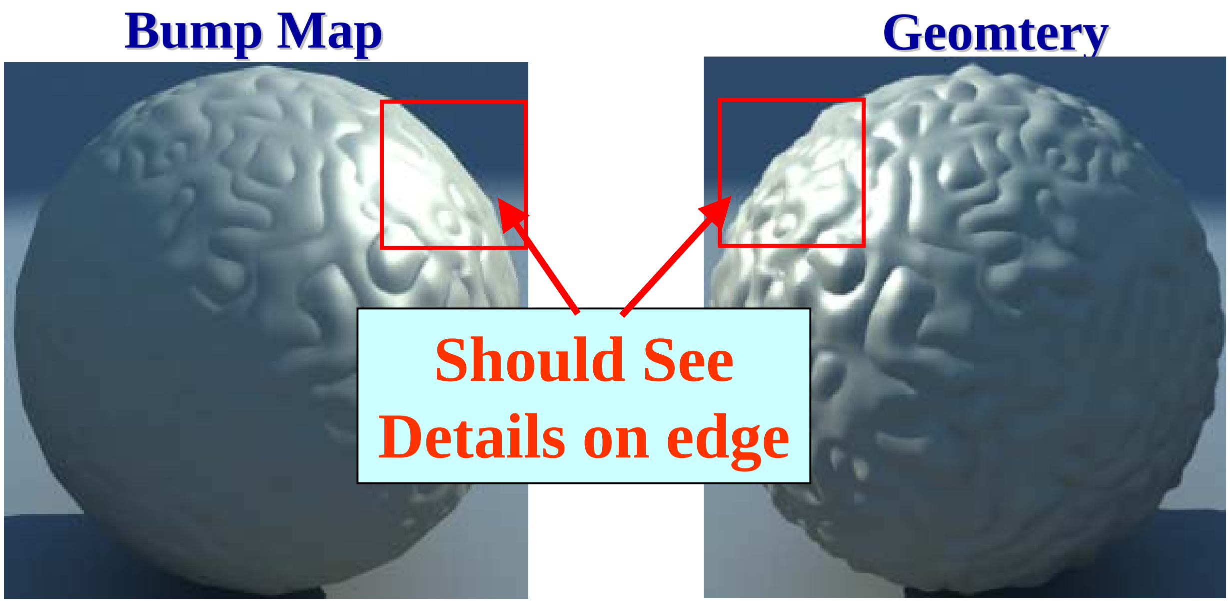
Bump Mapping

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 Mapping

Vertex Shader

```
attribute vec4 position;
attribute mat3 tangentBasis;
attribute vec2 texcoord;

uniform vec3 light;
uniform vec3 halfAngle;
uniform mat4 modelViewI;

varying vec2 uv;
varying vec3 lightVec;
varying vec3 halfVec;
varying vec3 eyeVec;

void main()
{
    // output vertex position
    gl_Position = gl_ModelViewProjectionMatrix * position;

    // output texture coordinates for decal and normal maps
```

Bump Mapping

Fragment Shader

```
uniform sampler2D decalMap;
uniform sampler2D heightMap;
uniform sampler2D normalMap;

uniform bool parallaxMapping;

varying vec2 uv;
varying vec3 lightVec;
varying vec3 halfVec;
varying vec3 eyeVec;

const float diffuseCoeff = 0.7;
const float specularCoeff = 0.6;

void main()
{
    vec2 texUV;

    if (parallaxMapping)
```

Displacement Mapping

<u>Main Idea:</u> 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

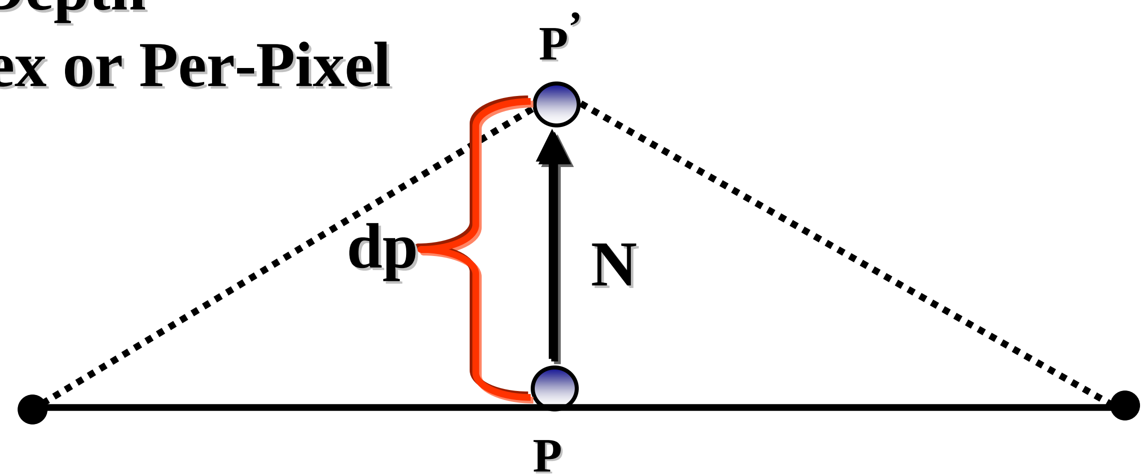
Without Displacement

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

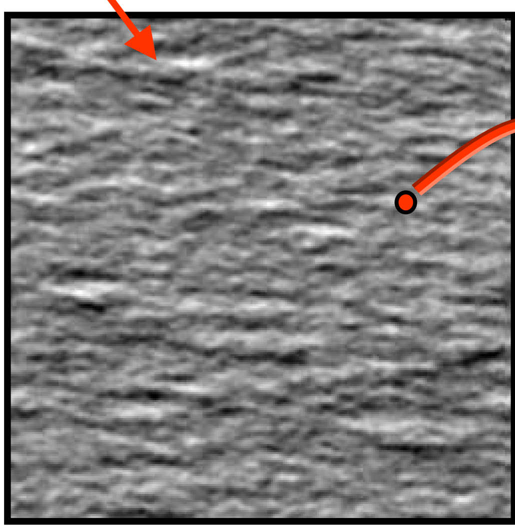
Displacement Mapping

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

Could be Heightfield



$P' = P + (N * dp)$



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

[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

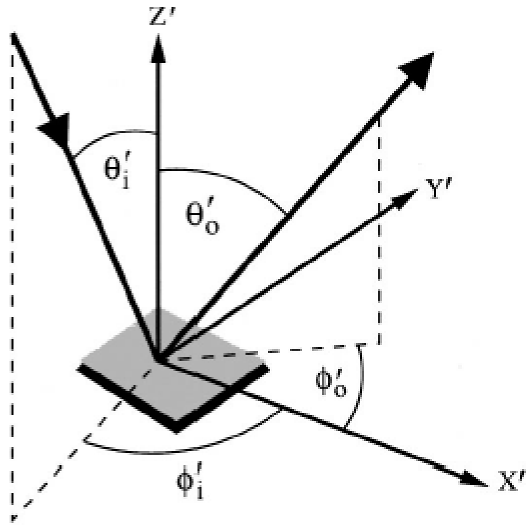
Questions?

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

Homework: Texture Shading

Pre-computing Reflectance



- Most Reflectance Functions can be factored or broken up with the parts being factorable.
- Factor over 2 variables: θ, ϕ

A)Ashikhmin-Shirley

B) Poulin-Fournier

C)Vinyl (measured)

D)Alum. Foil (measured)

Homework: Texture Shading

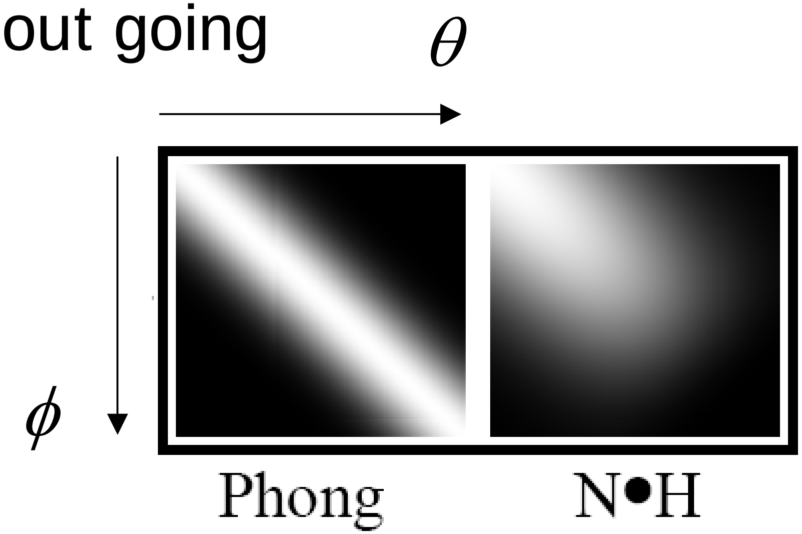
Texture reference(precompute & run time)

Precompute:

- Increment through θ, ϕ storing the evaluated/measured values in the appropriate texture coordinate

Run Time:

- Calculate the incoming and out going vector to get θ, ϕ
- Index into texture per



[Precomputed reflectance textures, Frequency Environment Mapping]

Homework: Texture Shading

More Examples

A)Ashikhmin-Shirley B) Poulin-Fournier C)Vinyl (measured) D)Alum. Foil (measured)