GPU Programming:
Environment Mapping
Bump Mapping
Displacement Mapping
Shadow Mapping

Spring 2025

Daniel G. Aliaga
Department of Computer Science
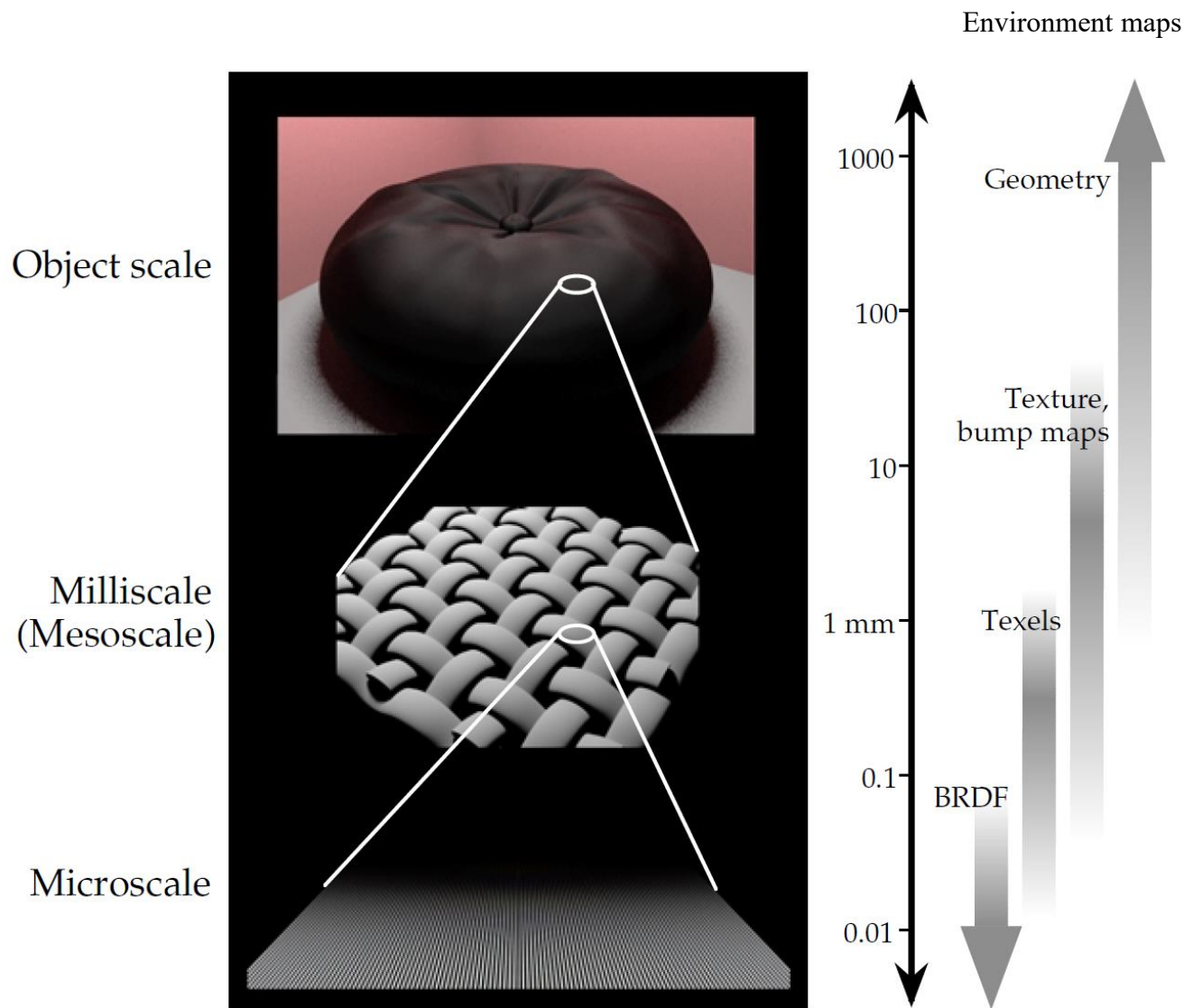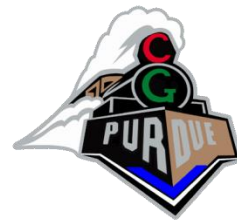Purdue University

Figure 1: Applicability of Techniques

Environment Mapping (or Reflection Mapping)

- Blinn, J. F. and Newell, M. E. Texture and reflection in computer generated images. Communications of the ACM Vol. 19, No. 10 (October 1976), 542-547
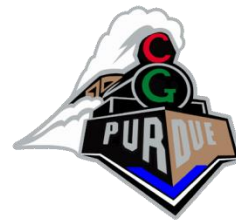
- The Abyss

- Terminator II

Environment Mapping

- Approximation
  - if the object is small compared to the distance to the environment, the illumination on the surface only depends on the direction of the reflected ray, *not* on the point position on the object

- Algorithm
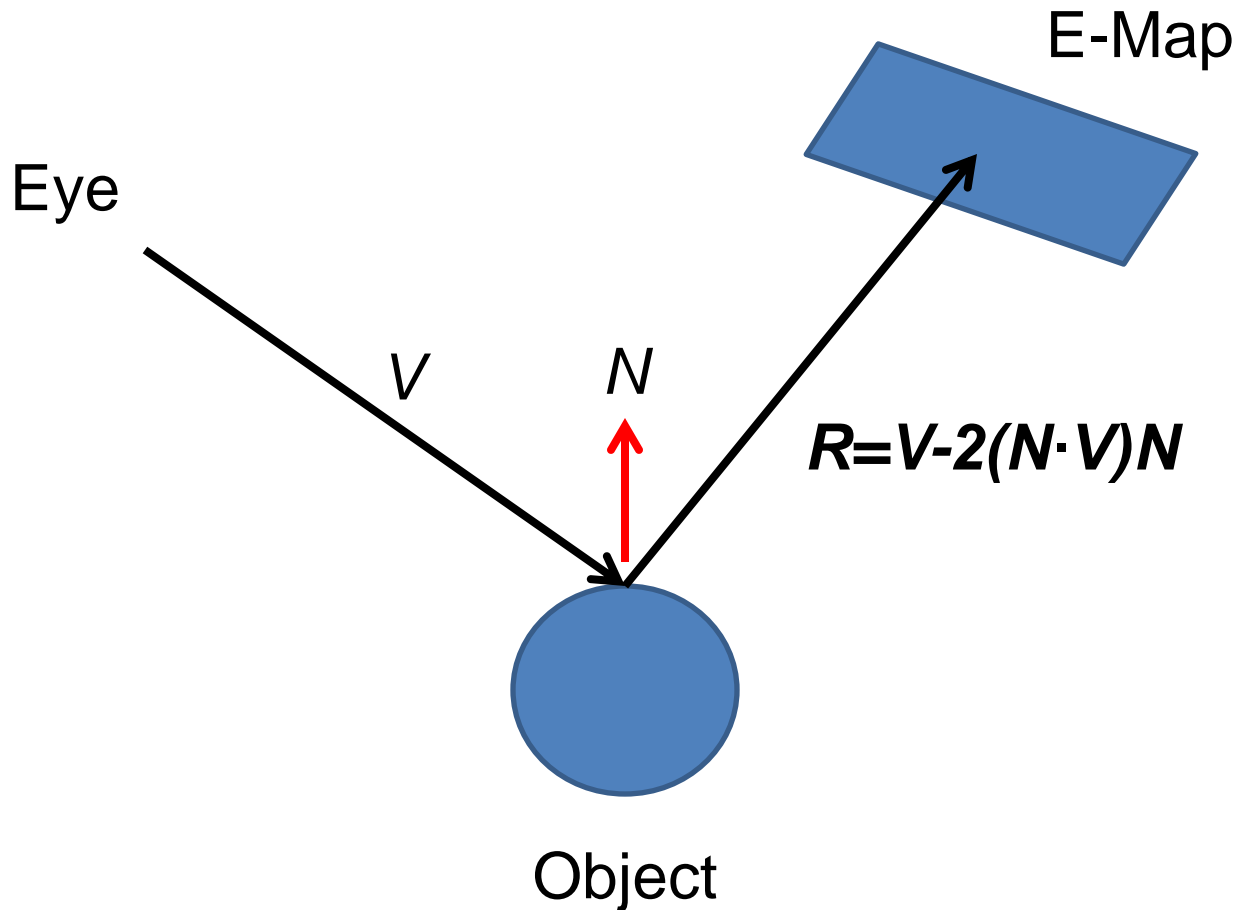 - pre-compute the incoming illumination and store it in a texture map

Environment Mapping

Eye

E-Map

V   *N*

R=?

Object

Environment Mapping

E-Map

Eye

V N

$R=V-2(N \cdot V)N$

Object
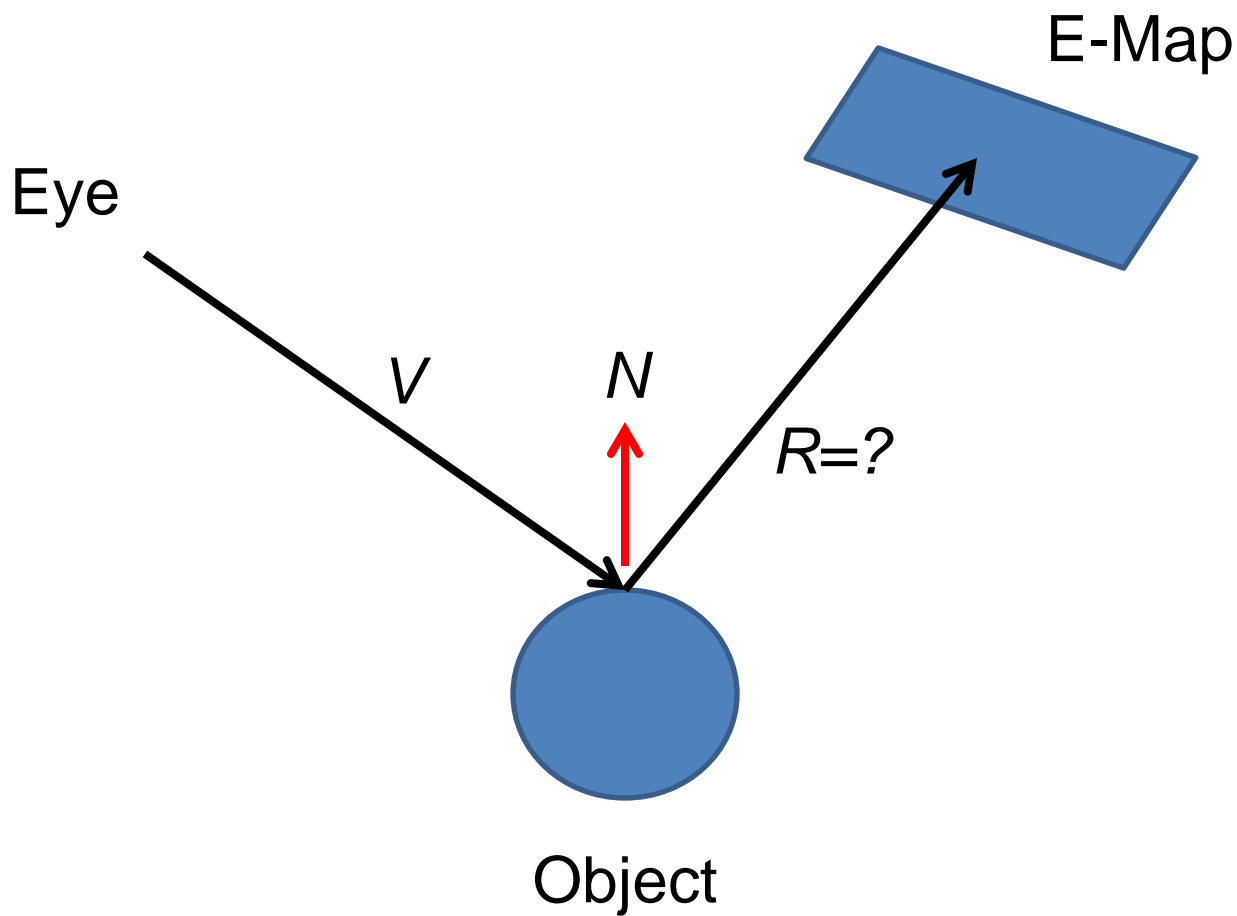
Environment Maps Forms

- Spherical Mapping

- Cubical Mapping (or Cube Map)
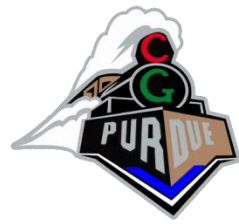
- Paraboloidal Mapping
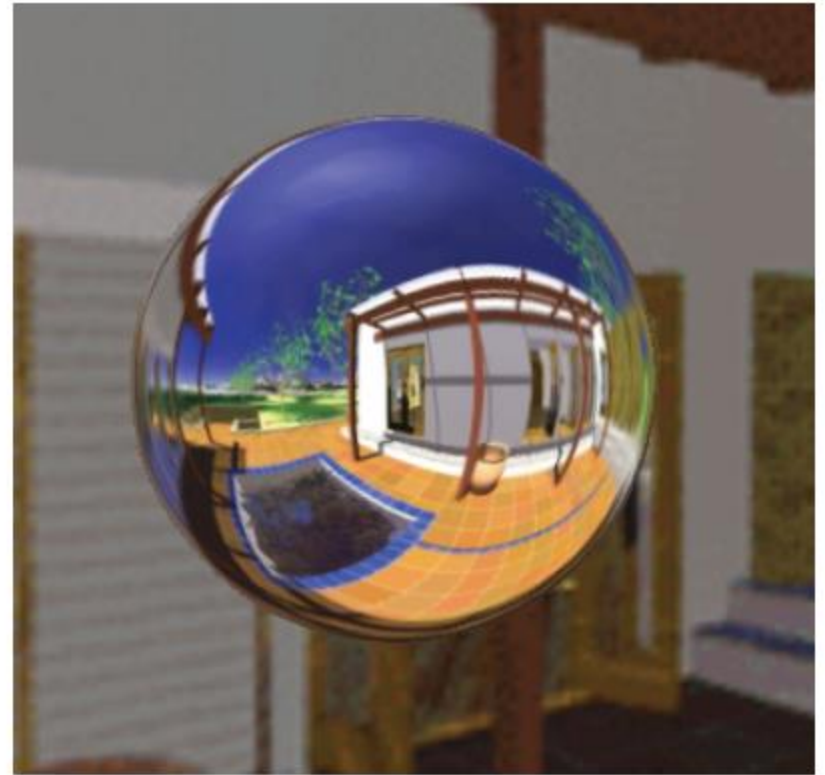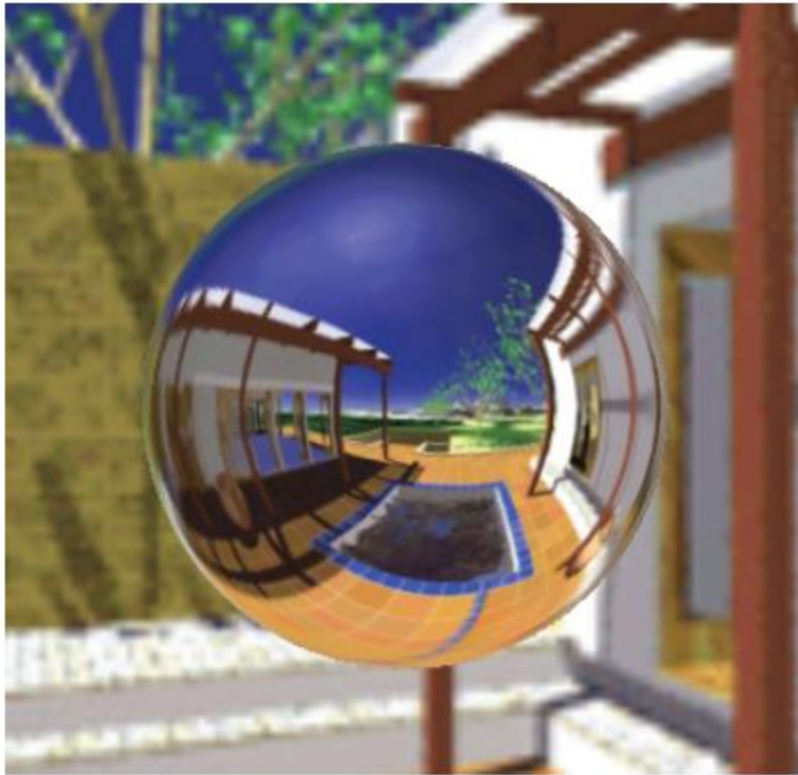
Spherical Mapping

Spherical Mapping
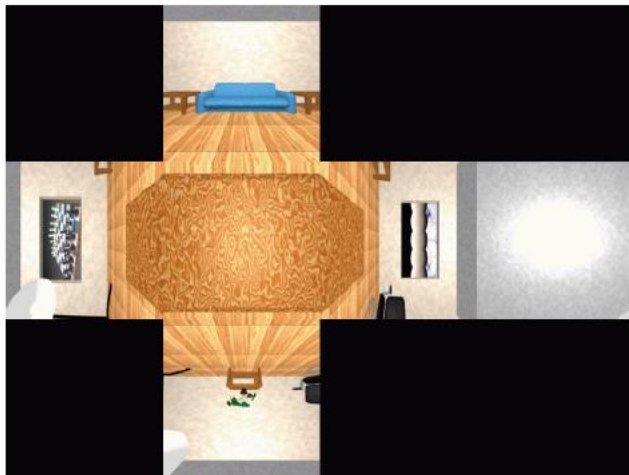

Matt Loper, MERL

Spherical Mapping

Matt Loper, MERL

Spherical Mapping: Renderings

Cubical Mapping

Cubical Mapping: Renderings

Bump Mapping

- Blinn, "Simulation of Wrinkled Surfaces", *Computer Graphics*, (Proc. Siggraph), Vol. 12, No. 3, August 1978, pp. 286-292. [PDF]
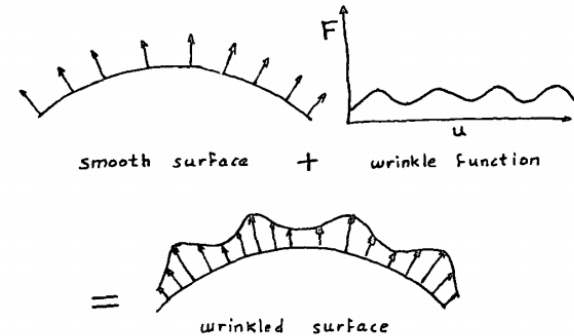
- Simulates small surface variations

- Key idea: tweak normals used for lighting (geometry stays the asme)

- Benefit: much more efficient, geometry-wise, than creating an approximation using very small triangles

14
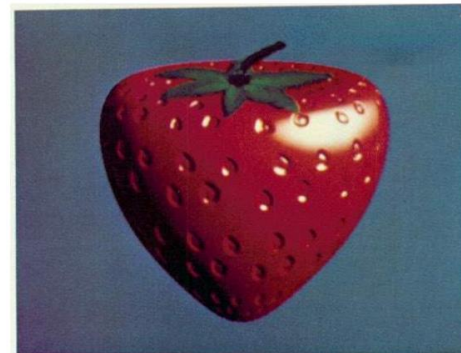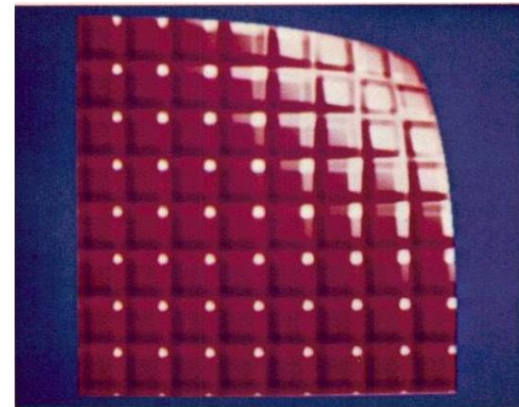
Bump Mapping

- Each texel stores two offsets (in u and in v)

Bump Mapping Demo

- https://apoorvaj.io/exploring-bump-mapping-with-webgl/
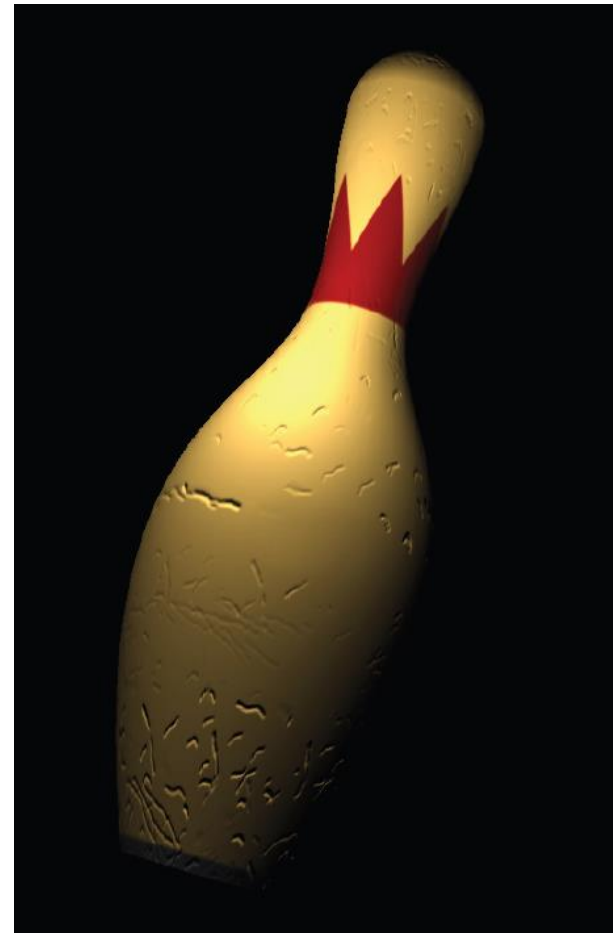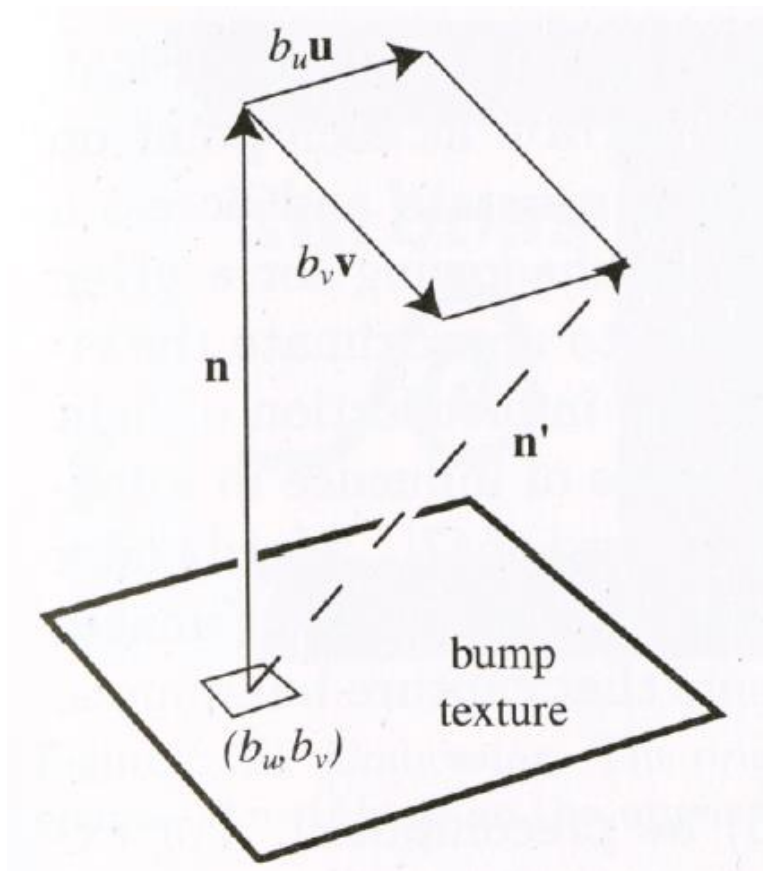
Normal Map (used for Bump Mapping)

- Use texel values to modify vertex/pixel normals of polygon
- Texel values correspond to normals (or heights) modifying the current normals
- RGB = ($\mathbf{n}$+1)/2
- $\mathbf{n}$ = 2*RGB − 1
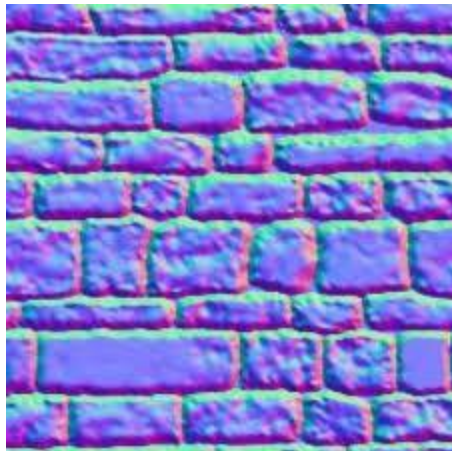
Bump Mapping

- The light source direction **L** and pixel normal **N** are represented in the global coord **x, y, z**

- The bump map normal **n** is in its local coordinates, which is called *tangent space* or *texture space*

    - *T: tangent vector*

    - *N: surface normal*

    - *B: bitangent*

    - *How to compute TNB?*
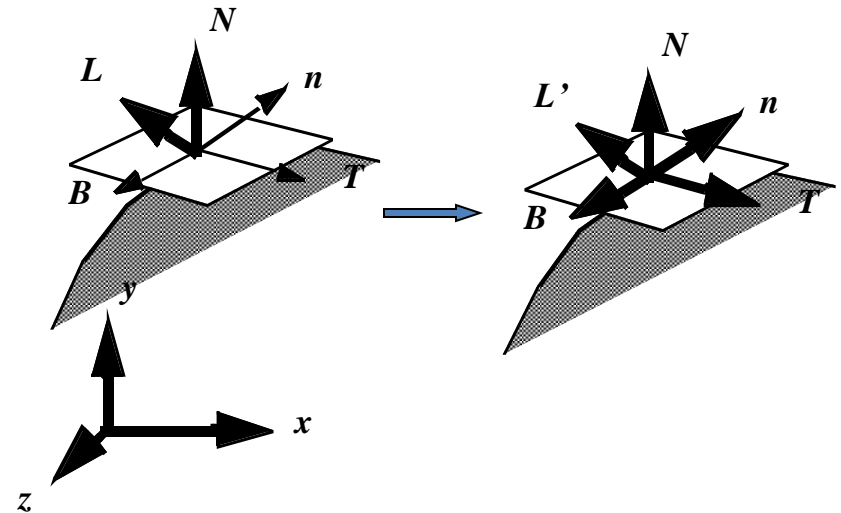
Bump Mapping

- Given triangle $\{v_1, v_2, v_3\}$:

  $T = (v_2 - v_1) \, / \, | v_2 - v_1 |$

  $N = T' \, / \, | T' | \, (\text{or } (v_2 - v_1) \times (v_3 - v_1))$

  $B = T \times N$
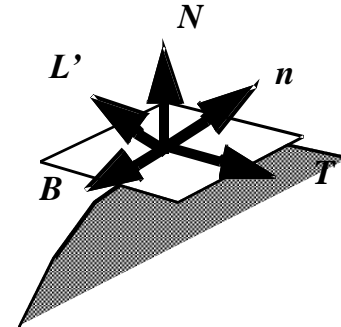
Bump Mapping

- Issue: adjacent triangles do not necessarily have similarly aligned T and B vectors which causes bump discontinuity

- Solutions:
  - 1: Compute TNB per triangle, and flip when it seems necessary
    - Might not work in all cases…

Bump Mapping

- Solutions:
  - 2: Assume a nicely organized mesh of triangles
    - Works, but assume a nicely organized mesh of triangles
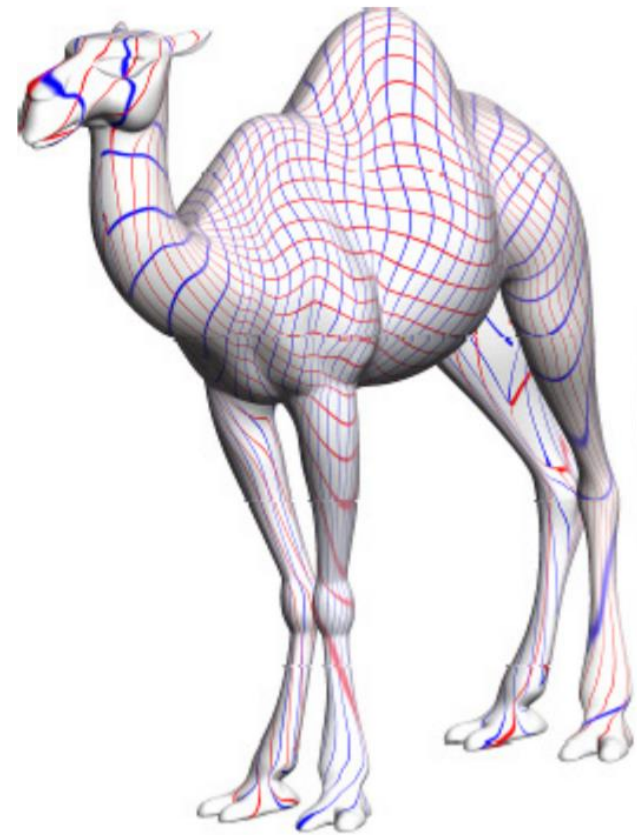
Bump Mapping

- Solutions:
  - 3: Use a 2D parameterization of the object surface
    - Works, but assumes a 2D parameterization
    - **How to compute such a 2D parameterization?**

Texture Coordinate Generation (or 2D/UV Parameterization)

- Recall texture mapping…

Texture Mapping

- Mechanism for attaching a texture (or image) to the modeled surface
  - *texels* – color samples in texture maps
  - corners of the image are (0, 0), (0, 1), (1, 1), and (1, 0)
  - tiling indicated with tex. coords. > 1
  - a pair of floats (s, t) for each (triangle) vertex

Texture

$P_2(0, 0)$

$P_1(1, 0)$

$P_3(0, 1)$

$P_4(1, 1)$

y

O x

z

Texture Mapping

$P_2(0, 0)$

$P_1(1, 0)$

$P_3(0, 1)$

$P_4(1, 1)$

a

c

P_2'

P_1'

P_3'

b

y

O

x

z

C
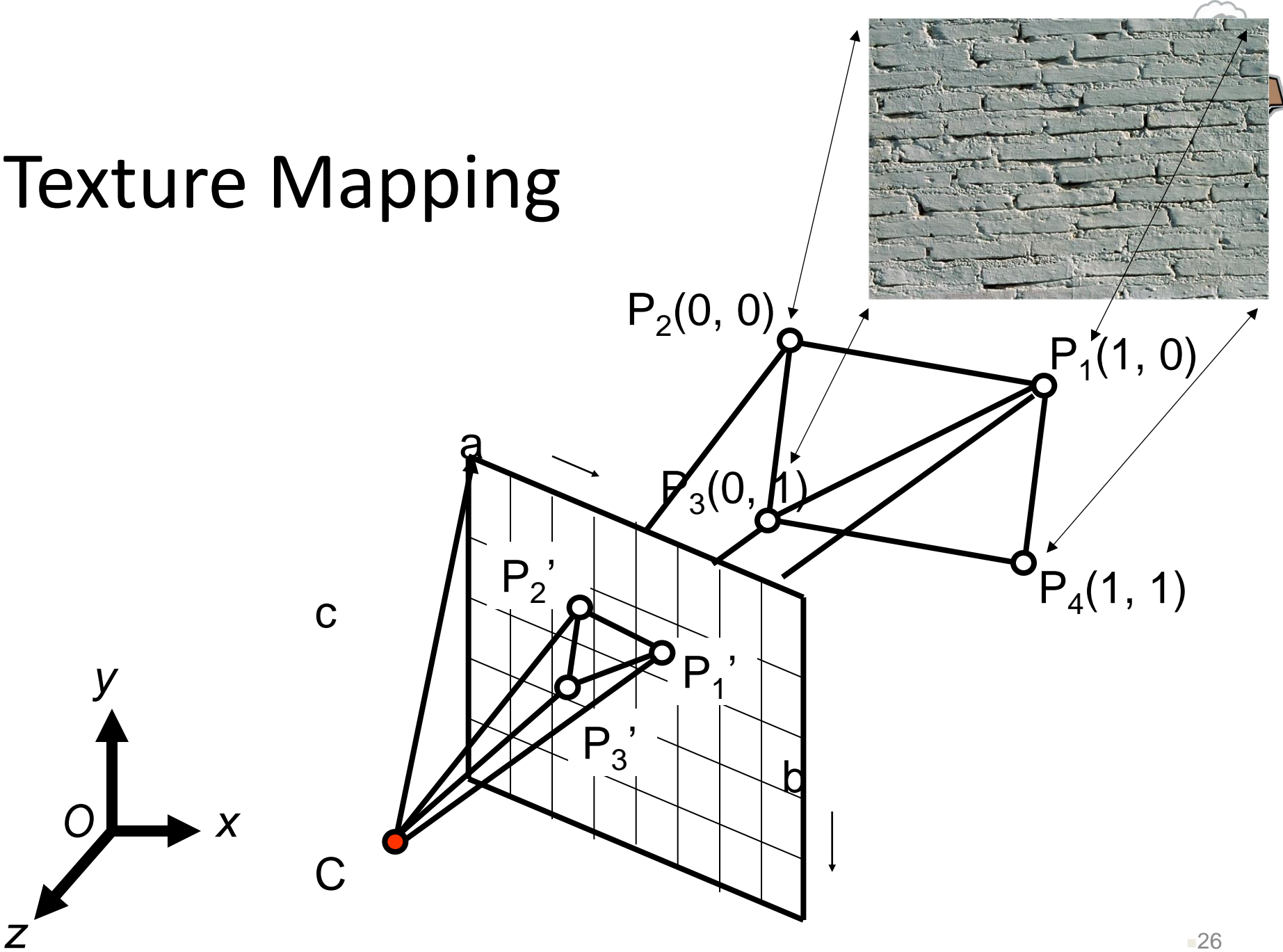
Texture Mapping

Problem*: how to compute the texture coordinates for an interior pixel?*

$P_2(u_2, v_2)$

$P_1(u_1, v_1)$

P (u,v)

$P_3(u_3, v_3)$

a

c

$P_2{}'$

P'

$P_1{}'$

$P_3{}'$

b

C

y

O

x

z
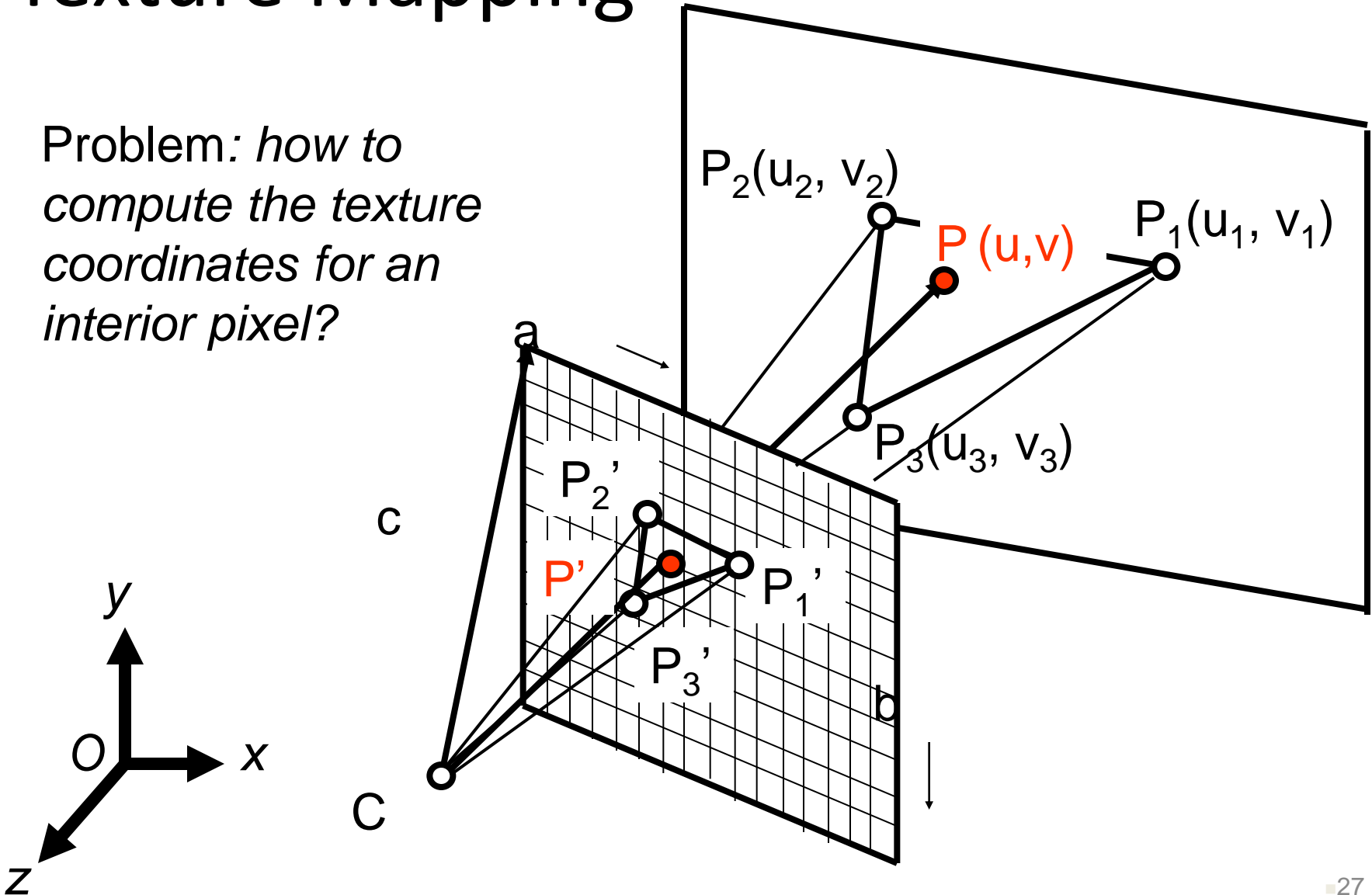
Texture Mapping

Solution: *interpolate vertex texture coordinates*

$P_2(u_2, v_2)$

$P (u, v)$

$P_1(u_1, v_1)$

$P_3(u_3, v_3)$

a

c

P_2'

P'

P_1'

P_3'

b

C

y

O

x

z
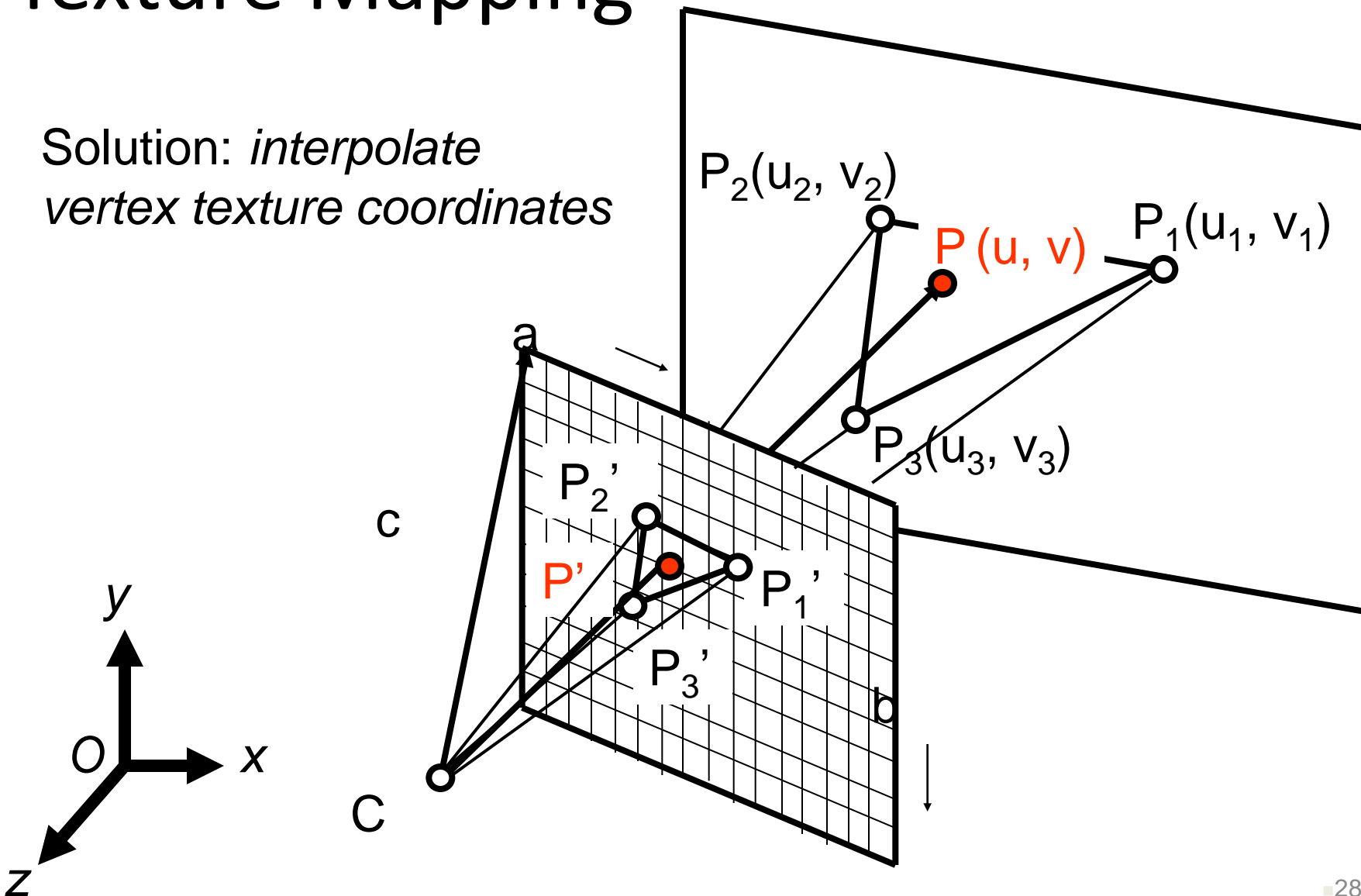
Parameter Interpolation

- Texture coordinates, colors, normals, etc.
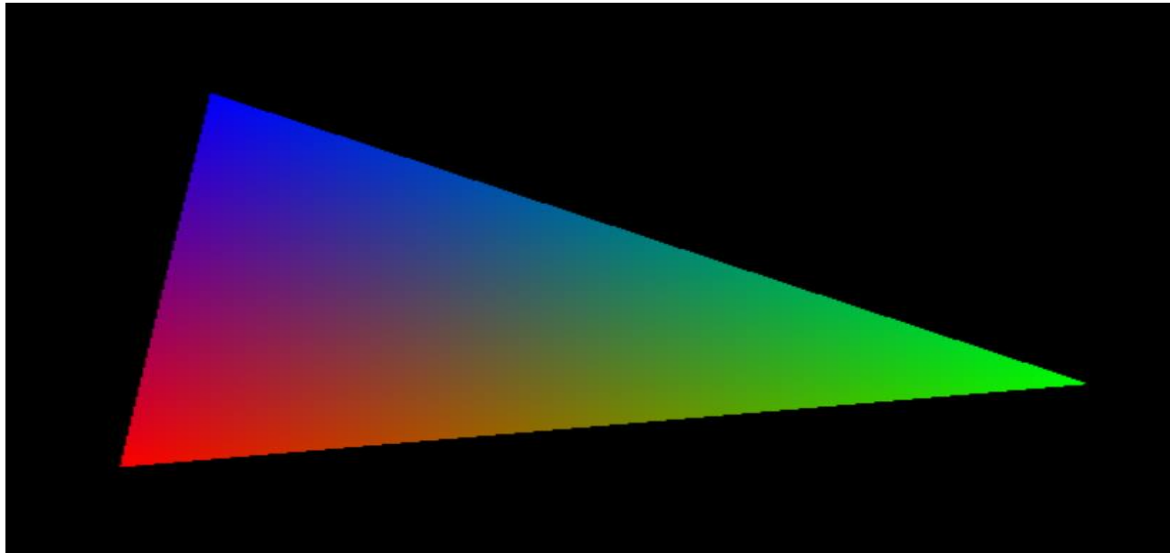
- How?
  - Use barycentric coordinates…
  - **Or, this is actually done by the GPU as "varying" parameters in the fragment shader**

Interpolation on the GPU

From the vertex values, the GPU interpolates texture coordinates for the intermediate pixels (as well as other values if so desired)

$P_1'(u_1, v_1)$

$P'(u, v)$

$P_3'(u_3, v_3)$

$P_2'(u_2, v_2)$
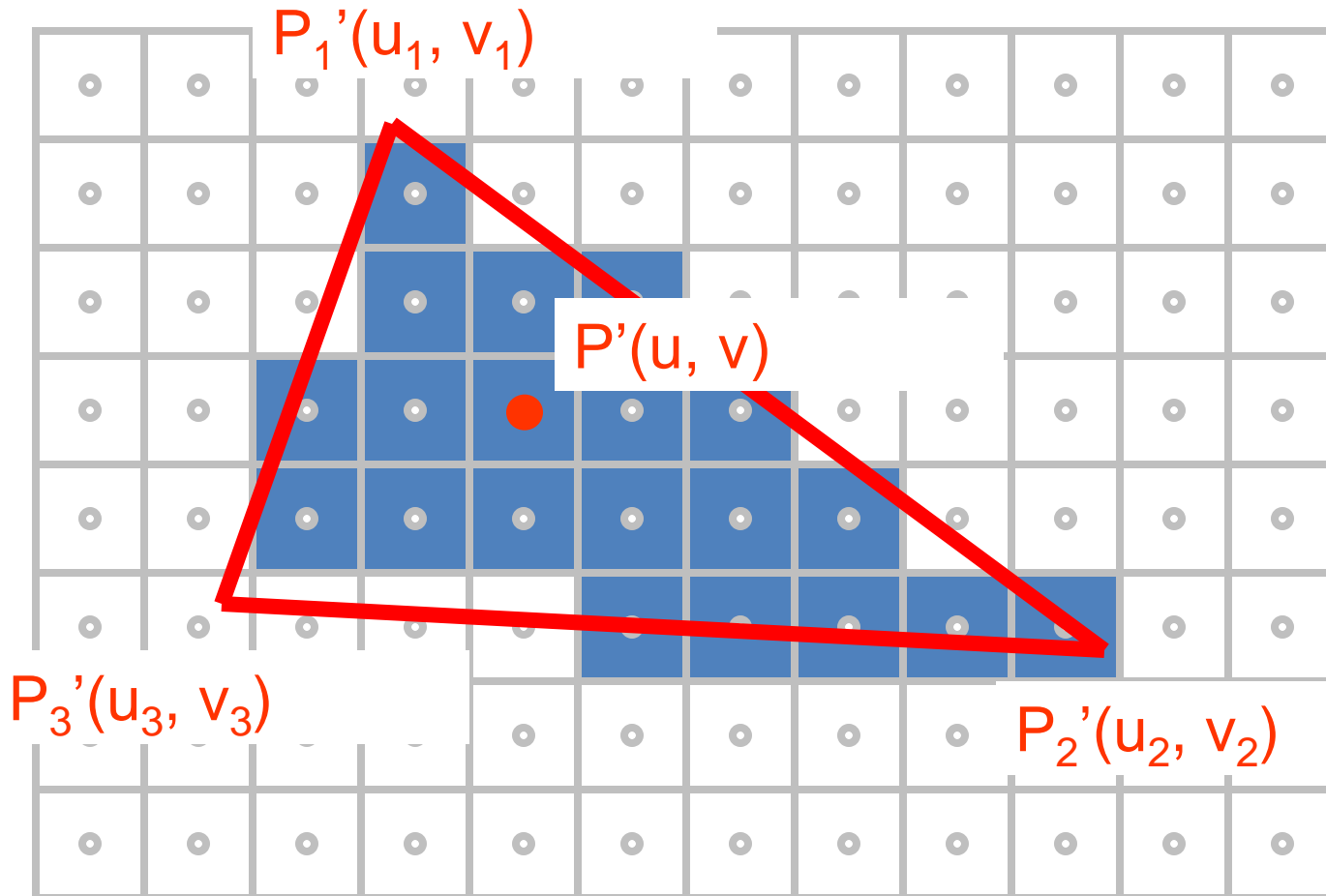
Fragment Shader

```
// Fragment shader
uniform sampler2D tex;
varying vec2 v_texCoord;

void main(void)
{
    gl_FragColor = texture2D(tex, v_texCoord);
}
```

Texture Coordinate Generation (or 2D/UV Parameterization)

- How to generate/compute texture coordinates?

Texture Coordinate Generation (or 2D/UV Parameterization)

- How to generate/compute texture coordinates?
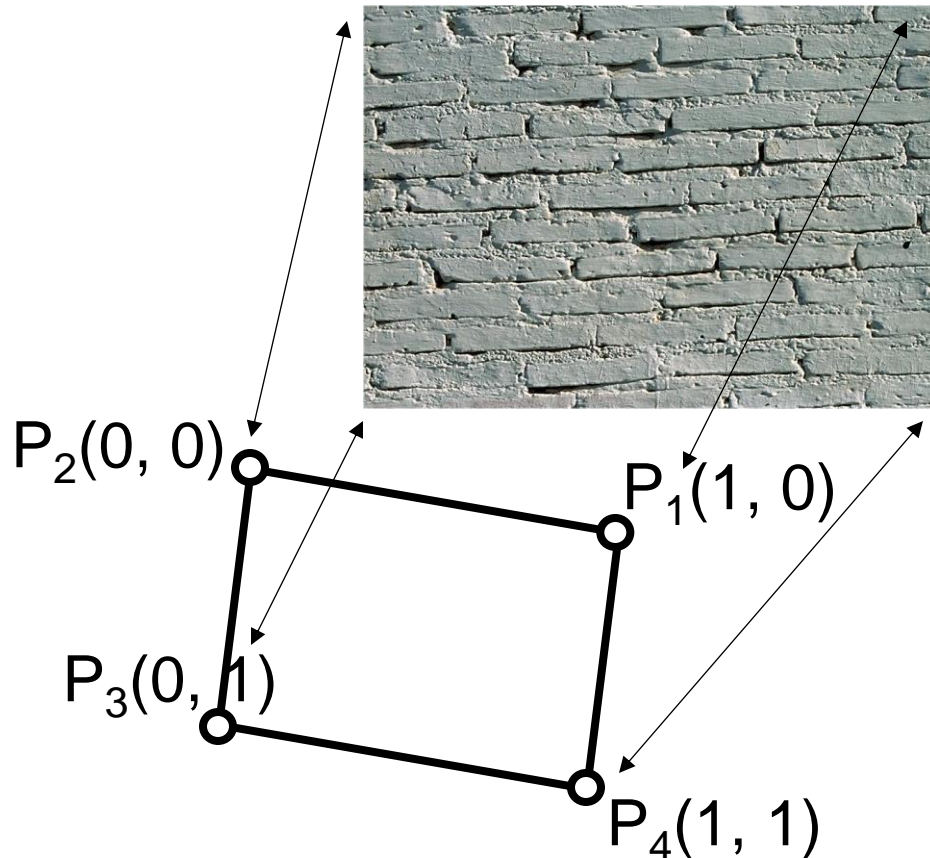

$P_2(0, 0)$

$P_1(1, 0)$

$P_3(0, 1)$

$P_4(1, 1)$

Texture Coordinate Generation (or 2D/UV Parameterization)

- How to generate/compute texture coordinates?

Planar projection

Cylindrical projection

Spherical projection

Projective texture mapping
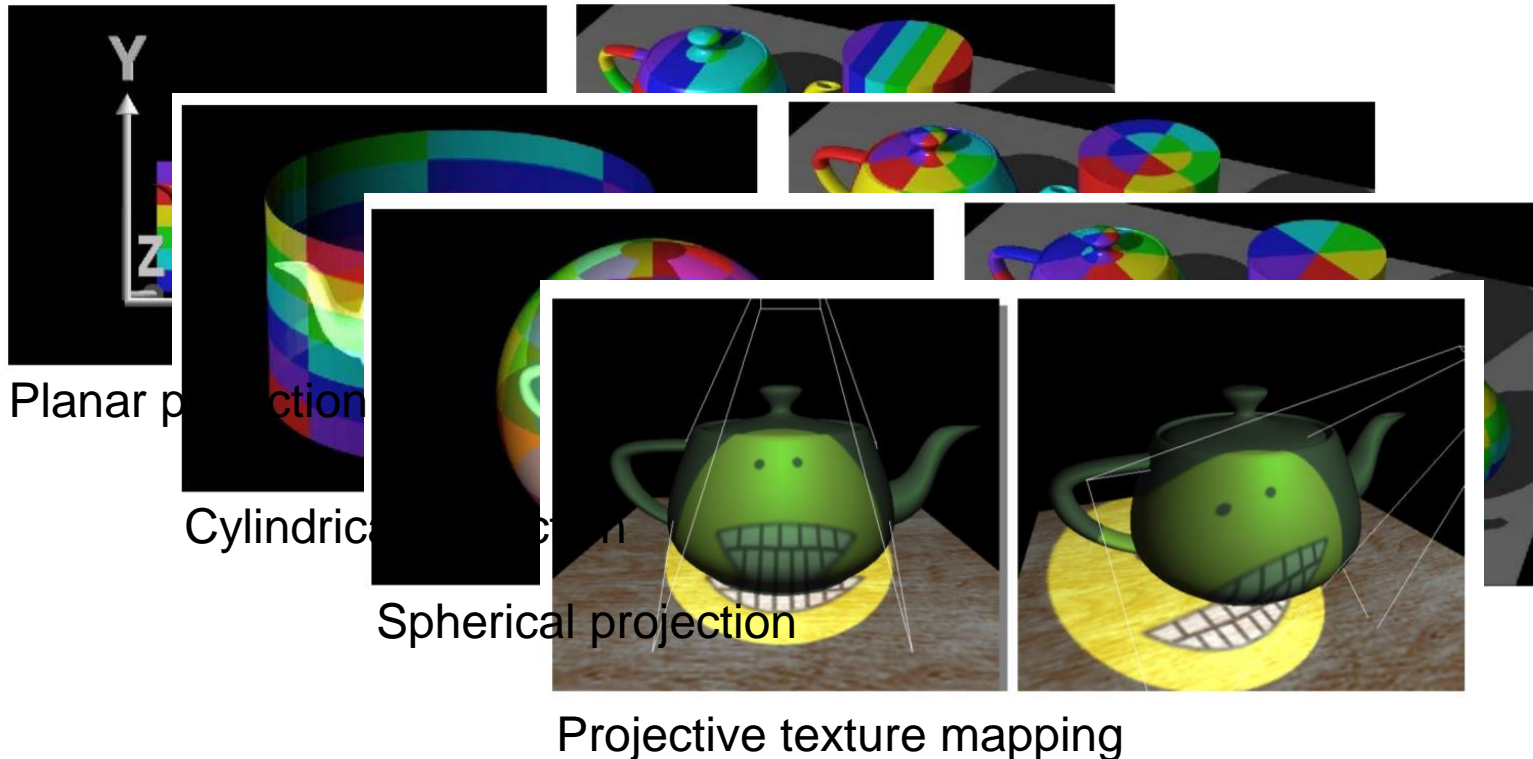
How to compute spherical projection texture coords?
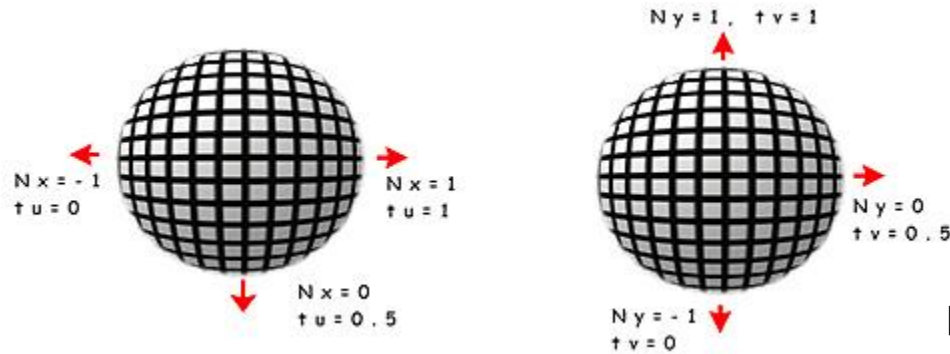
- Option A:

[images by Robert Dunlop]

$$u = \frac{\operatorname{asin}(N_x)}{\pi} + 0.5$$

$$v = \frac{\operatorname{asin}(N_y)}{\pi} + 0.5$$

How to compute spherical projection texture coords?

- Option B:
  - Compute normal as vector from center thru point

$$N = (p - c)/\|p - c\|$$

  Then, option A:

$$u = \frac{\mathrm{asin}(N_x)}{\pi} + 0.5$$

$$v = \frac{\mathrm{asin}(N_y)}{\pi} + 0.5$$

TNB Frame

- You have texture coordinates and surface normal
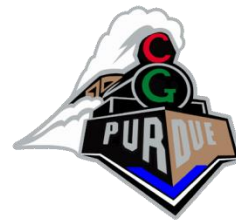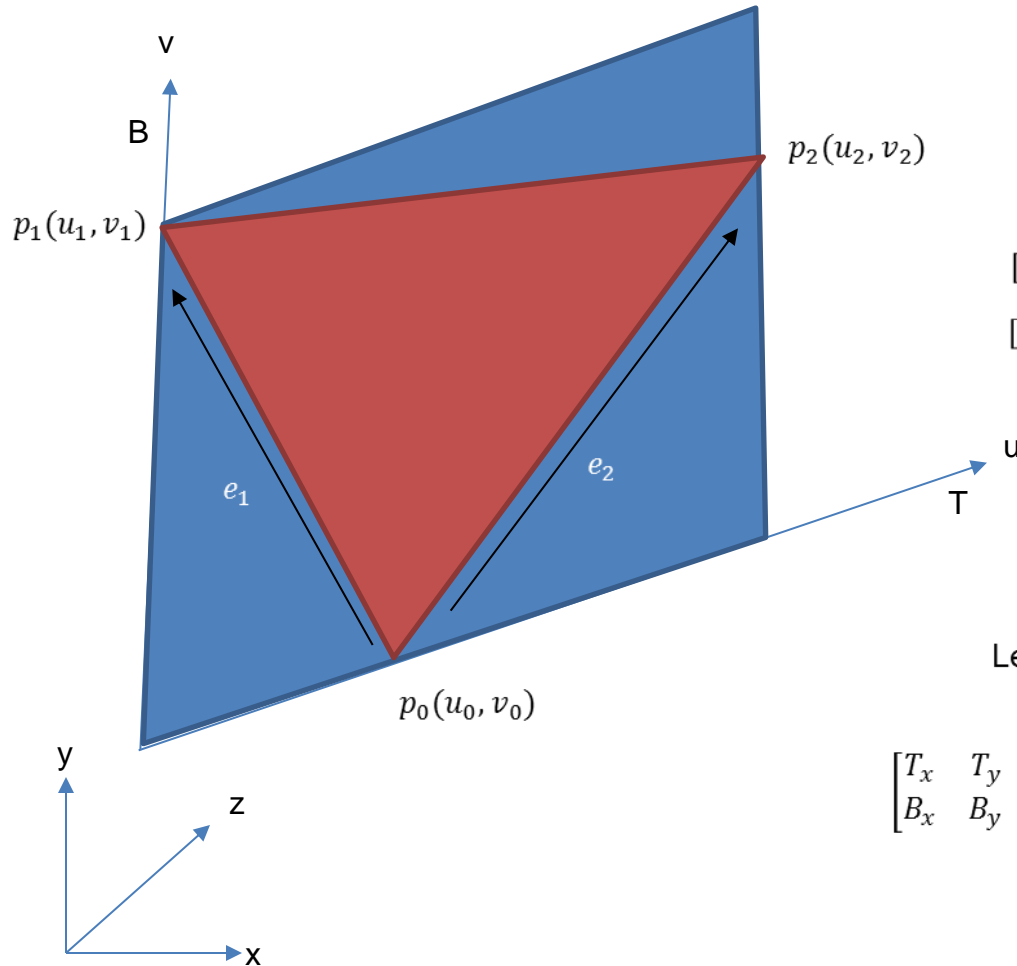- How to compute TNB frames aligned between pixels/triangles?

TNB Frame

$$e_1 = (u_1 - u_0)T + (v_1 - v_0)B$$
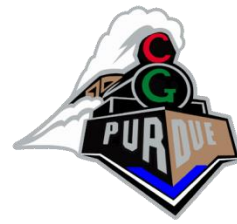$$e_2 = (u_2 - u_0)T + (v_2 - v_0)B$$

$$[e_{1x} \quad e_{1y} \quad e_{1z}] = \Delta u_1 [T_x \quad T_y \quad T_z] + \Delta v_1 [B_x \quad B_y \quad B_z]$$

$$[e_{2x} \quad e_{2y} \quad e_{2z}] = \Delta u_2 [T_x \quad T_y \quad T_z] + \Delta v_2 [B_x \quad B_y \quad B_z]$$

$$\begin{bmatrix} e_{1x} & e_{1y} & e_{1z} \\ e_{2x} & e_{2y} & e_{2z} \end{bmatrix} = \begin{bmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

Left multiply by inverse of $\begin{bmatrix} \Delta u_1 & \Delta v_1 \\ \Delta u_2 & \Delta v_2 \end{bmatrix}$ and obtain T and B:

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix} = \frac{1}{\Delta u_1 \Delta v_2 - \Delta u_2 \Delta v_1} \begin{bmatrix} \Delta v_2 & -\Delta v_1 \\ -\Delta u_2 & \Delta u_1 \end{bmatrix} \begin{bmatrix} e_{1x} & e_{1y} & e_{1z} \\ e_{2x} & e_{2y} & e_{2z} \end{bmatrix}$$

Adjacent TNB Frames

- For neighboring triangles with slightly different normals and tangent values, the TNB frames differ a small amount

- One option is to average the neighboring "tangent" vectors

- However, this might make the TNB frame no longer orthogonal

- Solution?
    - We can re-orthogonalize using a simplied version of Gram-Schmidt orthogonalization:
    - Given T and N
    - $T = normalize\,(T - (T \cdot N)N)$
    - $B = N \times T$

Fragment Shader Simple

```glsl
vec3 CalcBumpedNormal()
{
    // grab a copy of the TNB computed as described in previous slides
    vec3 Normal = normalize(Normal0);
    vec3 Tangent = normalize(Tangent0);
    vec3 Bitangent = normalize(Bitangent0);

    vec3 BumpMapNormal = texture(gNormalMap, TexCoord0).xyz;
    BumpMapNormal = 2.0 * BumpMapNormal - vec3(1.0, 1.0, 1.0);
    vec3 NewNormal;
    mat3 TBN = mat3(Tangent, Bitangent, Normal);
    NewNormal = TBN * BumpMapNormal;
    NewNormal = normalize(NewNormal);
    return NewNormal;
}

void main()
{
    vec3 Normal = CalcBumpedNormal();
    ...
```

Fragment Shader

```
vec3 CalcBumpedNormal()
{
    // grab a copy of the TN computed as described in previous slides
    vec3 Normal = normalize(Normal0);
    vec3 Tangent = normalize(Tangent0);
    Tangent = normalize(Tangent - dot(Tangent, Normal) * Normal);   // re-orthogonalize
    vec3 Bitangent = cross(Tangent, Normal); // we don't actually need to use the precomputed Bitangent0

    vec3 BumpMapNormal = texture(gNormalMap, TexCoord0).xyz;
    BumpMapNormal = 2.0 * BumpMapNormal - vec3(1.0, 1.0, 1.0);
    vec3 NewNormal;
    mat3 TBN = mat3(Tangent, Bitangent, Normal);
    NewNormal = TBN * BumpMapNormal;
    NewNormal = normalize(NewNormal);
    return NewNormal;
}

void main()
{
    vec3 Normal = CalcBumpedNormal();
    ...
```

Displacement Mapping

- Bump mapping
  - can be at pixel level
  - has no geometry/shape change
- Displacement Mapping
  - Actually modify the surface geometry (vertices)
  - re-calculate the normals
  - Can include bump mapping

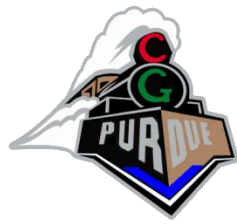

Bump Mapping

Displacement Mapping

Displacement Mapping

- Bump mapped normals are inconsistent with actual geometry. No shadow.
- Displacement mapping affects the surface geometry
  - Texture stores "offset along the normal"
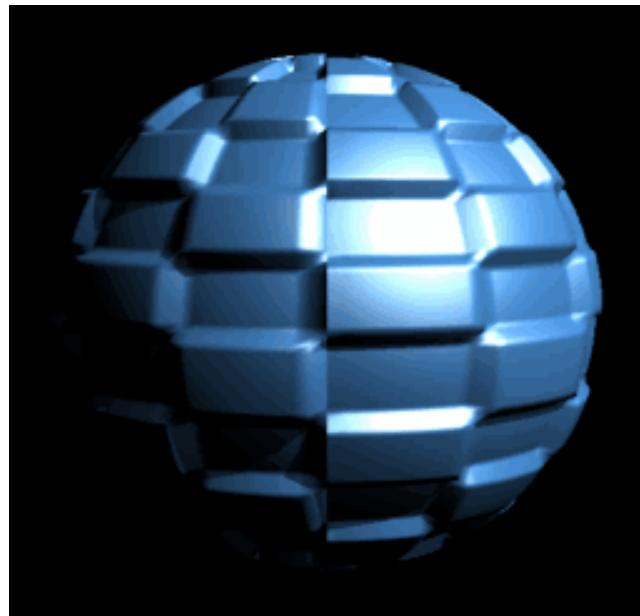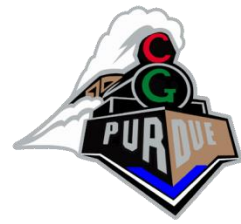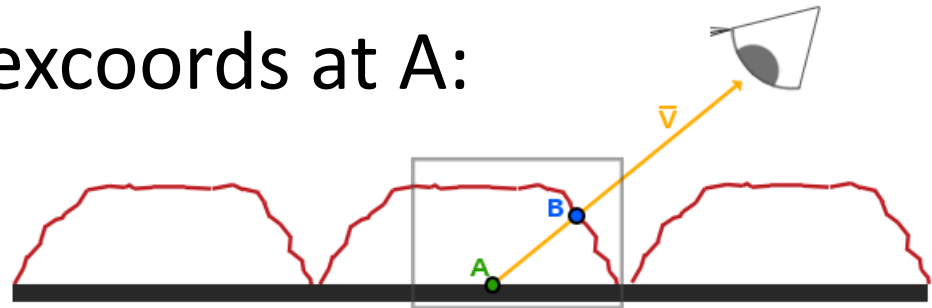
Short Video with Cool Music

- [https://www.youtube.com/watch?v=1mdR2i mNeZI](https://www.youtube.com/watch?v=1mdR2imNeZI)

Even More…

- Parallax Mapping
  - Offsets texture coordinates
  - https://www.youtube.com/watch?v=6PpWqUqeqeQ
  - Improvements: Steep Parallax Mapping, Parallax Occlusion Mapping

- Relief Mapping
  - Offsets heights to recompute normals
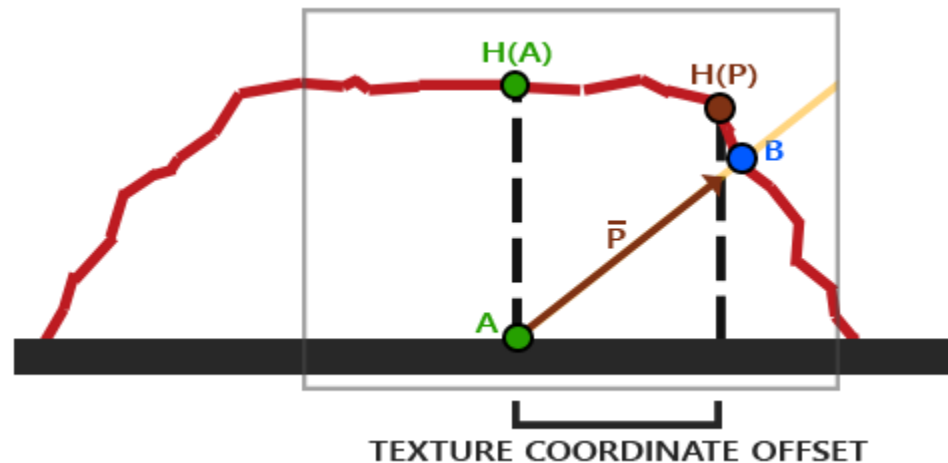 - https://www.youtube.com/watch?v=_erYebogWUw
  - https://www.youtube.com/watch?v=5gorm90TXJM

Parallax Mapping (briefly)

- Normally, you use texcoords at A:

- Instead, we want to walk-back by P to find B texcoords:

[see https://learnopengl.com/Advanced-Lighting/Parallax-Mapping]
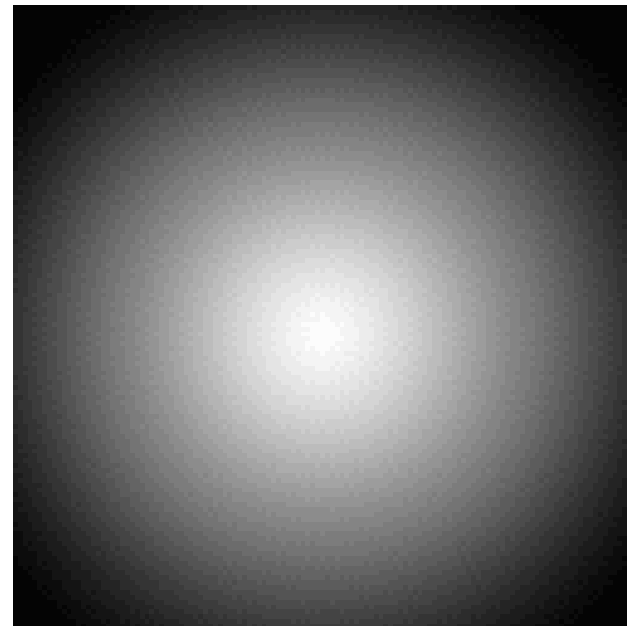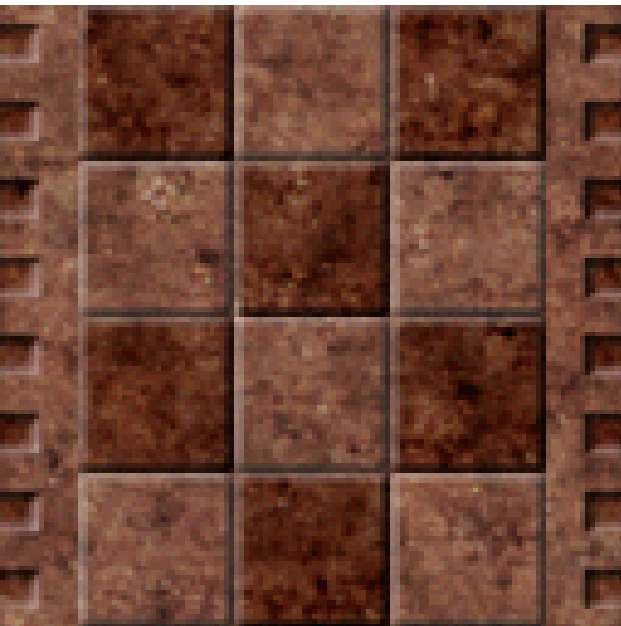
Light Mapping

- Pre-render special lighting effects
- Multi-texturing idea: arbitrary texel-by-texel shading calc'd from multiple texture maps
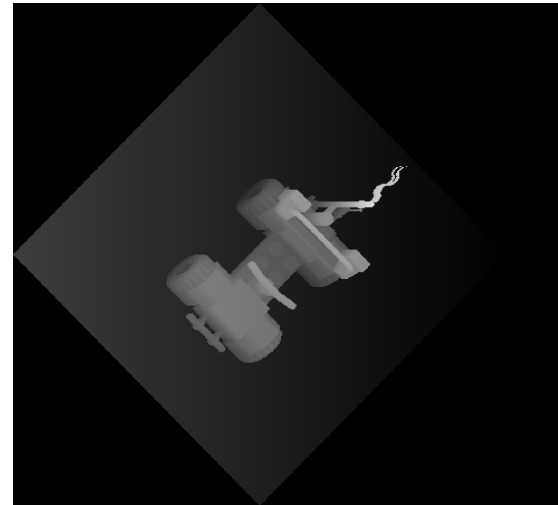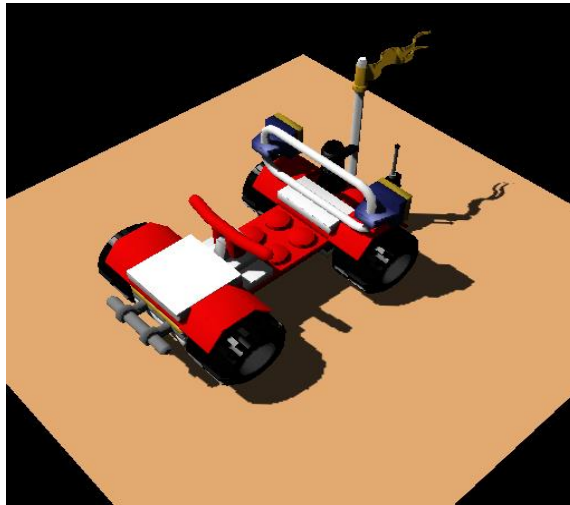

Reflectance Texture $\times$ **Light Map** (Illumination Texture) $=$ Display texture
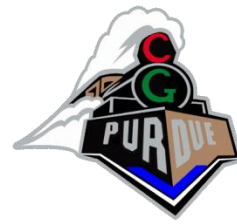
Shadow Mapping

- Render scene from light's point of view
 - Store depth of each pixel

Shadow Mapping

- Render scene from light's point of view
  - Store depth of each pixel
  - From light's point of view, any pixel blocked is in the shadow.
- When shading a surface:
  - Transform surface pixel into light coordinates
  - Compare current surface depth to stored depth. If depth > stored depth, the pixel is in shadow; otherwise pixel is lit
  - Note: can be very expensive timewise…

Resolution Problem:
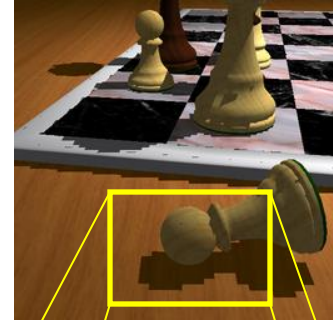
single shadow map pixel

What can be done?

Higher resolution helps but does not solve…