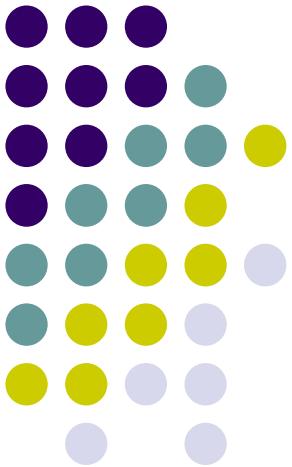
Computer Graphics (CS 563)
Lecture 2: Advanced Computer Graphics
Part 2: Texturing

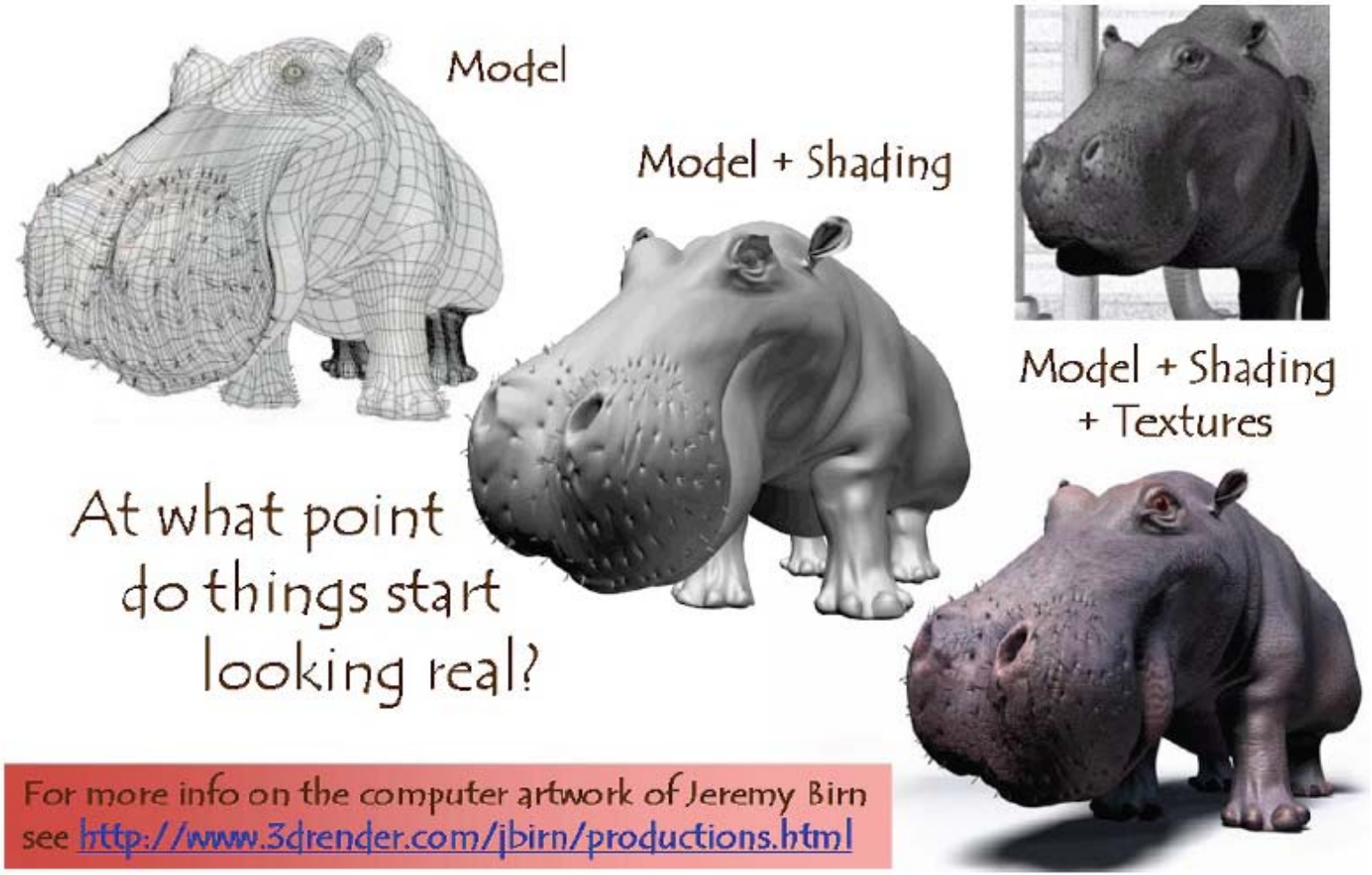
Prof Emmanuel Agu

Computer Science Dept.

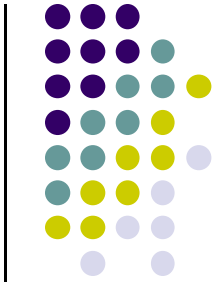
Worcester Polytechnic Institute (WPI)



Texturing: The Quest for Realism



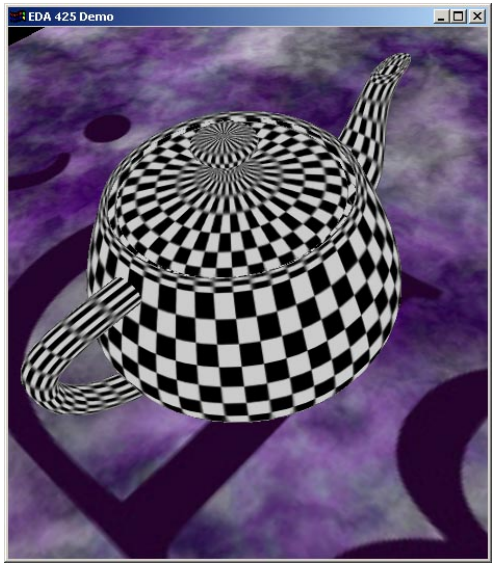
Texturing: Glue n-dimensional images onto geometrical objects



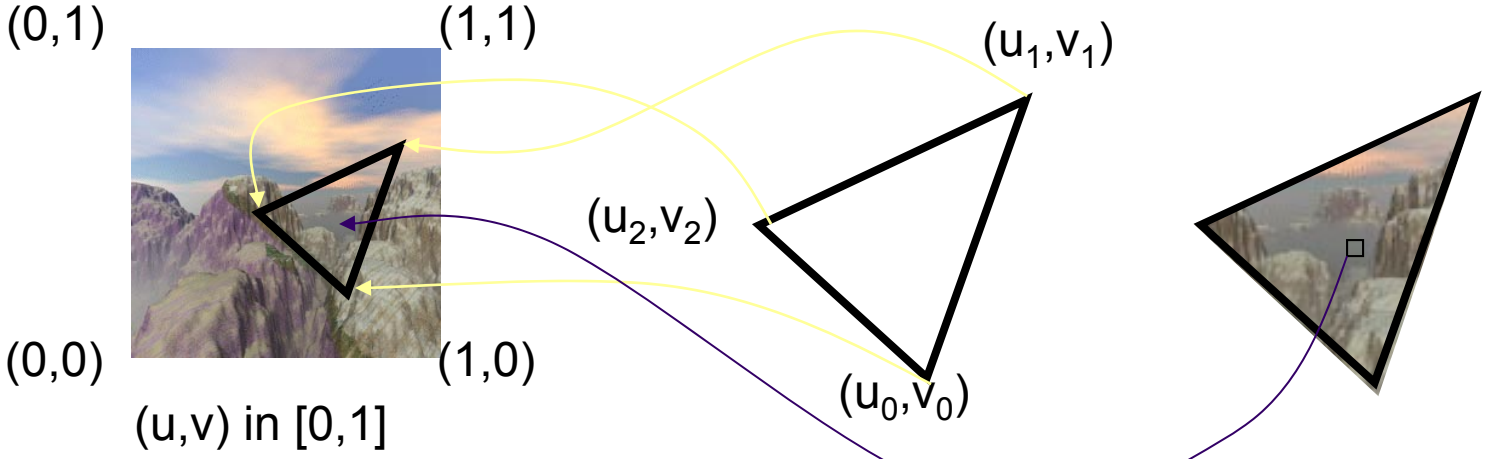
- Purpose: more realism, and this is a cheap way to do it
  - Bump mapping
  - Also environment mapping, other effects

+

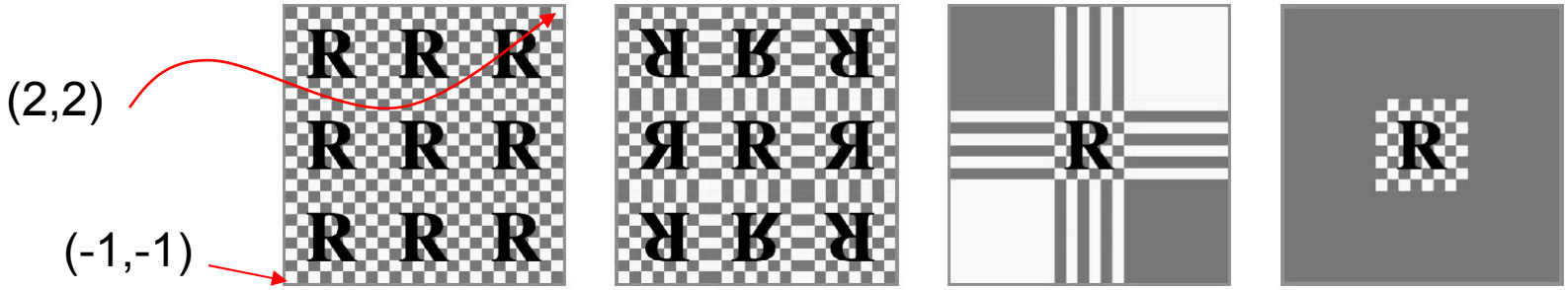
=



Texture coordinates



- What if (u,v) >1.0 or <0.0 ?
- To repeat textures, use just the fractional part
  - Example: 5.3 -> 0.3
- Repeat, mirror, clamp, border:

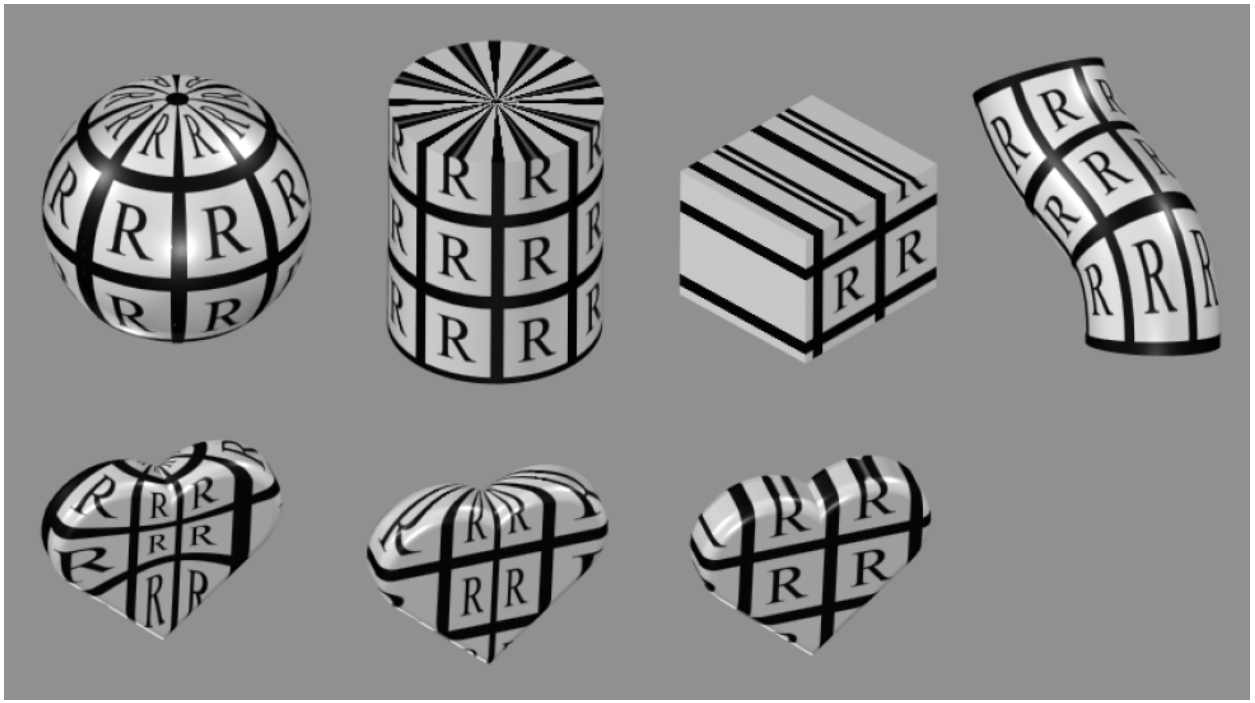


Projector Function

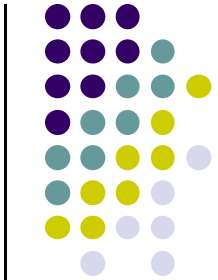
- Different ways to project texture onto objects

  L-R: spherical, cylinderical, planar and natural

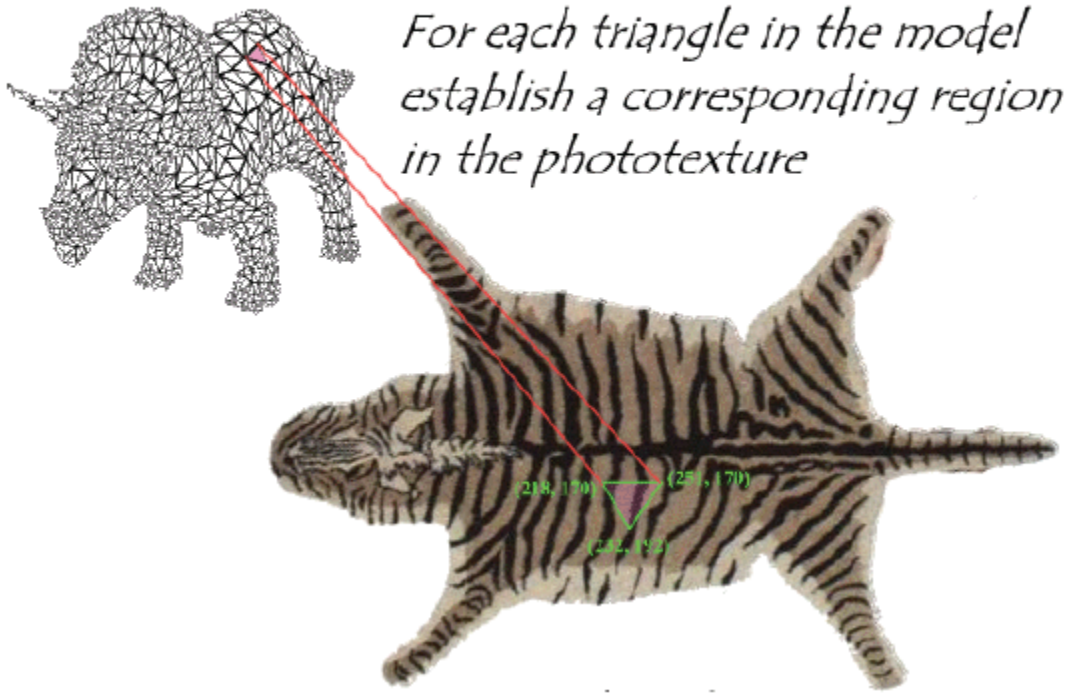
- Natural? Some shapes have defined mappings (e.g. parametric curved surfaces



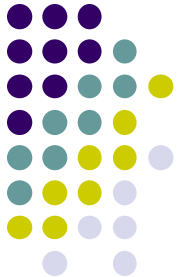
Hot Research Topic: Parametrization



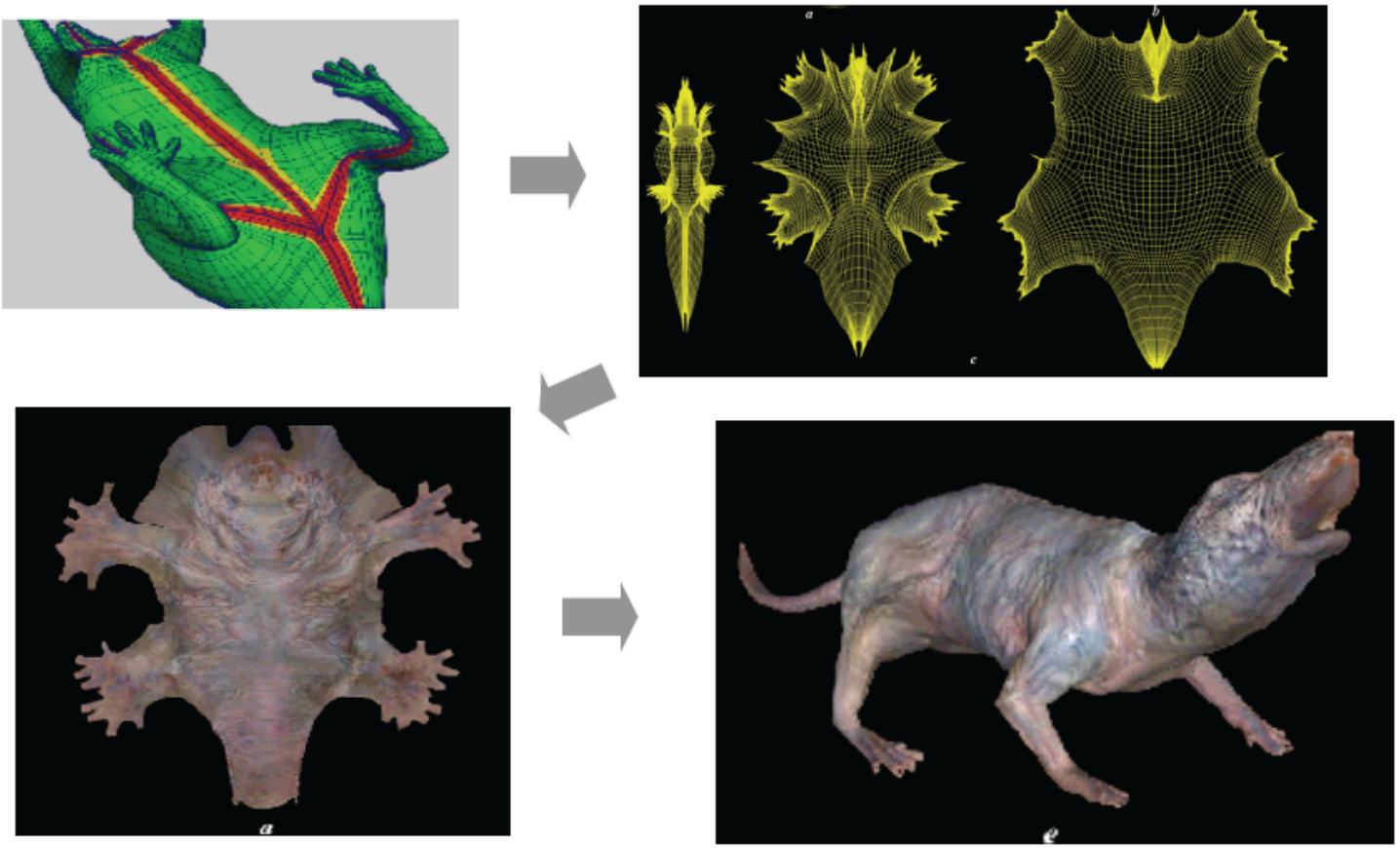
- The concept is very simple: define a mapping from the surface to the plane



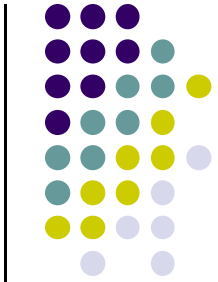
Parametrization in Practice



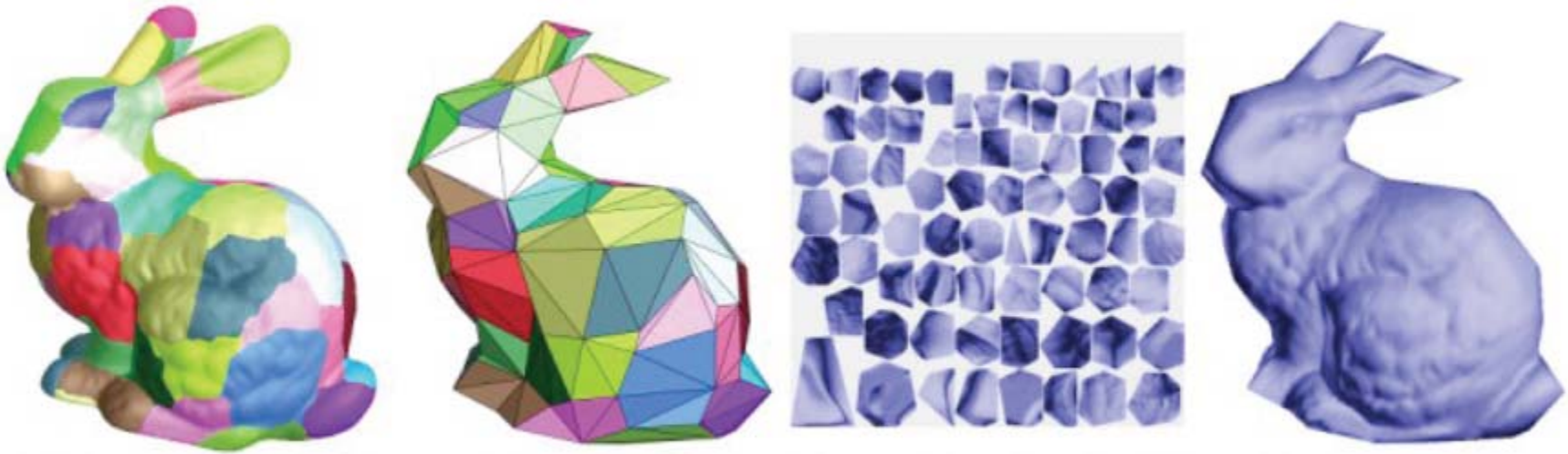
- Texture creation and parametrization is an art form
- Option: Unfold the surface



Parametrization in Practice

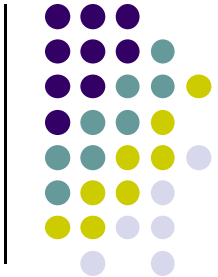


- Option: Create a Texture Atlas
- Break large mesh into smaller pieces

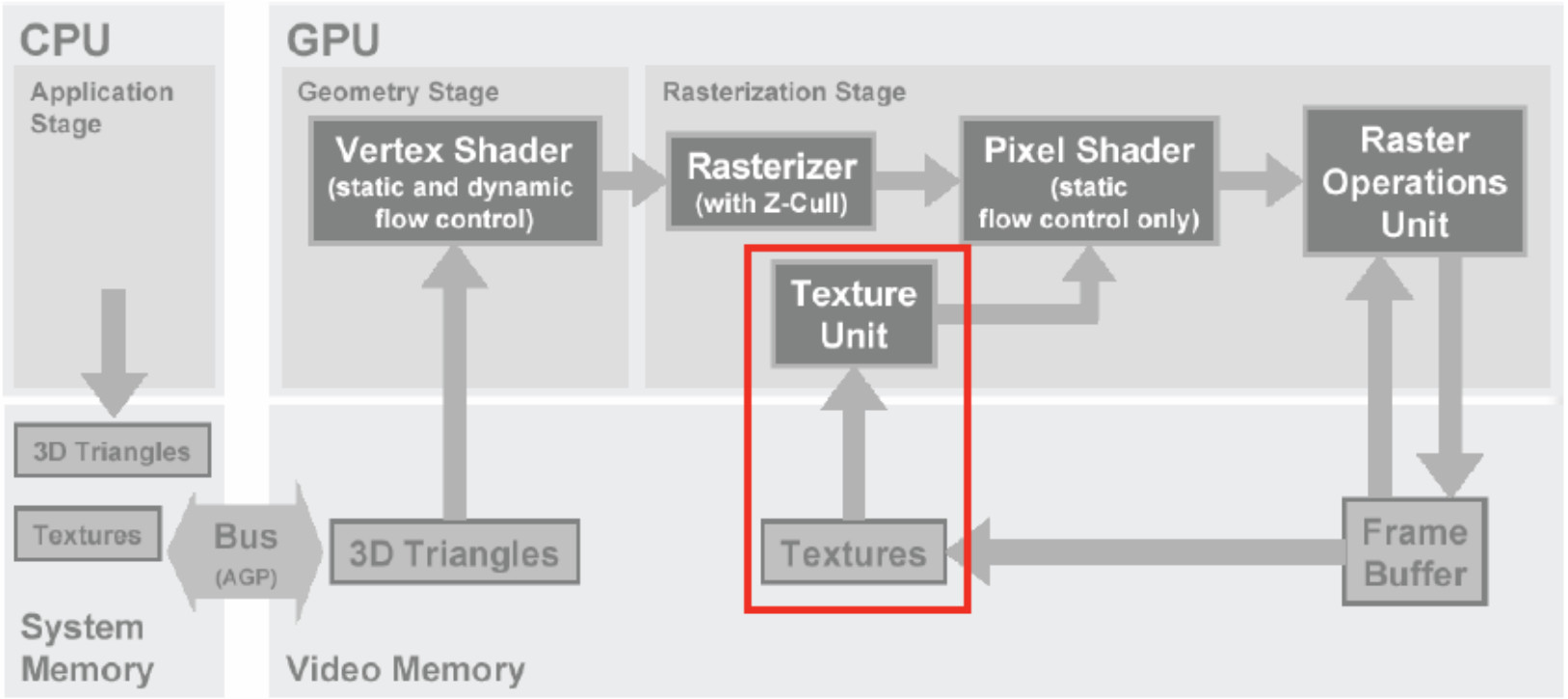


(a) charts on original mesh M (b) base mesh M' (c) texture atlas (before pull-push) (d) textured base mesh

Texturing Hardware

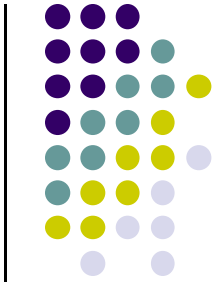


- Texture unit accessible by pixel processor



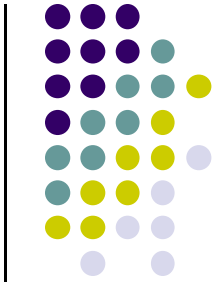
- **Note:** Textures also accessible from vertex shader on new hardware

Texture Magnification

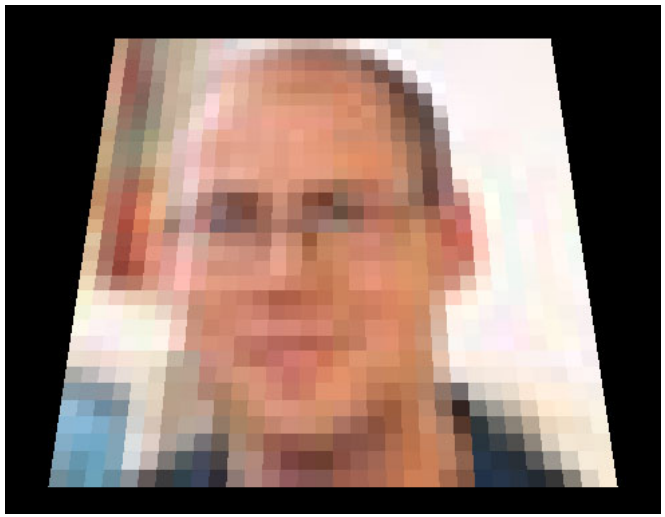


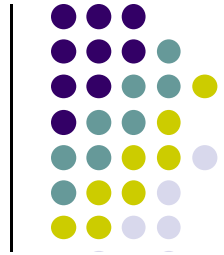
- 48 x 48 image projected (stretched) onto 320 x 320 pixels
- **Left:** Nearest neighbor filter
- **Middle:** Bilinear filter (average of 4 nearest texels)
- **Right:** Cubic filter (weighted avg. of 5 nearest texels)

Texture magnification



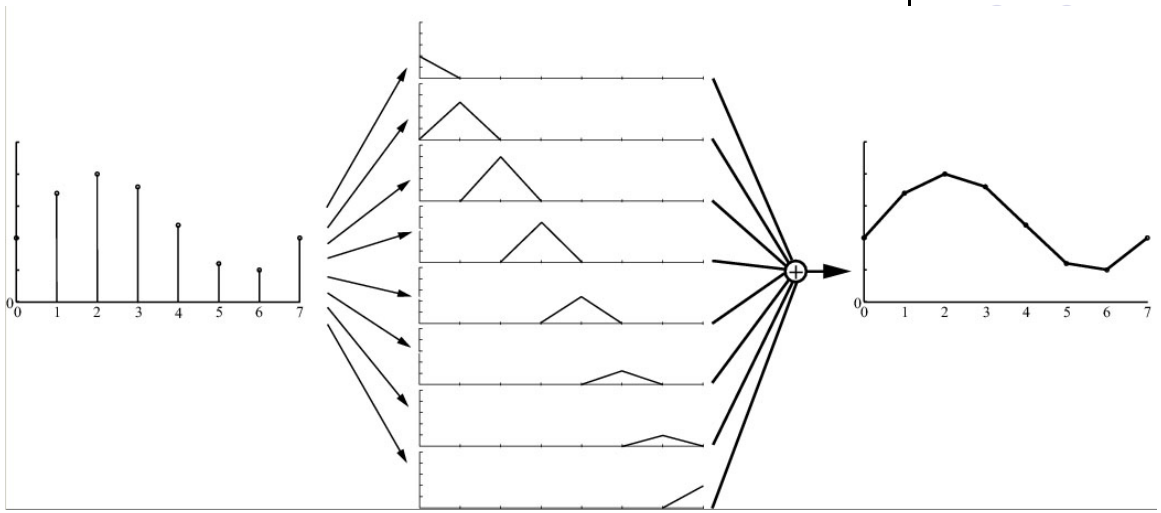
- What does the theory say?
- sinc(x) is not feasible in real time
- Box filter (nearest-neighbor) is
- Poor quality



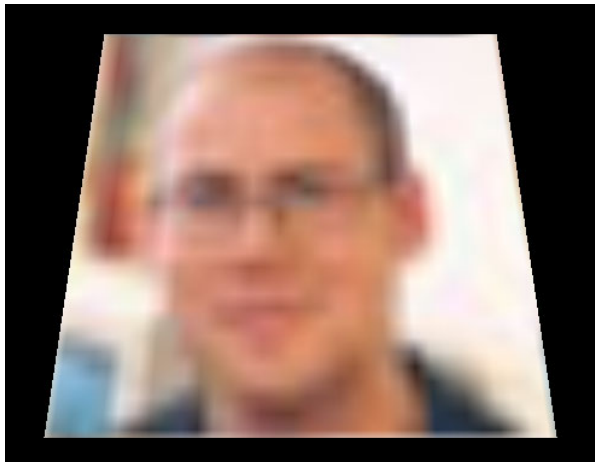


Texture magnification

- Tent filter is
- Linear

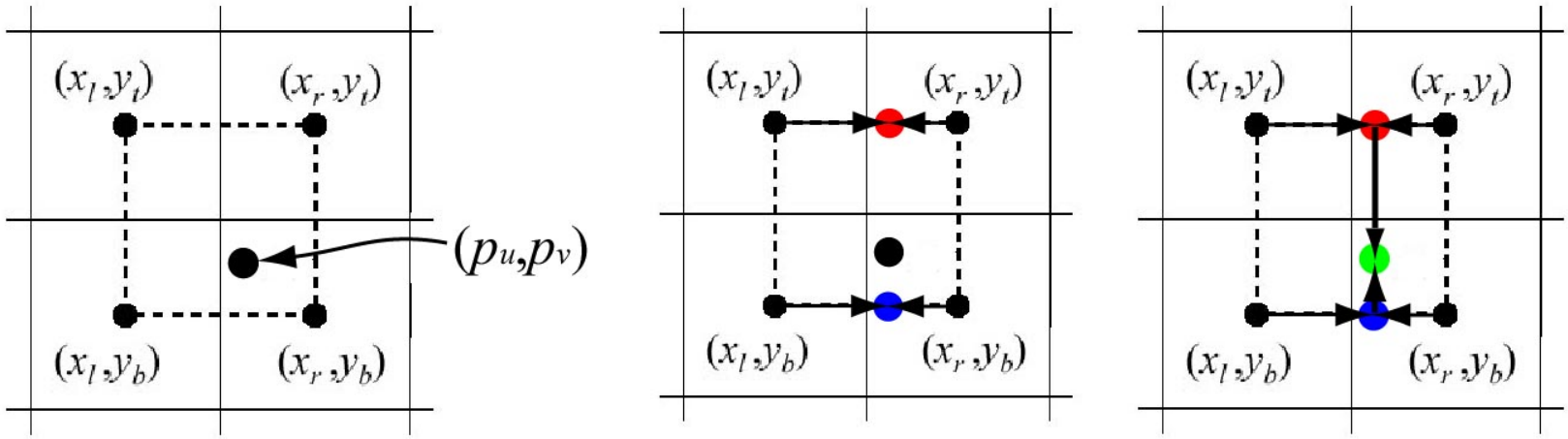


- Looks better
- Simple in 1D:
- (1-t)*color0+t*color1
- How about 2D?

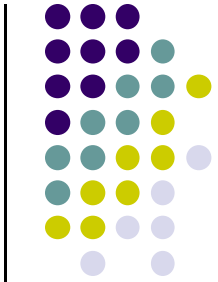


Bilinear interpolation

- Texture coordinates (p_u,p_v) in [0,1]
- Texture images size: n*m texels
- Nearest neighbor would access:
  - (floor(n*u), floor(m*v))
- Interpolate 1D in x & y



Bilinear interpolation

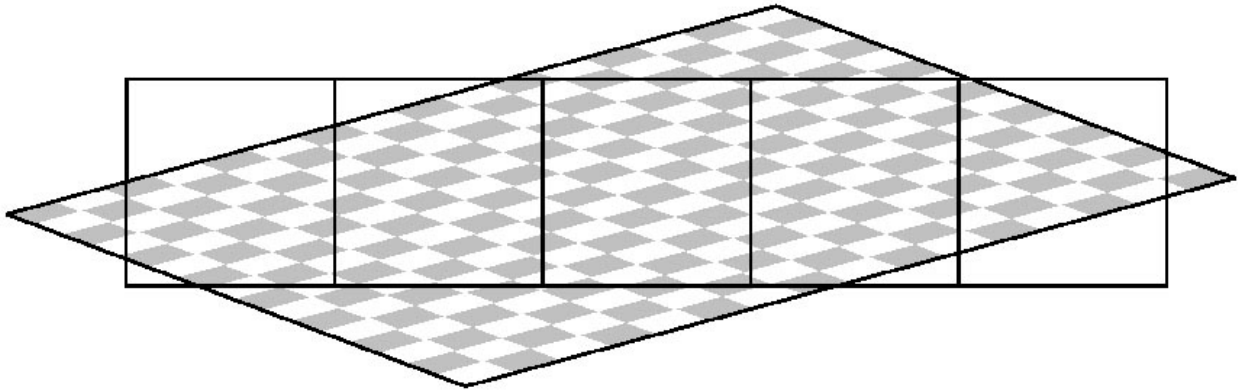


- Check out this formula at home
- $\mathbf{t}(u,v)$ accesses the texture map
- $\mathbf{b}(u,v)$ filtered texel

$$(u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor).$$

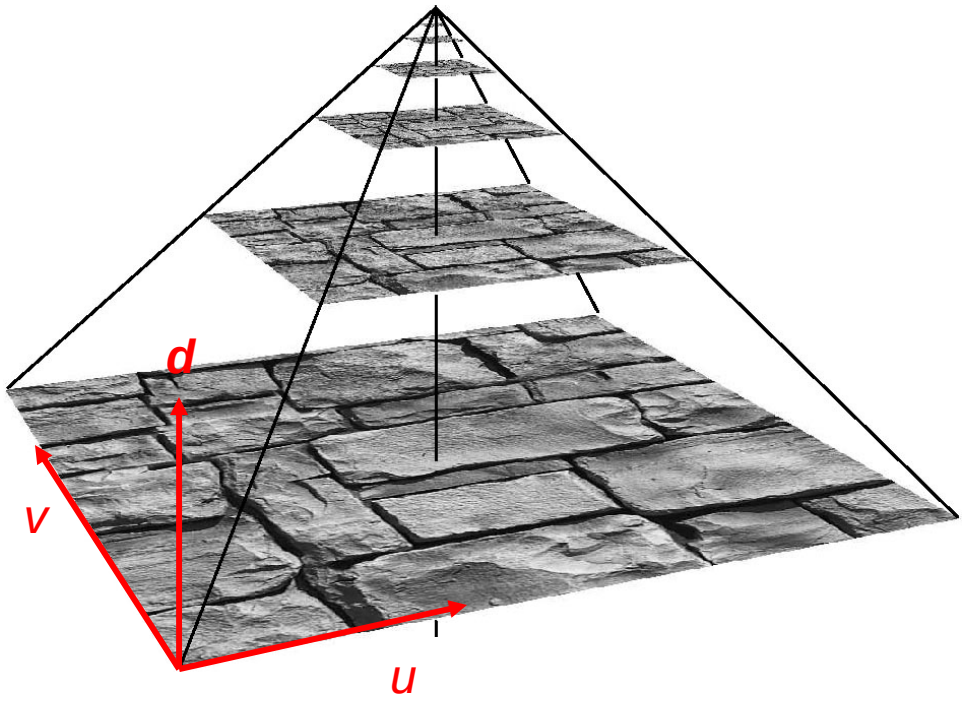
$$\begin{aligned}\mathbf{b}(p_u, p_v) =& (1-u')(1-v')\mathbf{t}(x_l, y_b) + u'(1-v')\mathbf{t}(x_r, y_b) \\ &+ (1-u')v'\mathbf{t}(x_l, y_t) + u'v'\mathbf{t}(x_r, y_t).\end{aligned}$$

Texture minification
What does a pixel ”see”?



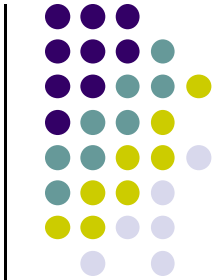
- Project large texture onto smaller group of pixels
- Theory (sinc) is too expensive
- Cheaper: average of texel inside a pixel
- Still too expensive, actually
- Use Mipmaps
  - Prefilter texture maps as shown on next slide

Mipmapping



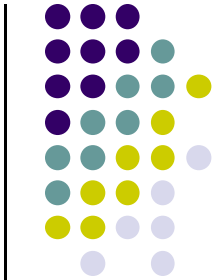
- Image pyramid
- Half width and height when going upwards
- Average over 4 ”parent texels” to form ”child texel”
- Depending on amount of minification, determine which image to fetch from

Mipmapping: Memory requirements

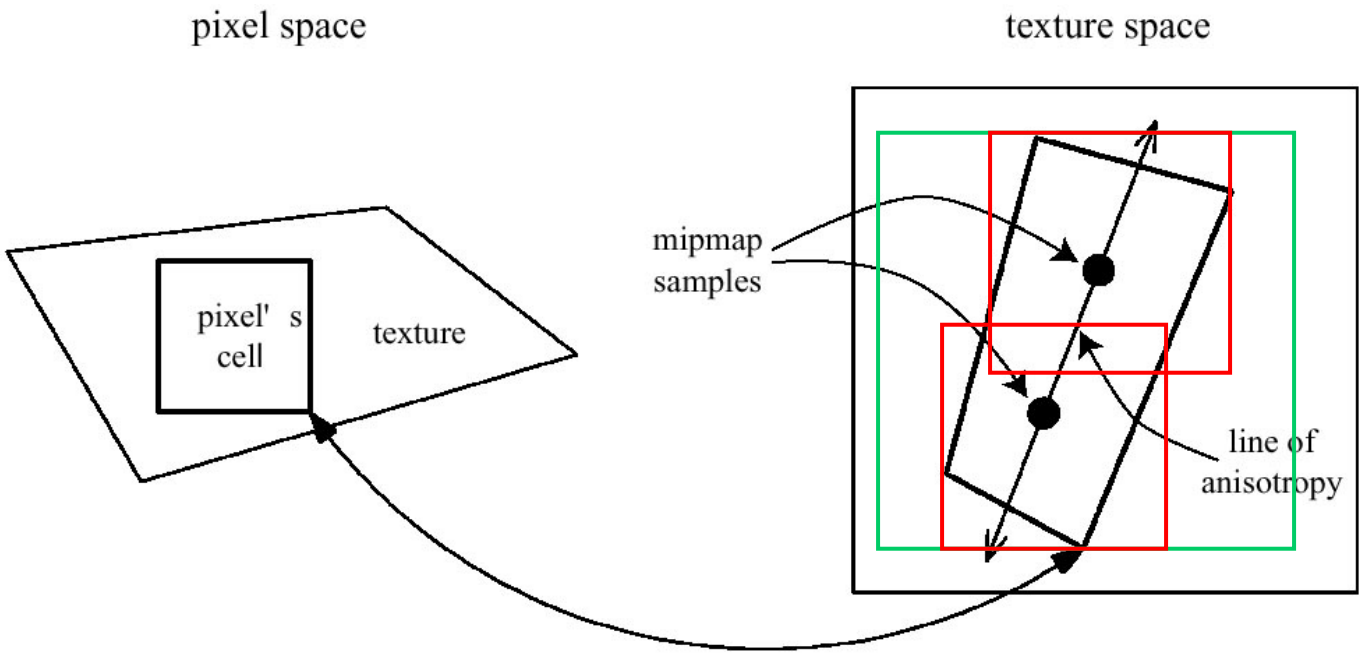


- Not twice the number of bytes…!
- Rather 33% more – not that much

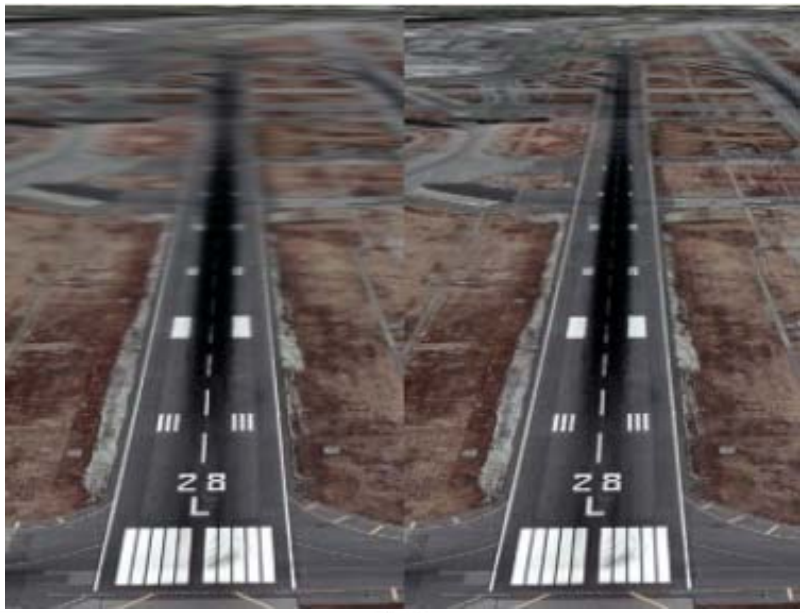
Anisotropic Texture Filtering



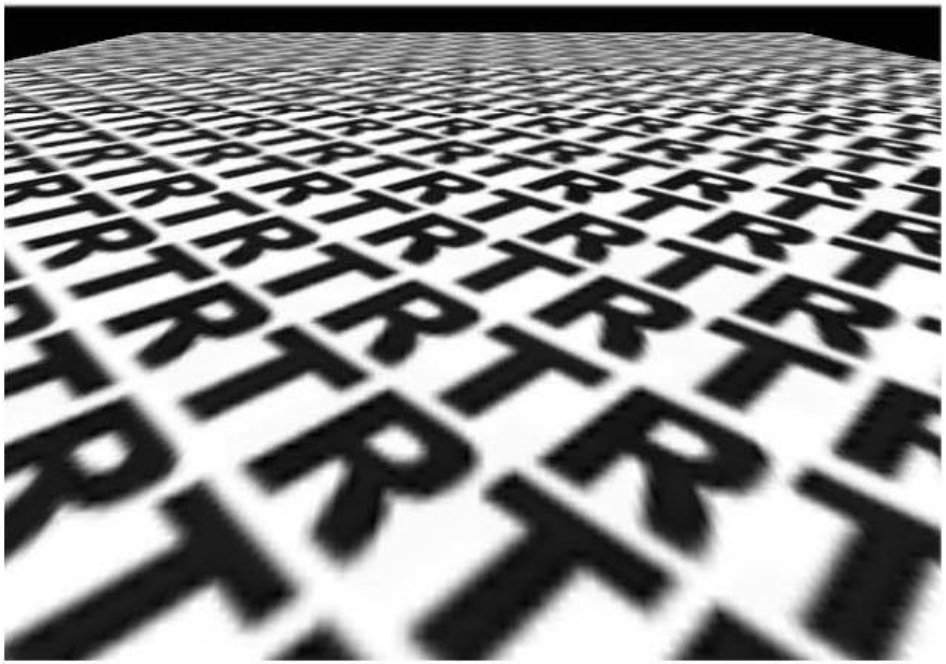
- Mipmaps look too blurry because they treat the projection of pixel as a square in texture space
- Anisotropic filtering treats it as a line segment, and takes a number of samples along it

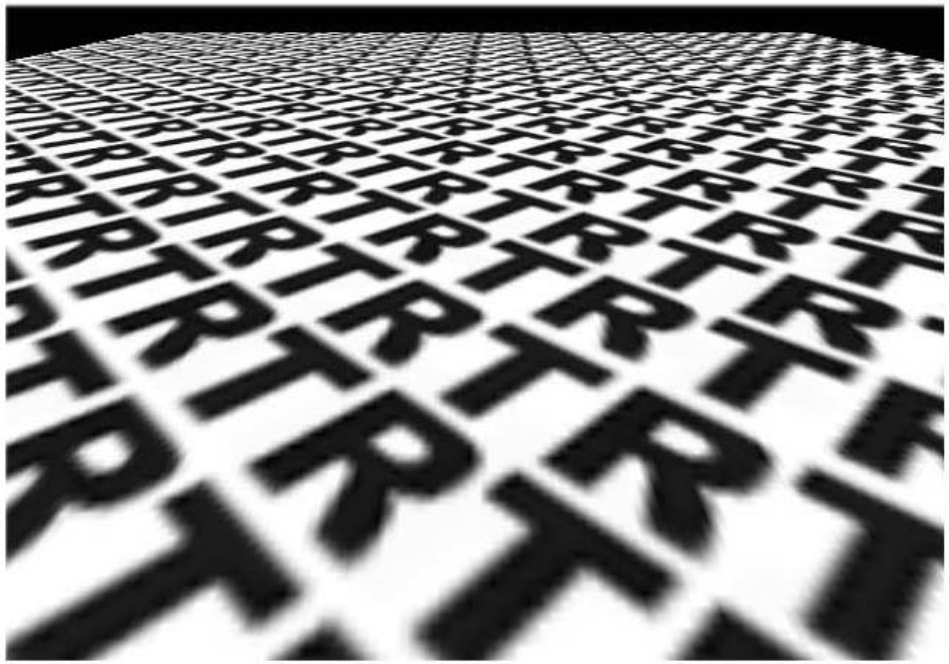


Anisotropic texture filtering

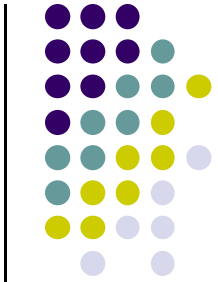


16 samples



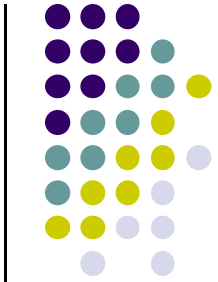


Mipmaps Vs Anisotropic Filtering



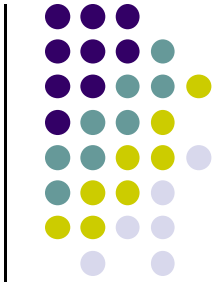
- Mipmaps:
  - require 1/3 extra memory, extremely
  - cheap to compute, very cache coherent
- Anisotropic filtering:
  - requires no extra memory,
  - much more expensive, takes several unordered samples of a larger chunk of memory

Miscellaneous



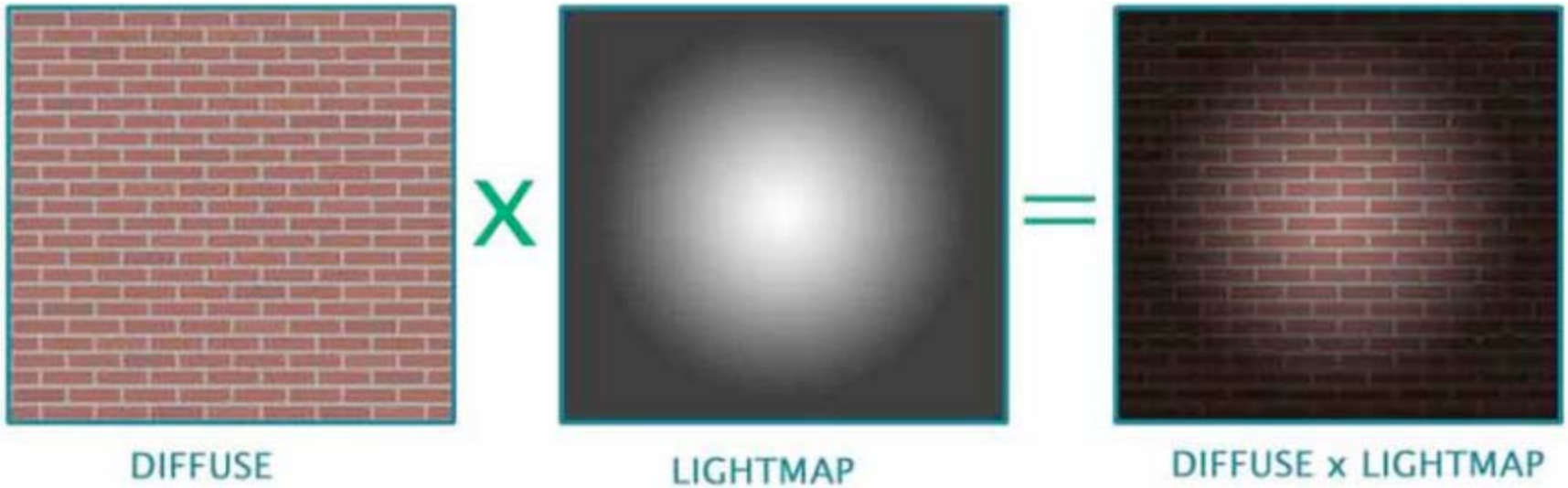
- How to apply texturing:
  - Modulate (multiply texture with lighting)
  - Replace (just use texture color)
  - Add, sub, etc (on newer hardware)
  - More in the book

Light Maps

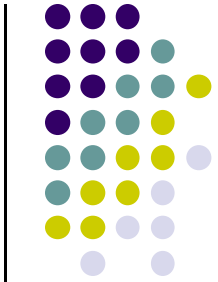


- Good shadows are complicated and expensive
- If lighting and objects will not change, neither are the shadows
- Can “bake” the shadows into a texture map as a preprocess step
- During shading, lightmap values are multiplied into resulting pixel

Light Maps

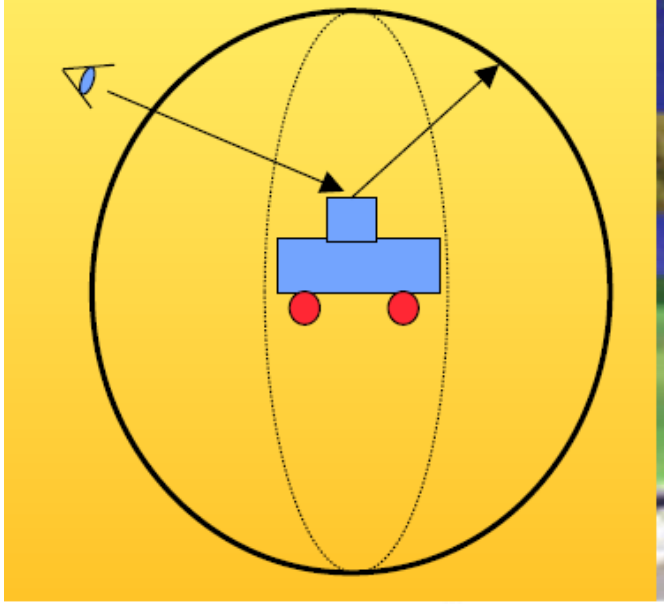


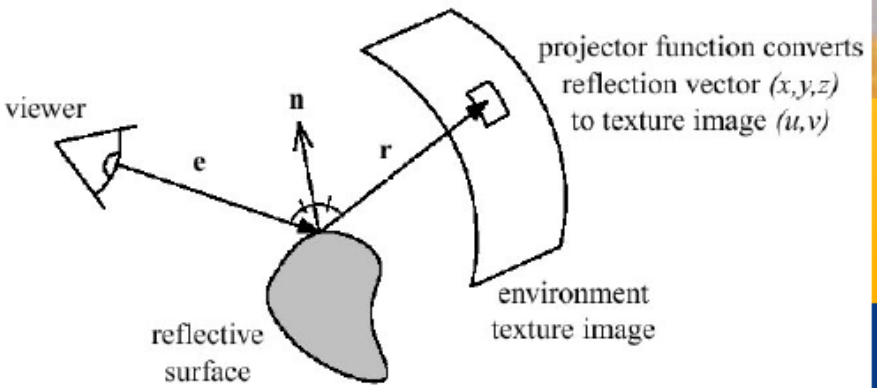
Specular Mapping



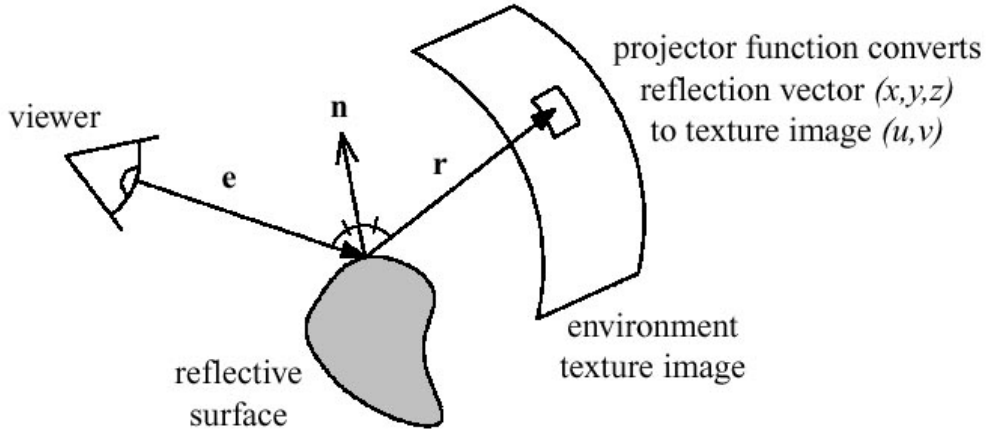
- Use a greyscale texture as a multiplier for the specular component

Environment mapping





Environment mapping



- Assumes the environment is infinitely far away
- Types of environment mapping
  - Sphere mapping
  - Cube mapping is the norm nowadays
    - Much less distortion
    - Gives better result

Sphere Map

- A sphere maps is basically a photograph of a reflective sphere in an environment

Paul DeBevec, www.debevec.org

Sphere map

- example

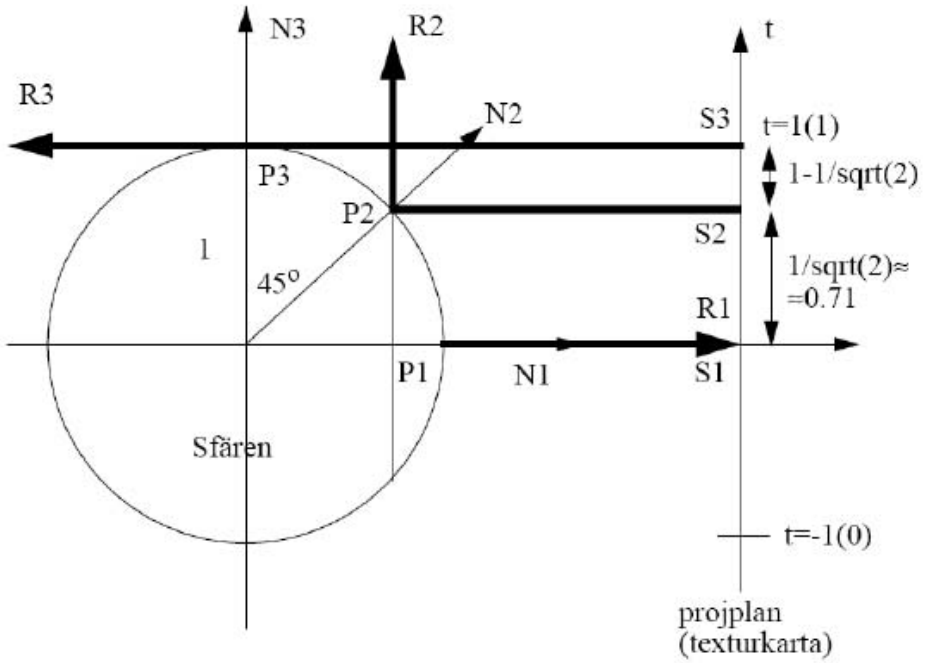
Sphere map
(texture)

Sphere map
applied on torus

Capturing a Sphere Map

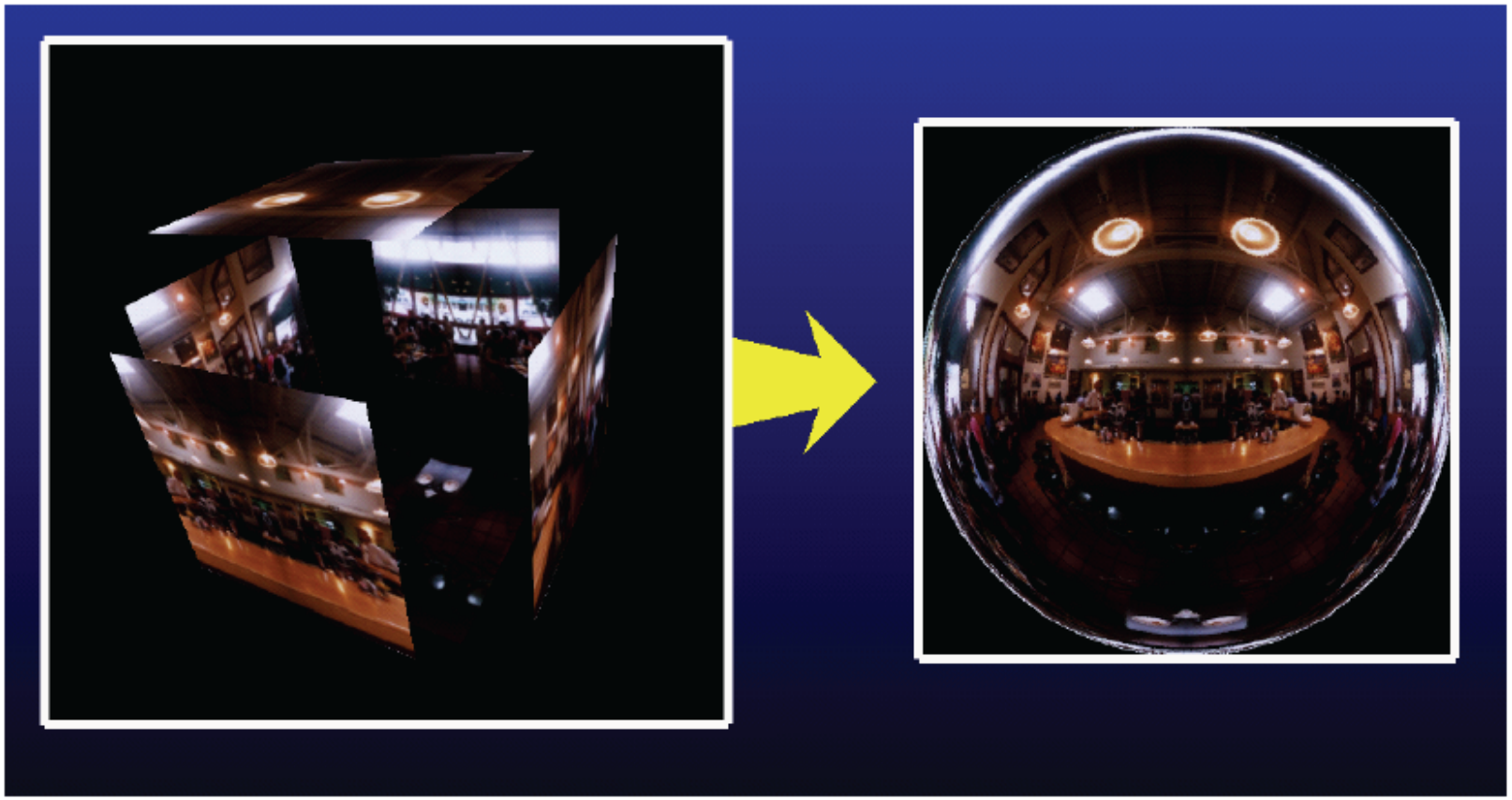
Sphere Map

- Infinitesimally small reflective sphere (infinitely far away)
  - i.e., orthographic view of a reflective unit sphere
- Create by:
  - Photographing metal sphere
  - Ray tracing
  - Transforming cube map to sphere map



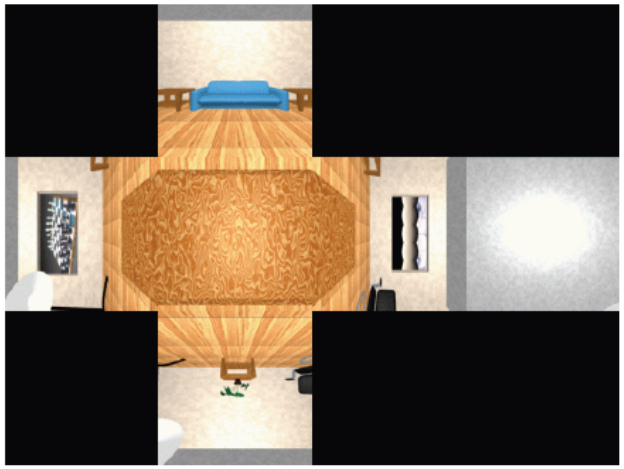
Cube Environment Map

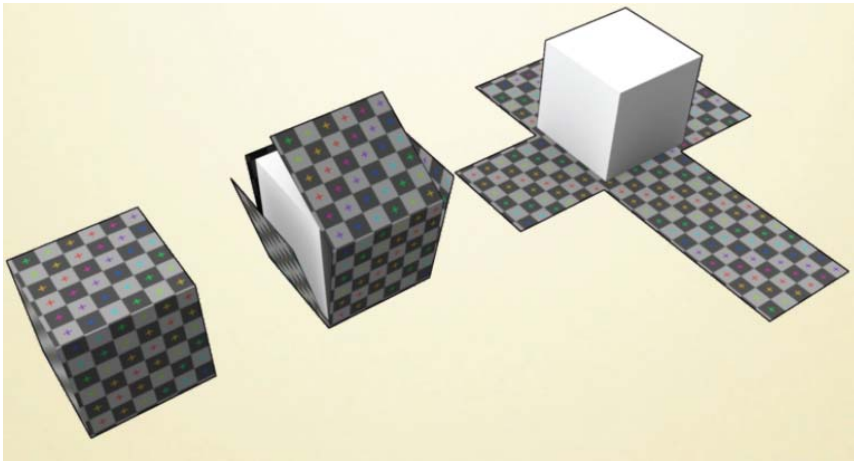
- The sphere can be replaced by a cube
- Simplifies the computations



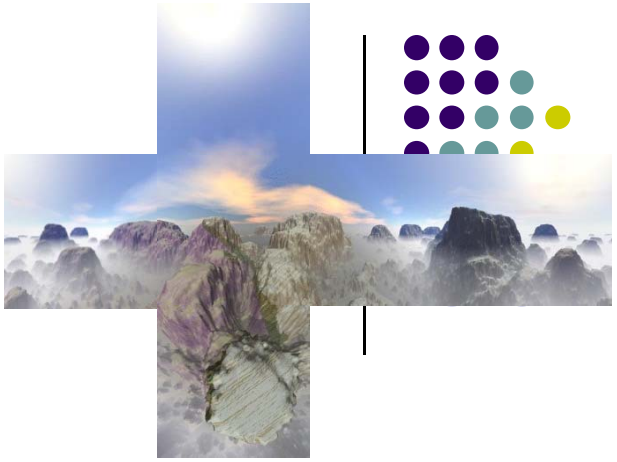
Cube Environment Map Example

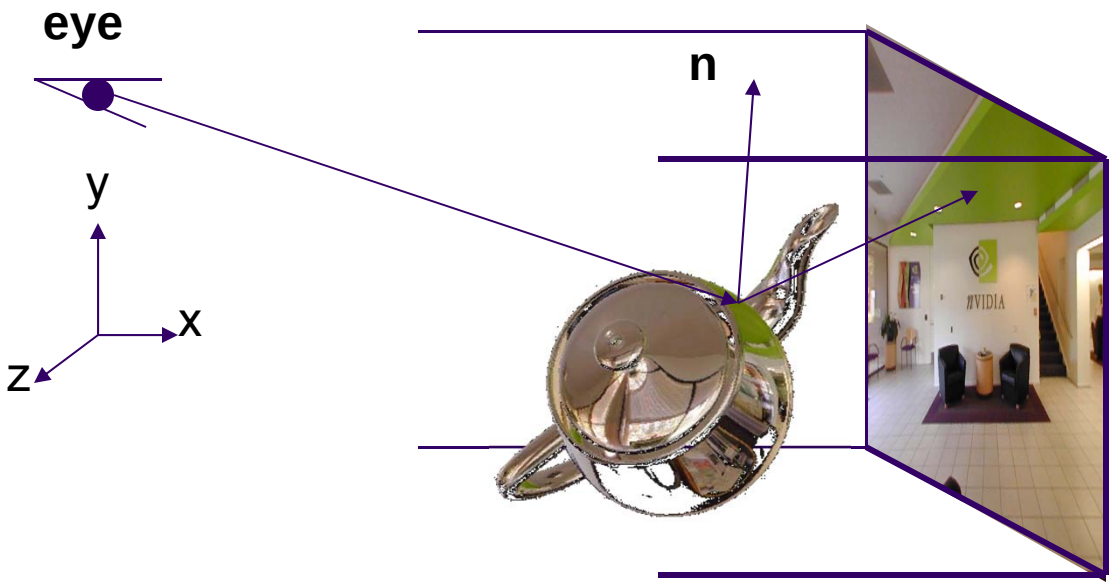
- Use six textures, one for each face of a cube that surrounds the object





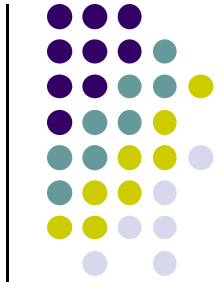
Cube mapping



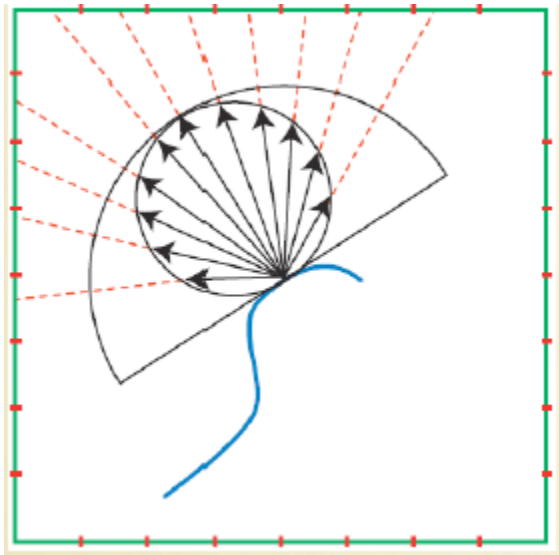


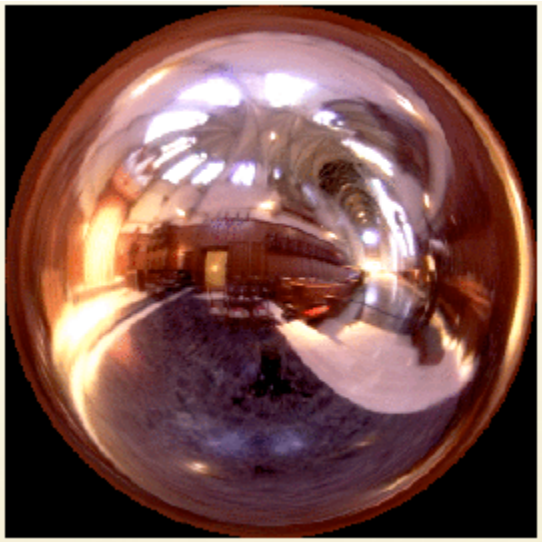
- Need to compute reflection vector, $\mathbf{r}$
- Use $\mathbf{r}$ by for lookup
- If hardware supports cube maps, then it does all the work

Irradiance Mapping

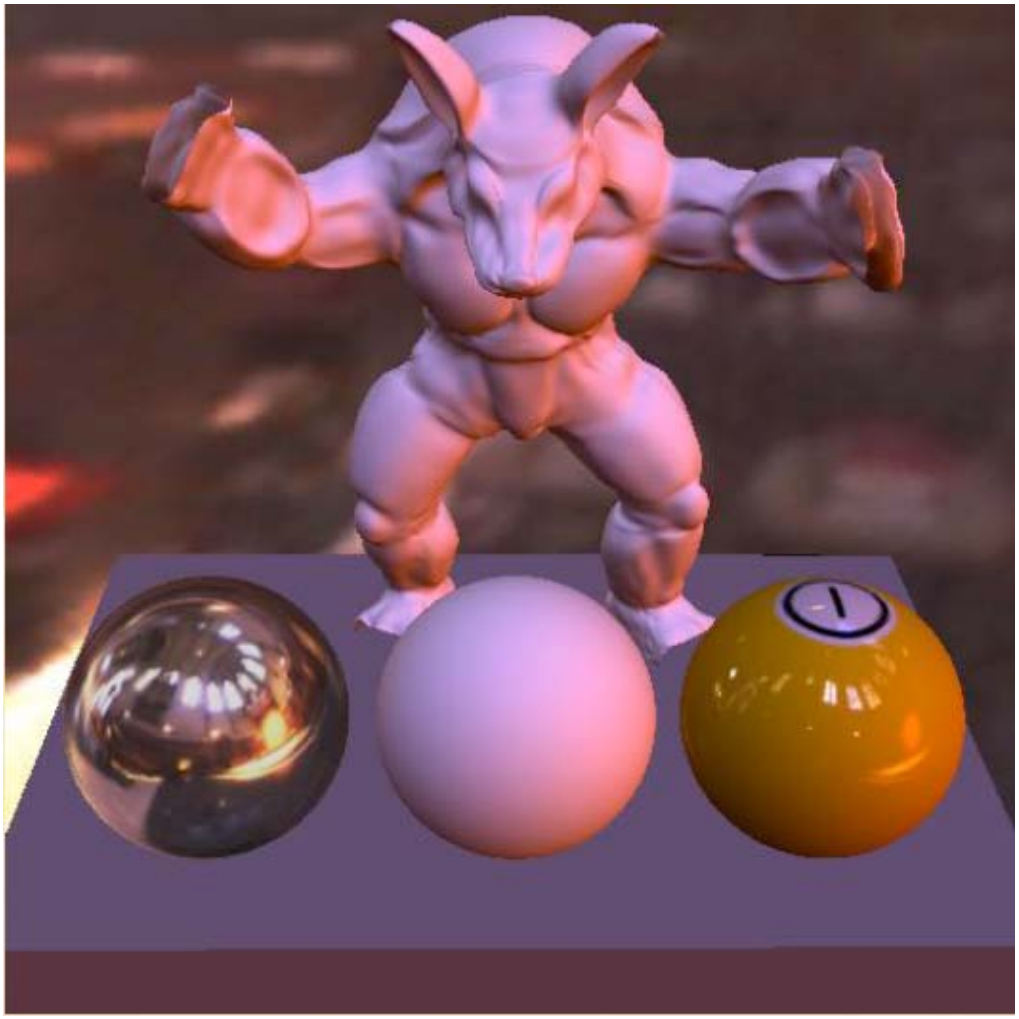


- You can reuse environment maps for diffuse reflections
- Integrate the map over a hemisphere at each pixel (basically blurs the whole thing out)



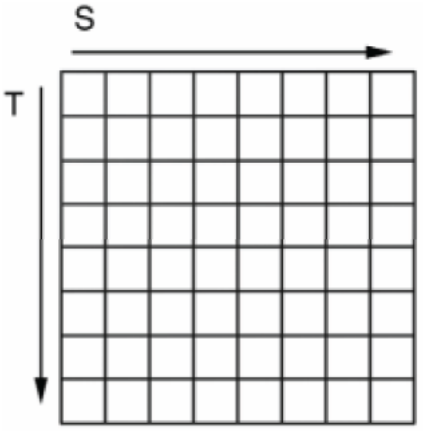


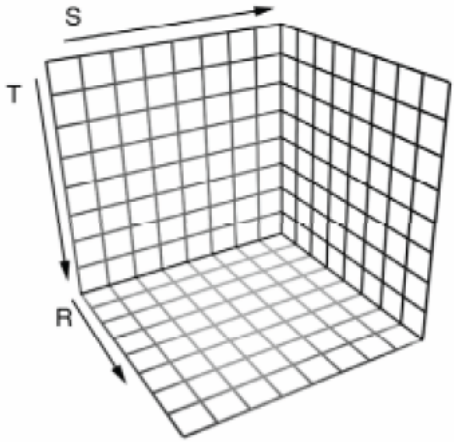
Irradiance Mapping Example

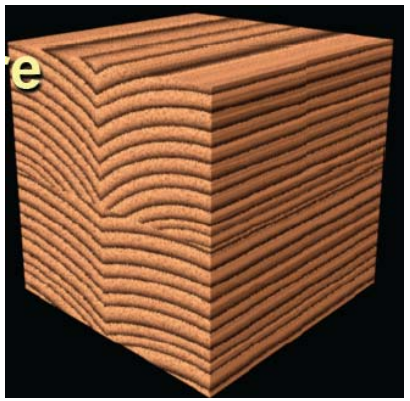


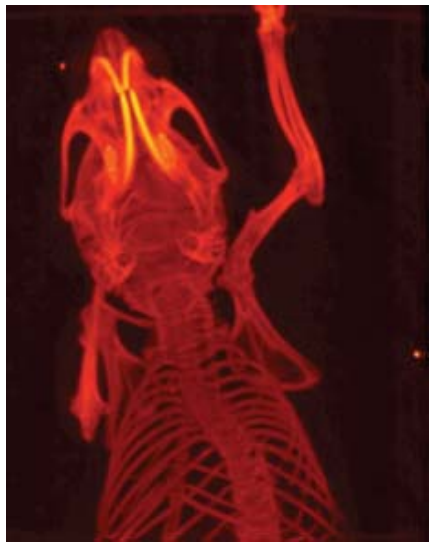
3D Textures

- 3D volumetric textures exist as well, though you can only render slices of them in OpenGL
- Generate a full image by stacking up slices in Z
- Used in visualization

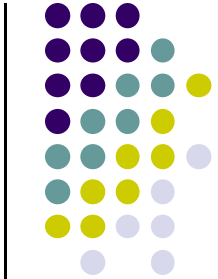




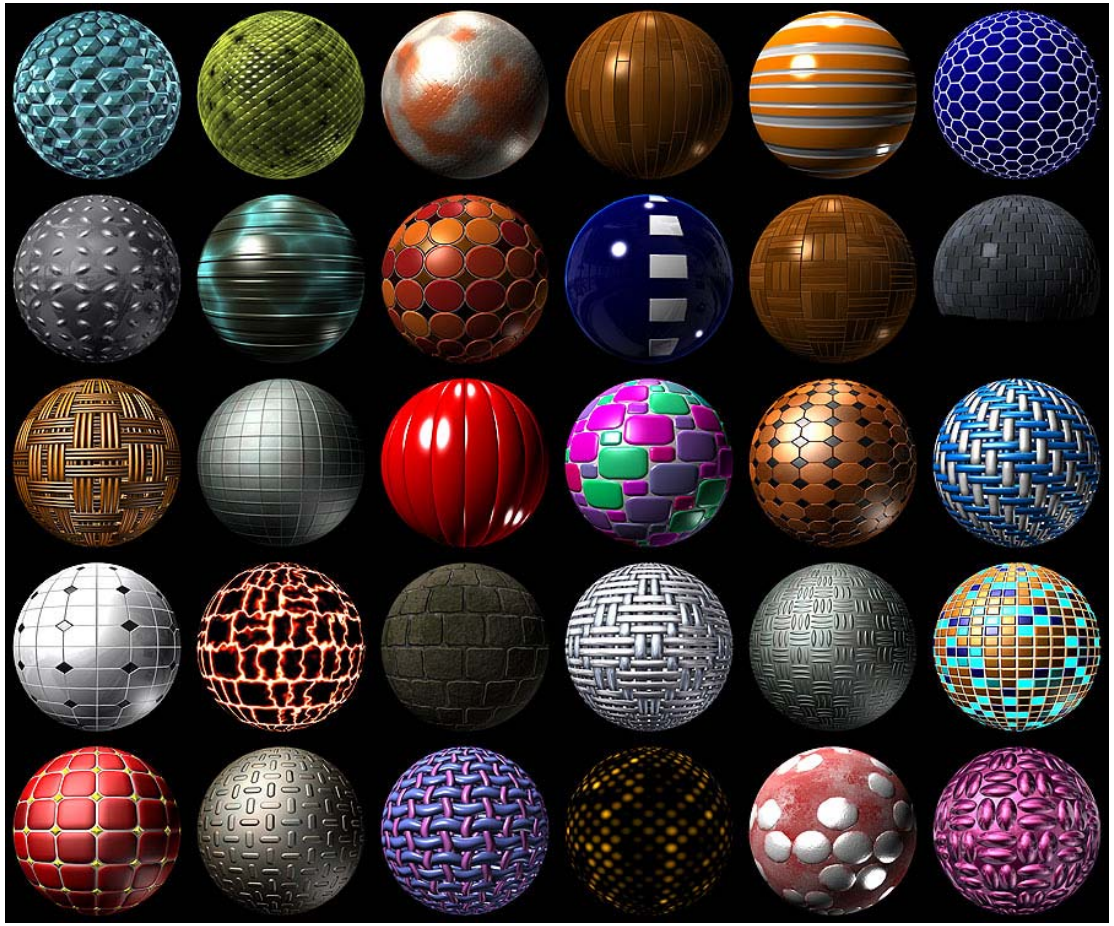




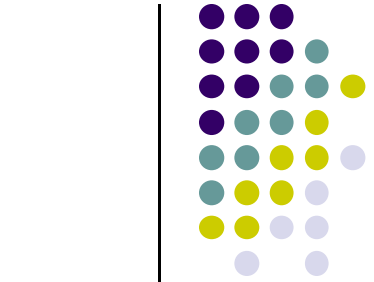
Procedural Texturing



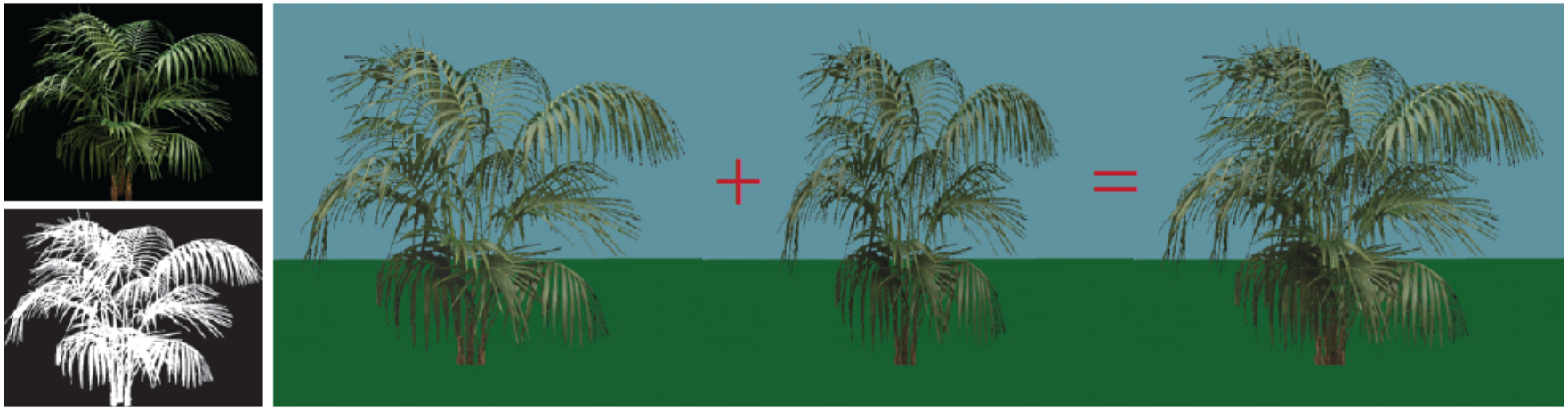
- Math functions that generate textures



Alpha Mapping



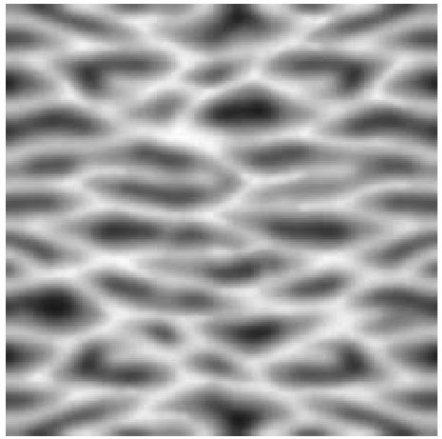
- Represent the alpha channel with a texture
- Can give complex outlines, used for plants

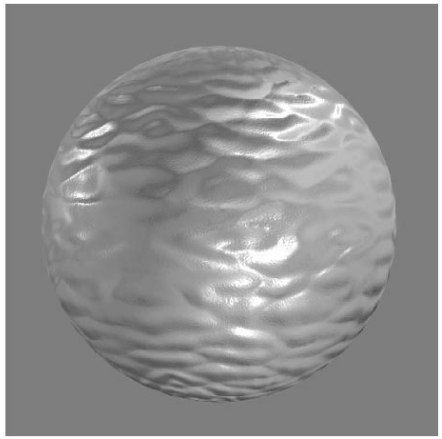


Render Bush
on 1 polygon

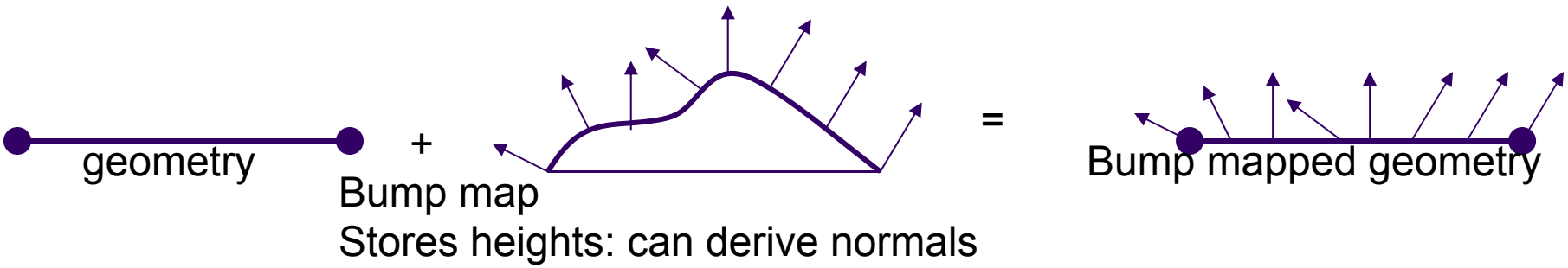
Render Bush
on polygon rotated
90 degrees

Bump mapping

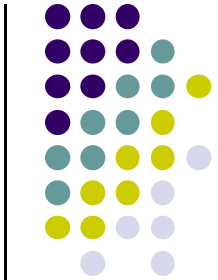




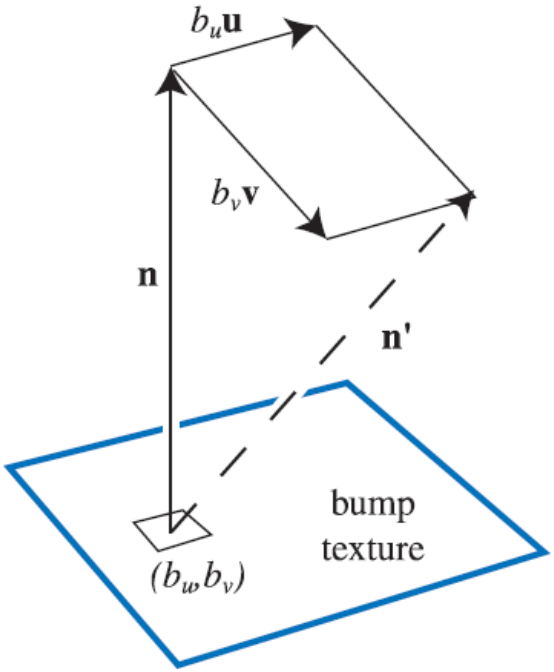
- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
  - Too expensive to model these geometrically
- Instead let a texture modify the normal at each pixel, and then use this normal to compute lighting

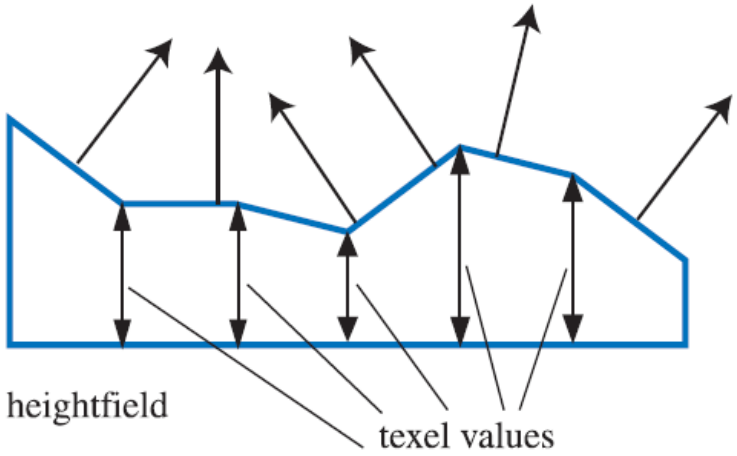


Bump mapping: Blinn's method

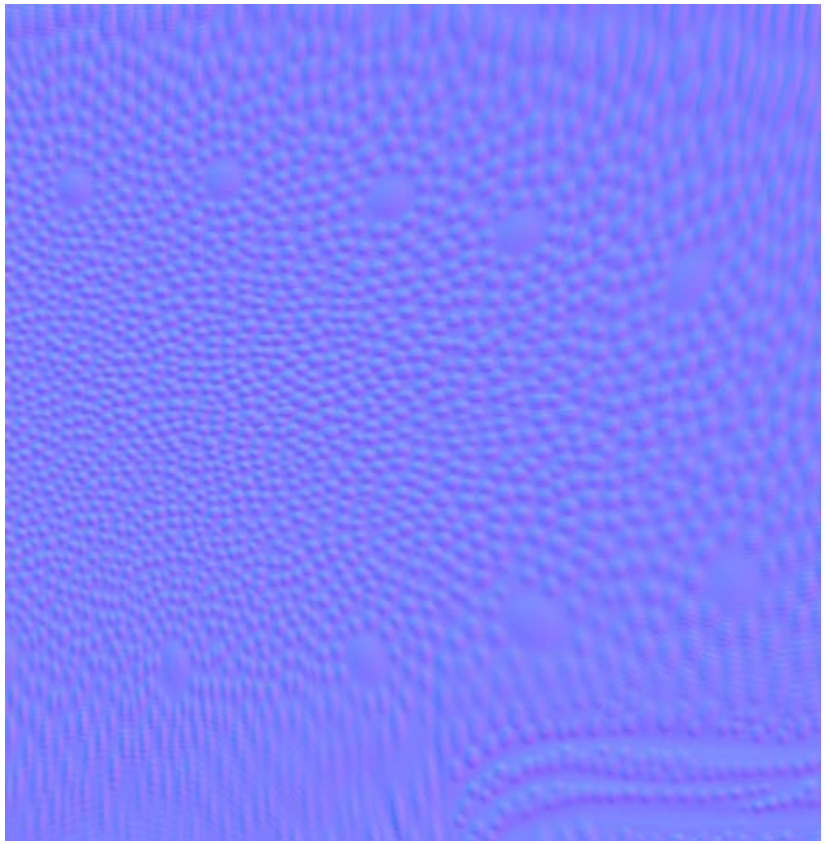


- Basic idea:
  - Distort the surface along the normal at that point
  - Magnitude is equal to value in heighfield at that location



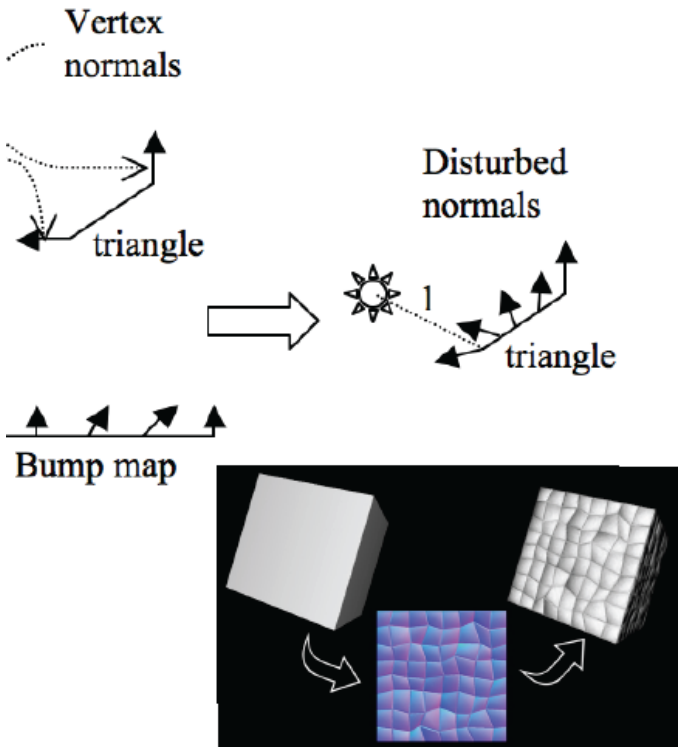


Bump mapping: examples

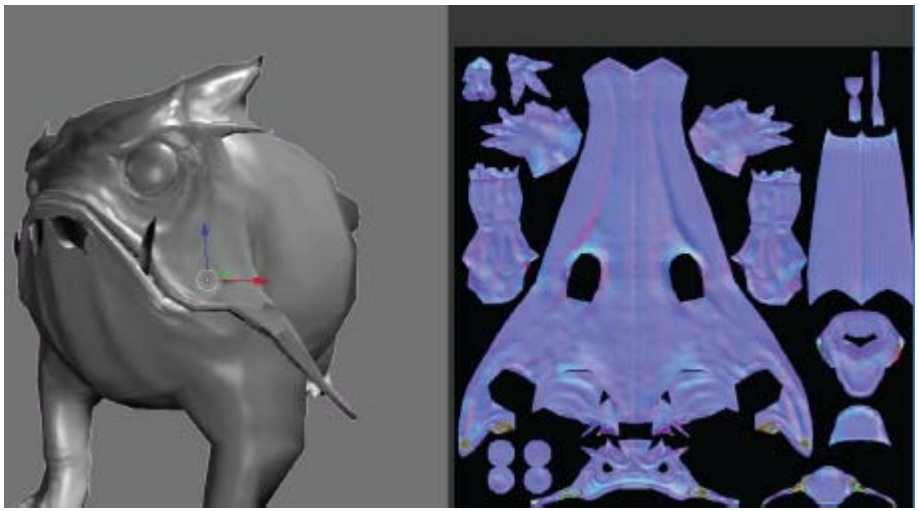


Bump Mapping Vs Normal Mapping

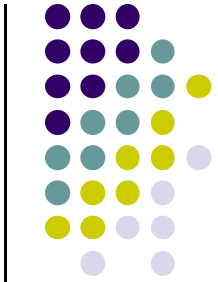
- **Bump mapping**
- (Normals $\mathbf{n}=(n_x, n_y, n_z)$ stored as *distortion of face orientation*. Same bump map can be tiled/repeated and reused for many faces)



- **Normal mapping**
- Coordinates of normal (relative to tangent space) are encoded in color channels
- Normals stored include face orientation + plus distortion.)

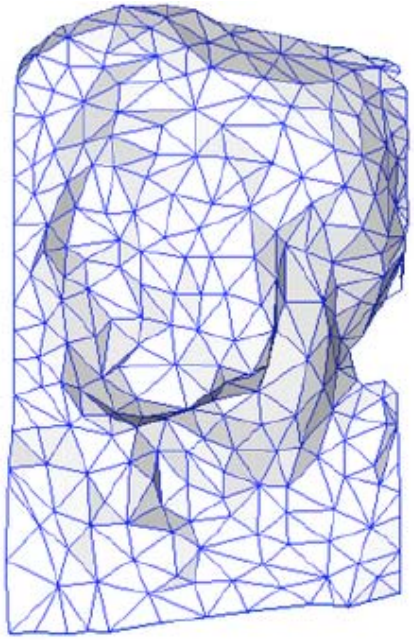


Normal Mapping



- Very useful for making low-resolution geometry look like it's much more detailed

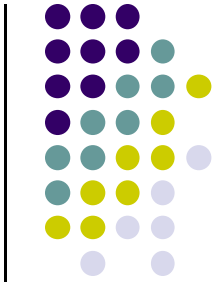
original mesh
4M triangles



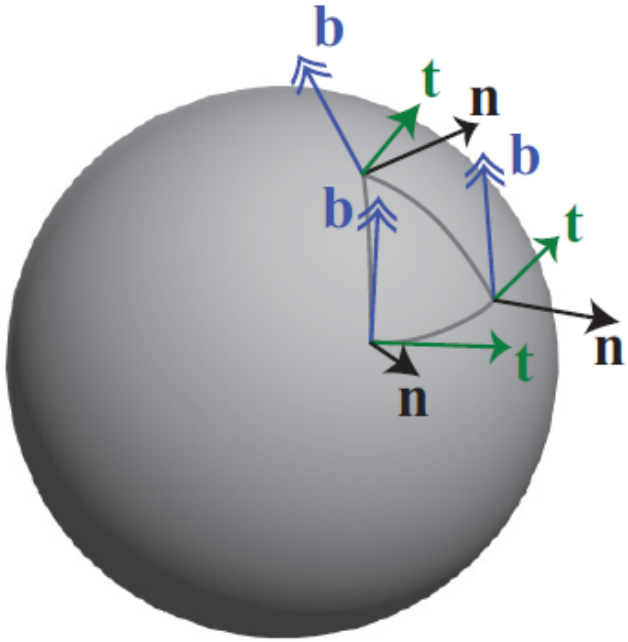
simplified mesh
500 triangles

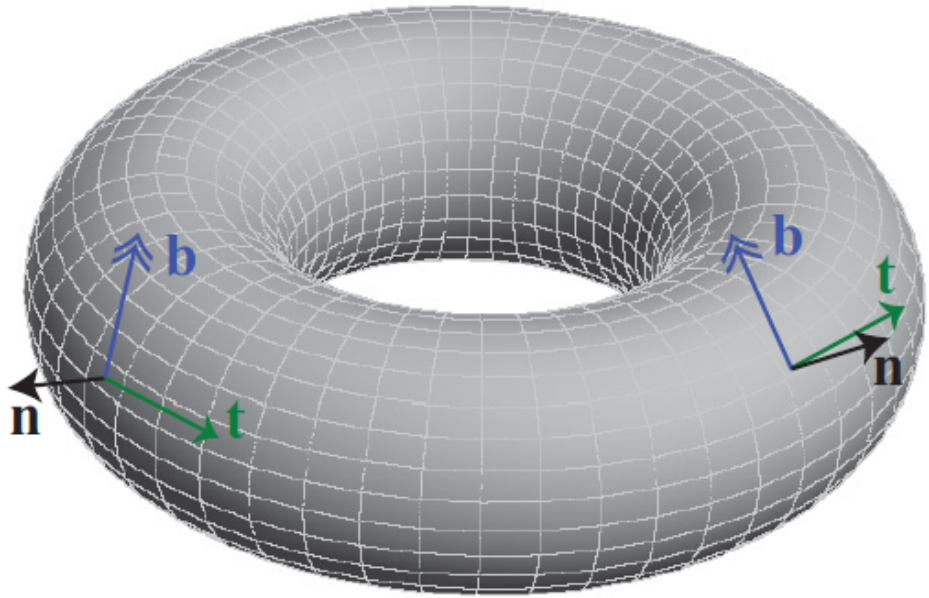
simplified mesh
and normal mapping
500 triangles

Tangent Space Vectors



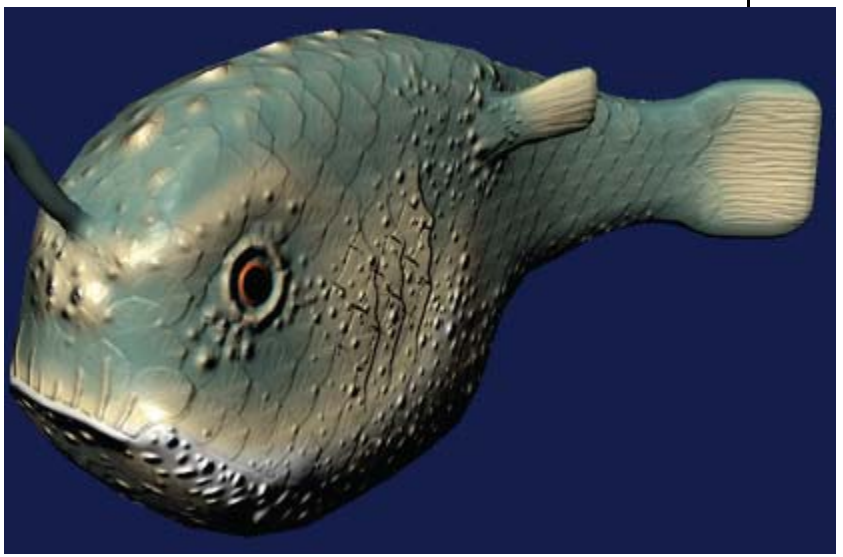
- Normals stored in local coordinate frame
- Need Tangent, normal and bi-tangent vectors

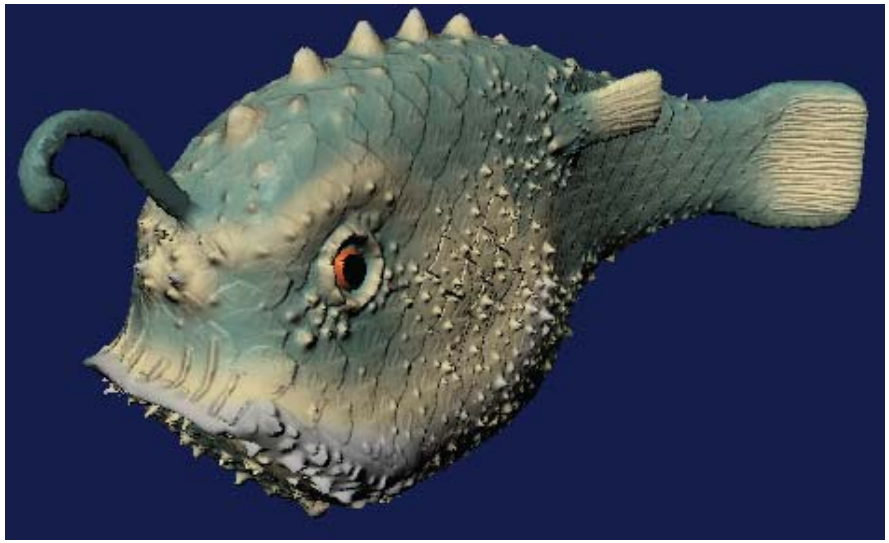




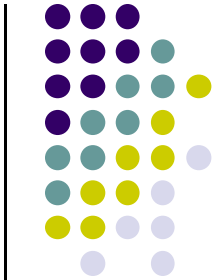
Displacement Mapping

- Uses a map to displace the surface at each position
- Offsets the position per pixel or per vertex
  - Offsetting per vertex is easy in vertex shader
  - Offsetting per pixel is architecturally hard

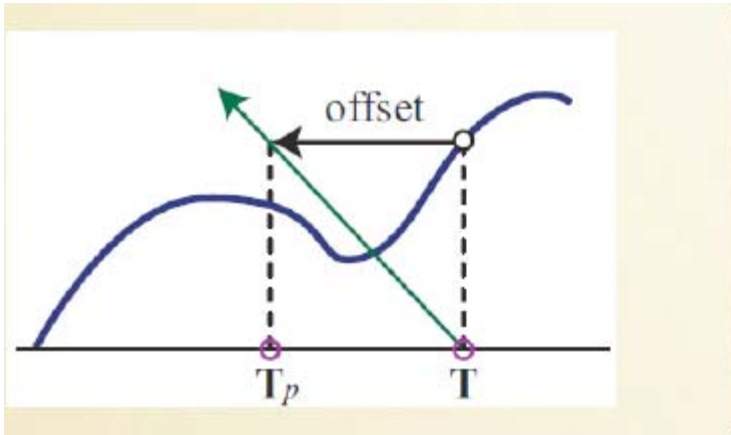




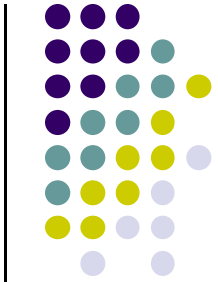
Parallax Mapping



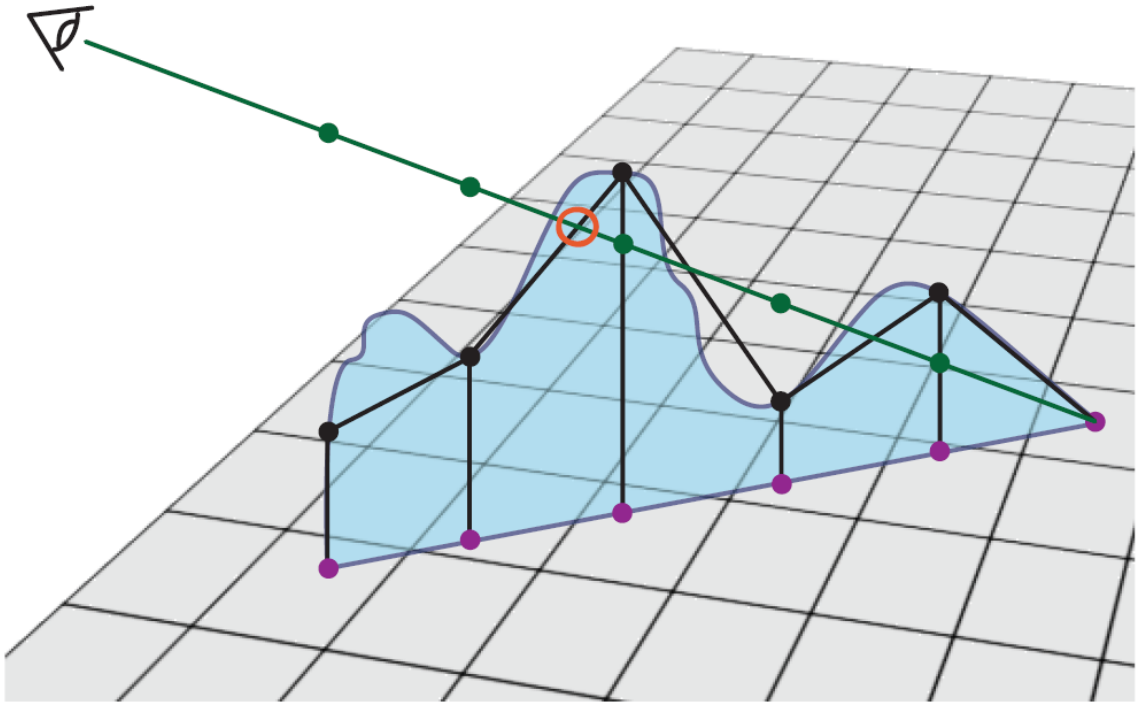
- Normal maps increase lighting detail, but they lack a sense of depth when you get up close
- Parallax mapping
  - simulates depth/blockage of one part by another
  - Uses heightmap to offset texture value / normal lookup
  - Different texture returned after offset



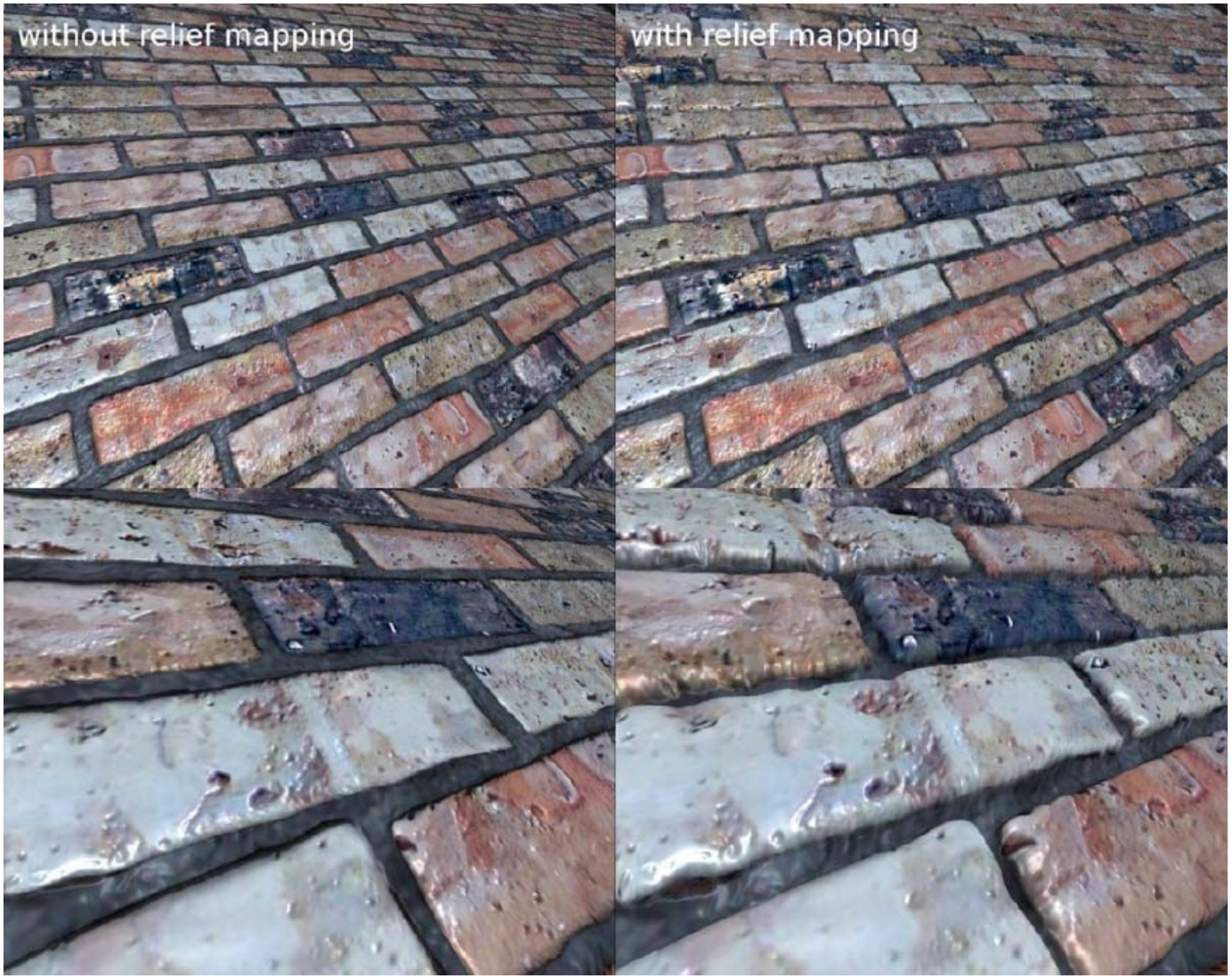
Relief Mapping

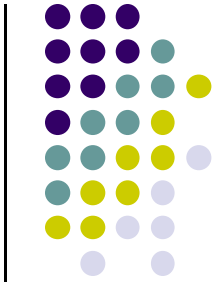


- Implement a heightfield raytracer in a shader
- Pretty expensive, but looks amazing



Relief Mapping Example





References

- UIUC CS 319, Advanced Computer Graphics Course
- David Luebke, CS 446, U. of Virginia, slides
- Chapter 1-6 of RT Rendering
- CS 543/4731 course slides
- Hanspeter Pfister, CS 175 Introduction to Computer Graphics, Harvard Extension School, Fall 2010 slides
- Christian Miller, CS 354, Computer Graphics, U. of Texas, Austin slides, Fall 2011
- Ulf Assarsson, TDA361/DIT220 - Computer graphics 2011, Chalmers Instititute of Tech, Sweden