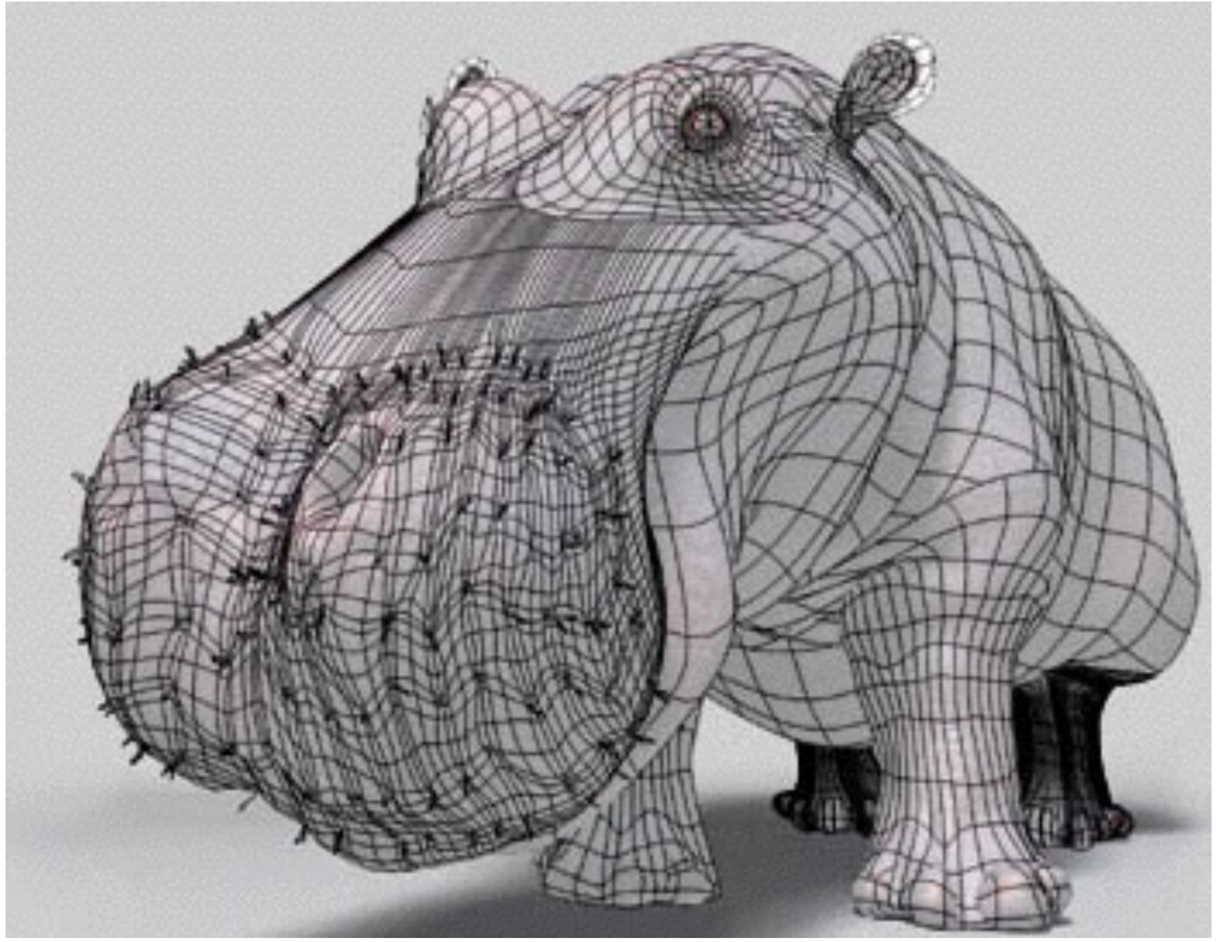
# Texture Mapping

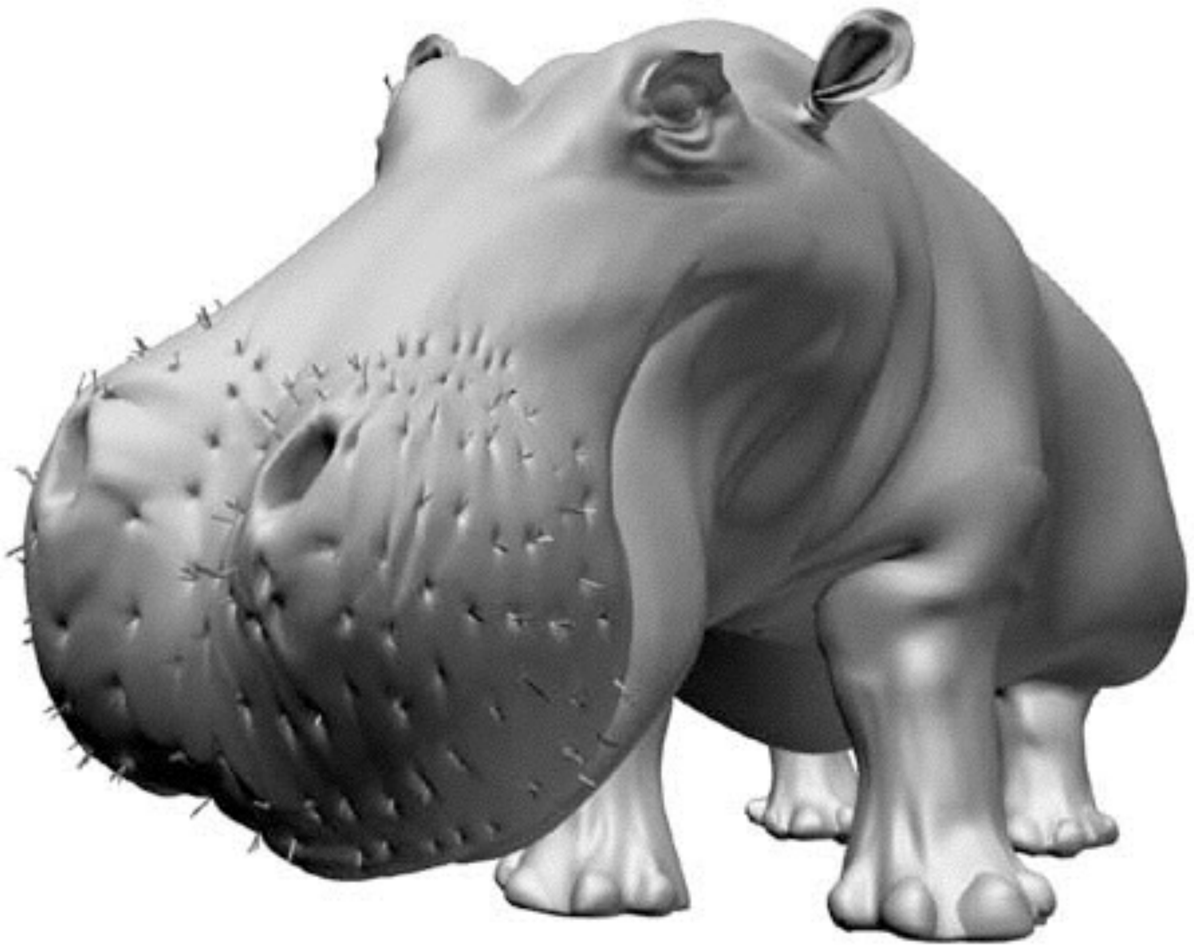
Connelly Barnes

CS 4810: Graphics

Acknowledgment: slides by Jason Lawrence, Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin

# Textures

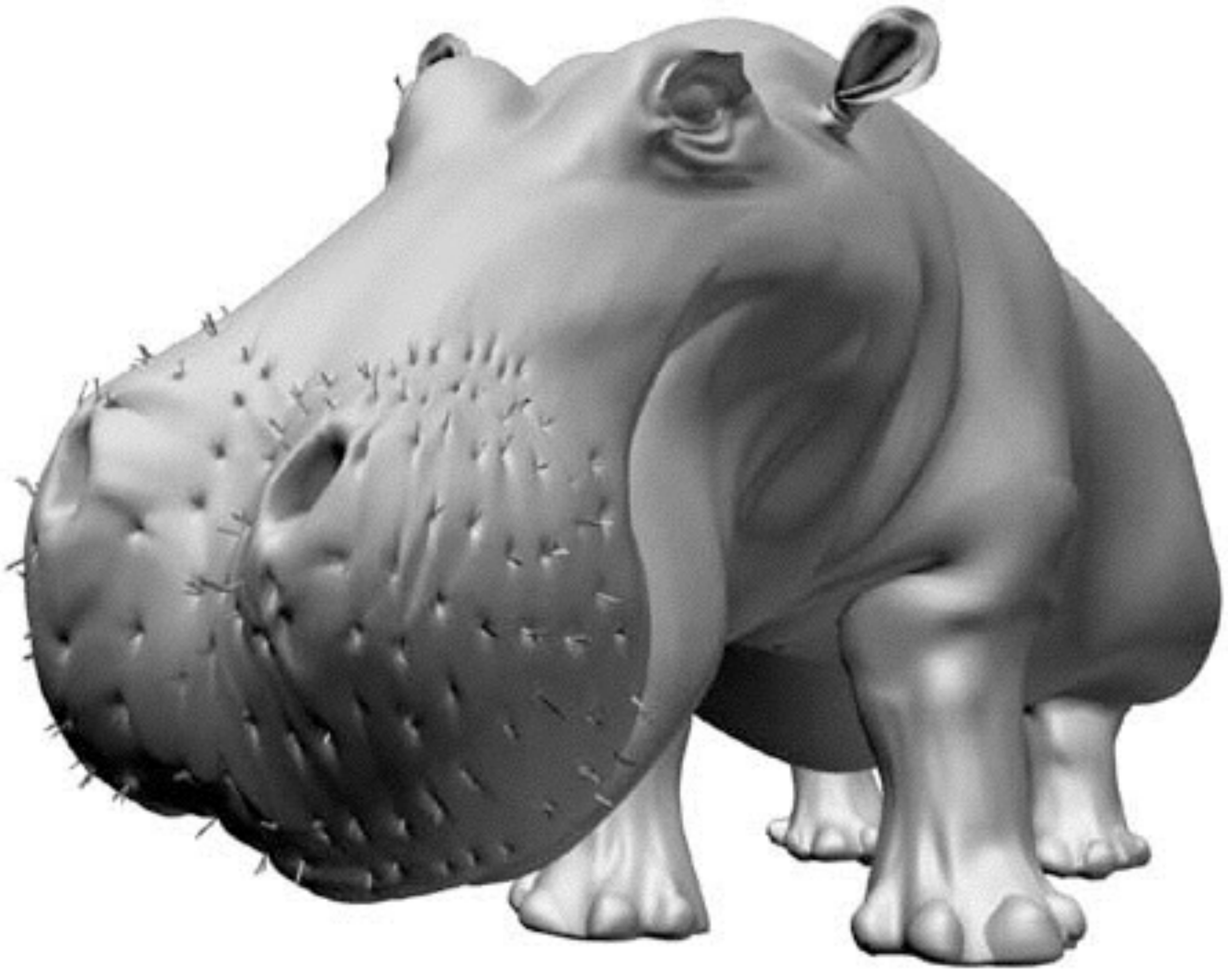




We know how to go from this… to this

J. Birn

# Textures



But what about this…

to this?

J. Birn

# Textures

- How can we go about drawing surfaces with complex detail?

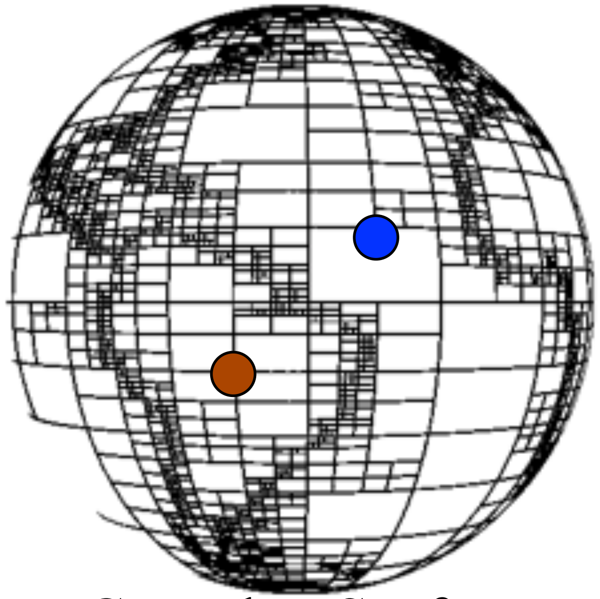
Target Model

# Textures

- How can we go about drawing surfaces with complex detail?

Target Model

- We could tessellate the sphere in a complex fashion and then associate the appropriate material properties to each vertex



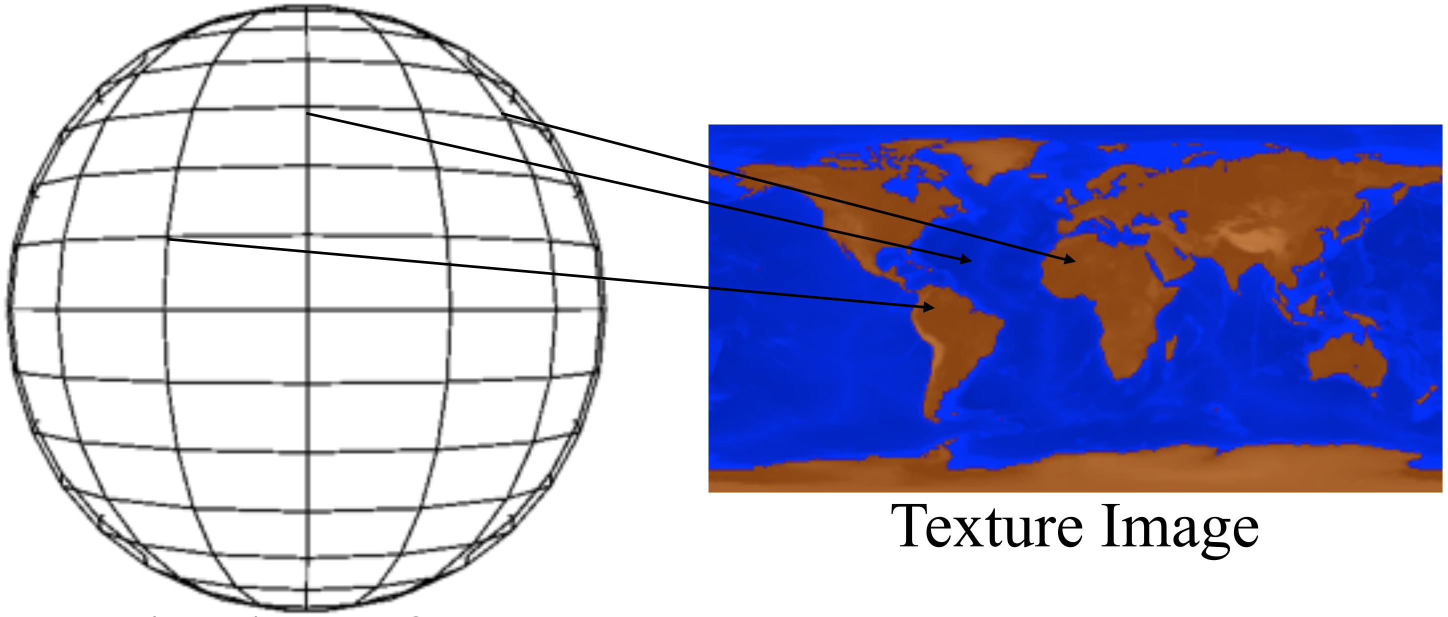
Complex Surface

# Textures

- How can we go about drawing surfaces with complex detail?

Target Model

- We could use a simple tessellation and use the location of surface points to look up the appropriate color values

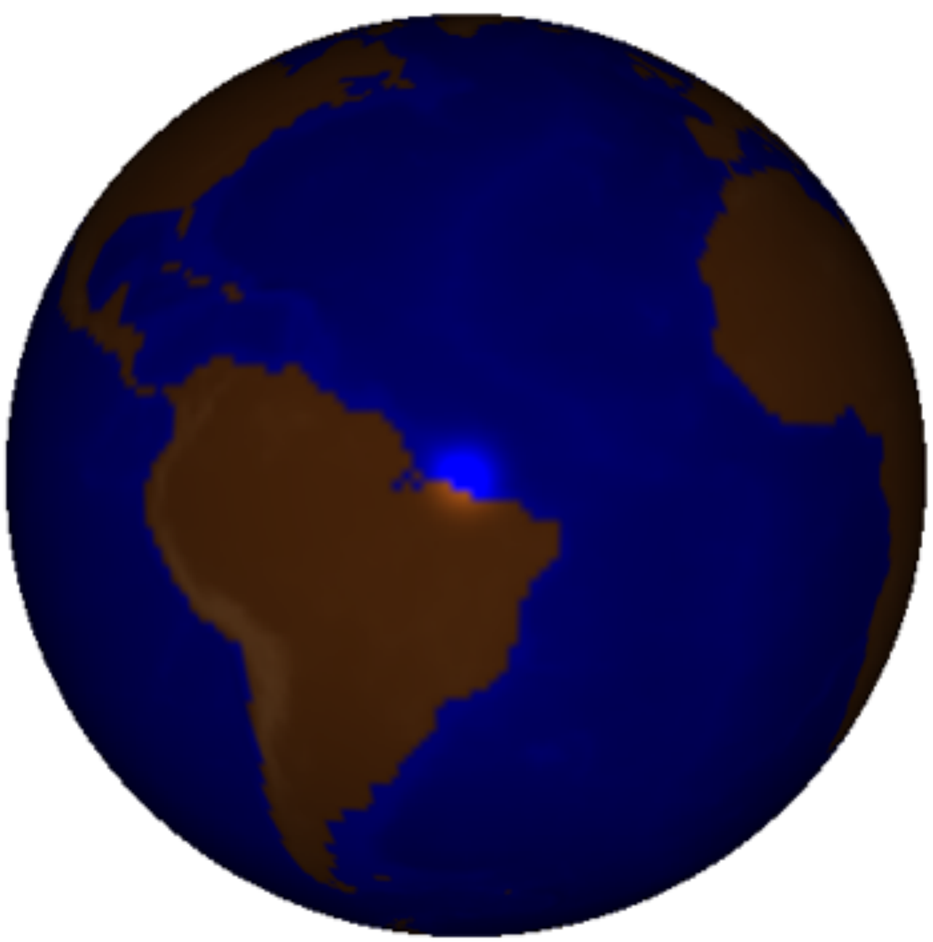


Simple Surface

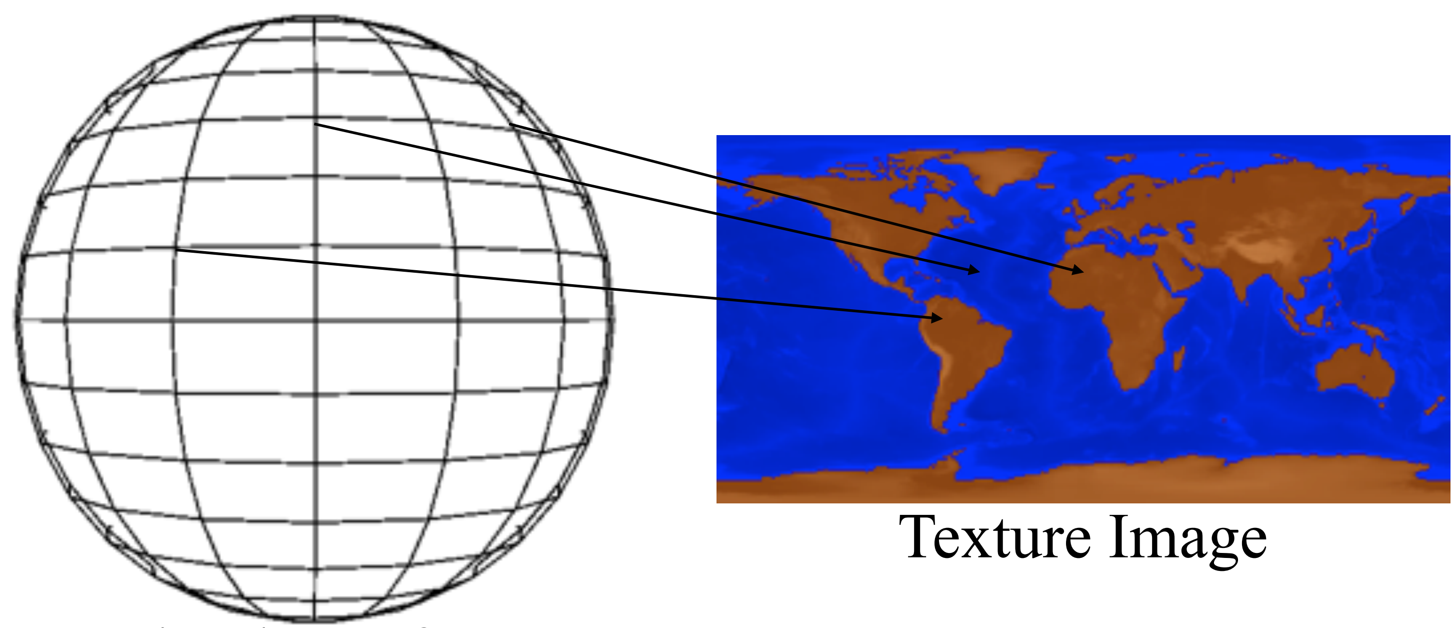
Texture Image

# Textures

- Advantages:
  - The 3D model remains simple
  - It is easier to design/modify a texture image than it is to design/modify a surface in 3D.



Target Model



Texture Image

Simple Surface

# Textures

Properties:

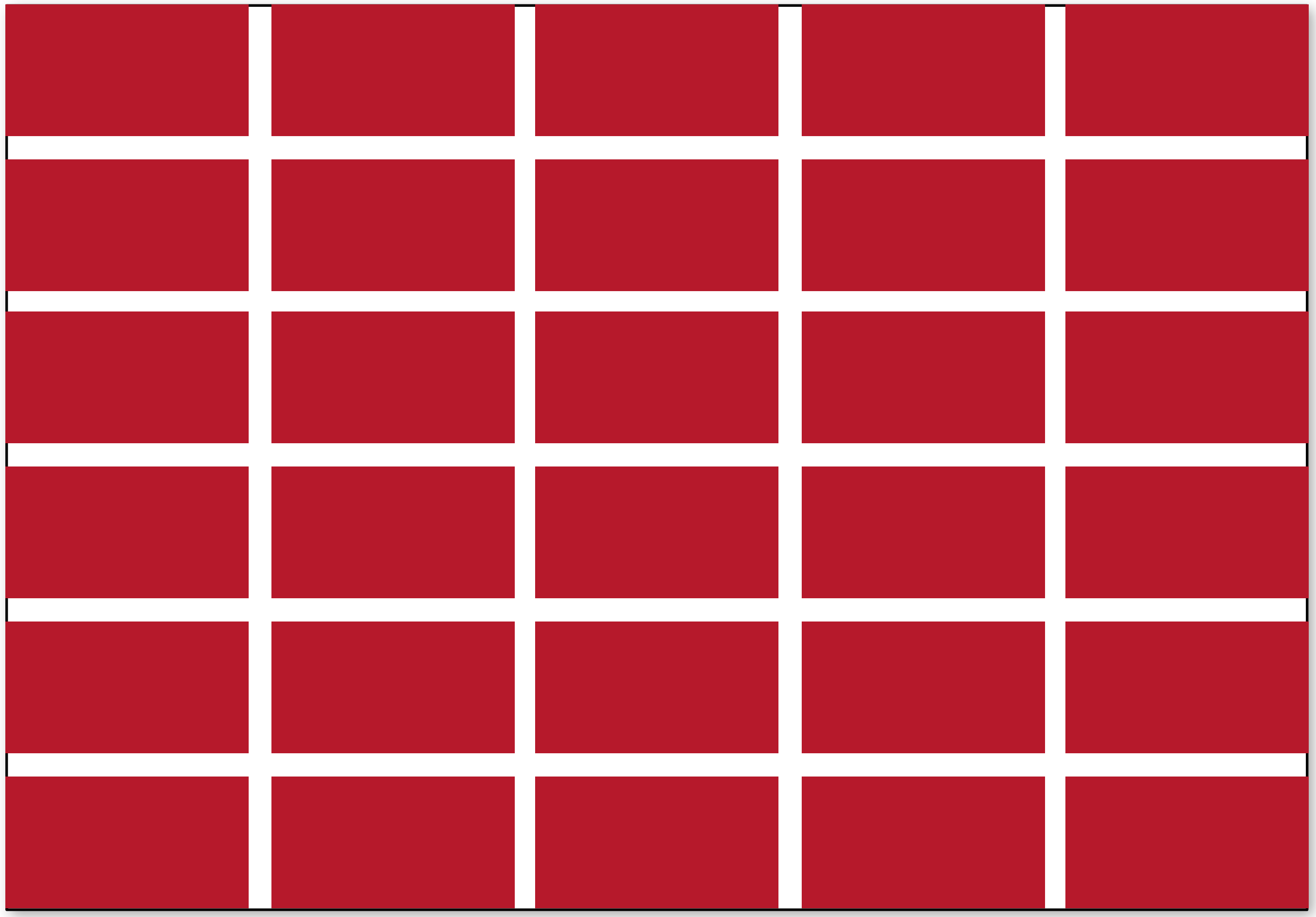
- Alter shading of individual pixels
- Implemented as part of shading process
- Rely on maps being stored as 1D, 2D, or 3D images
- Subject to aliasing errors

# Textures

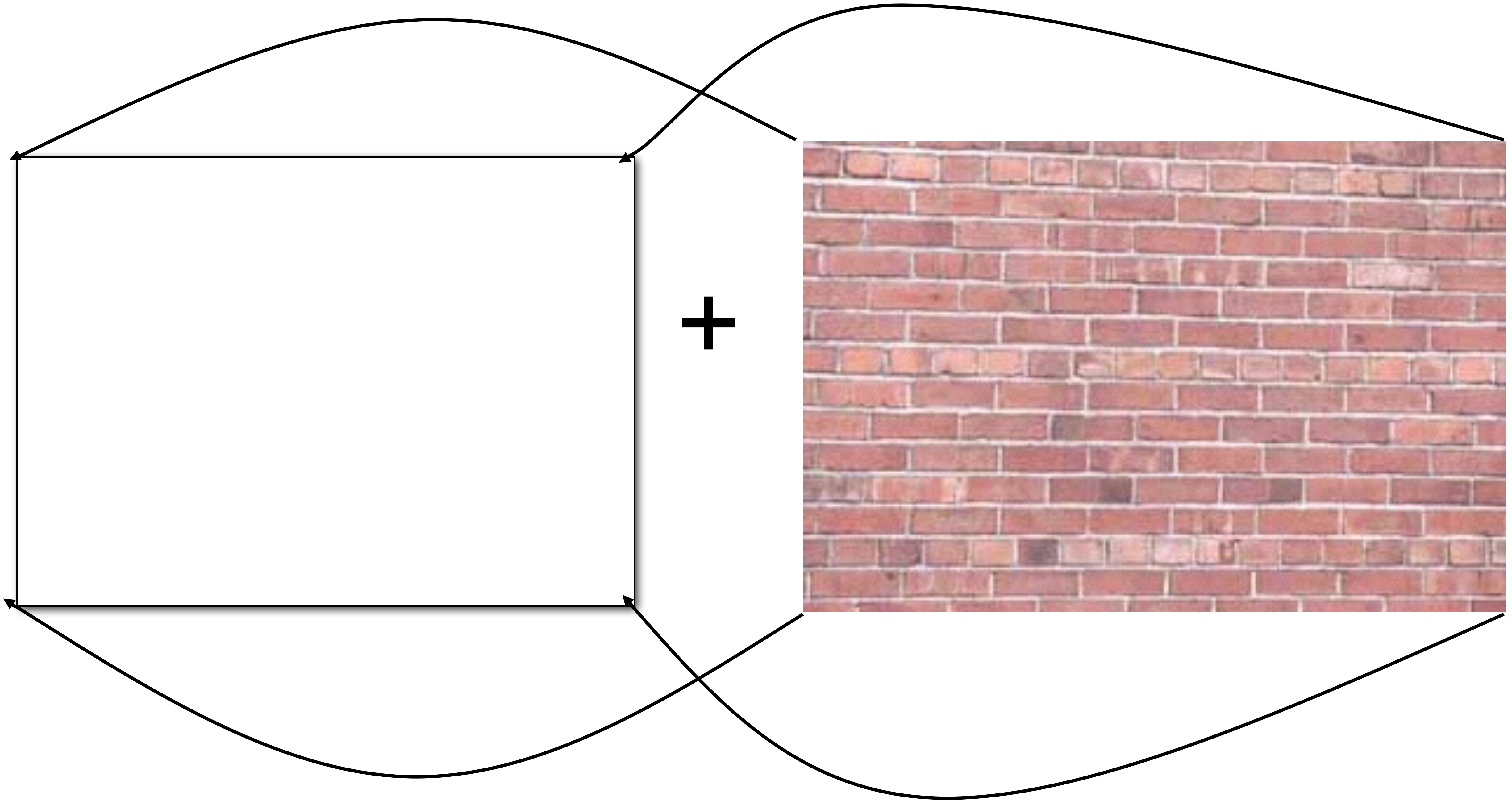
General Implementation Approach:

- Associate a collection of coordinates $(s_1,\ldots,s_n)$ to every vertex $(0 \leq s_i \leq 1)$
- Use the color of the image at position $(s_1,\ldots,s_n)$ to define the color of a vertex

# Another Example: Brick Wall

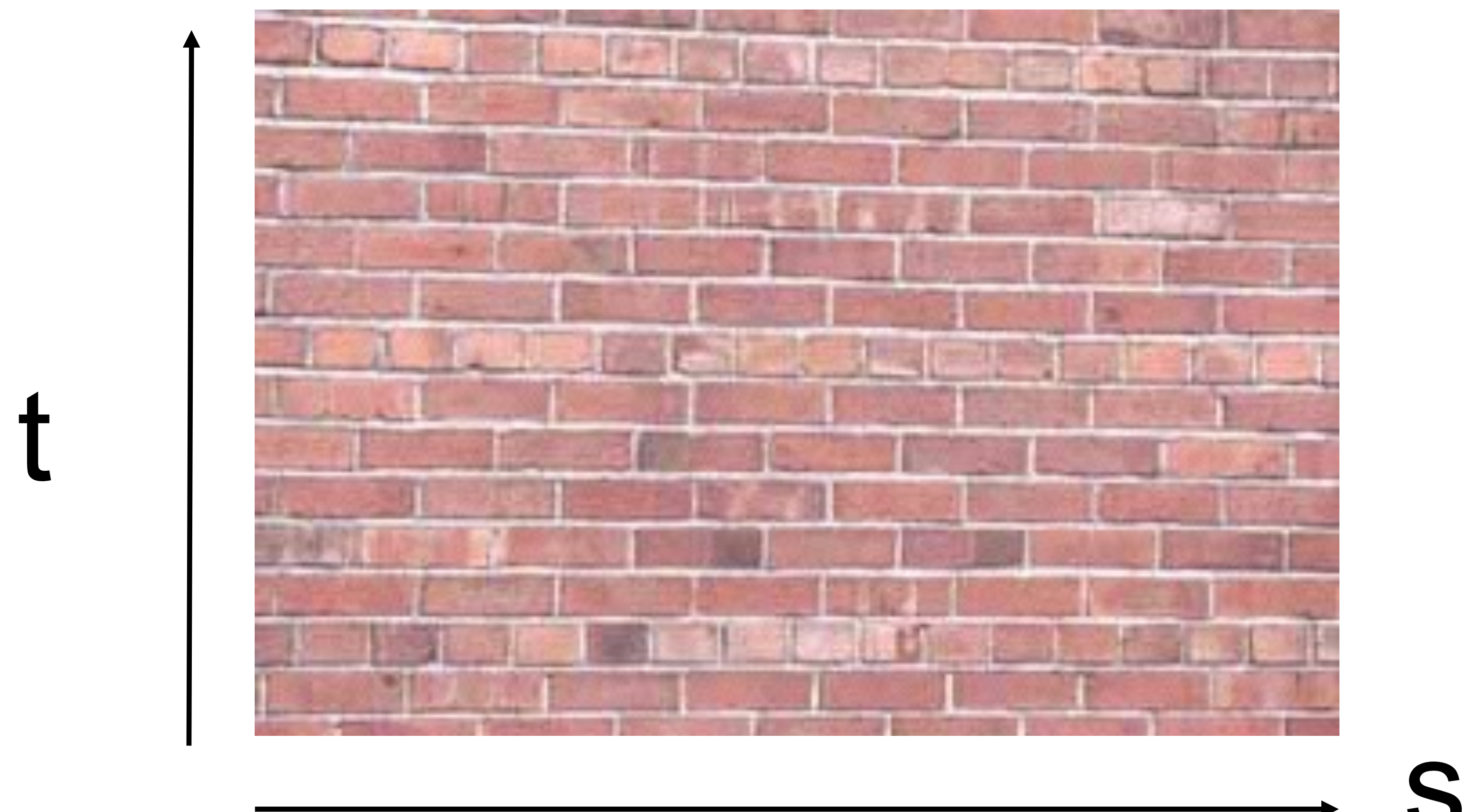


# Another Example: Brick Wall



# 2D Texture

- Coordinates described by variables *s* and *t and* range over interval (0,1)
- Texture elements are called *texels*
- Often 4 bytes (rgba) per texel

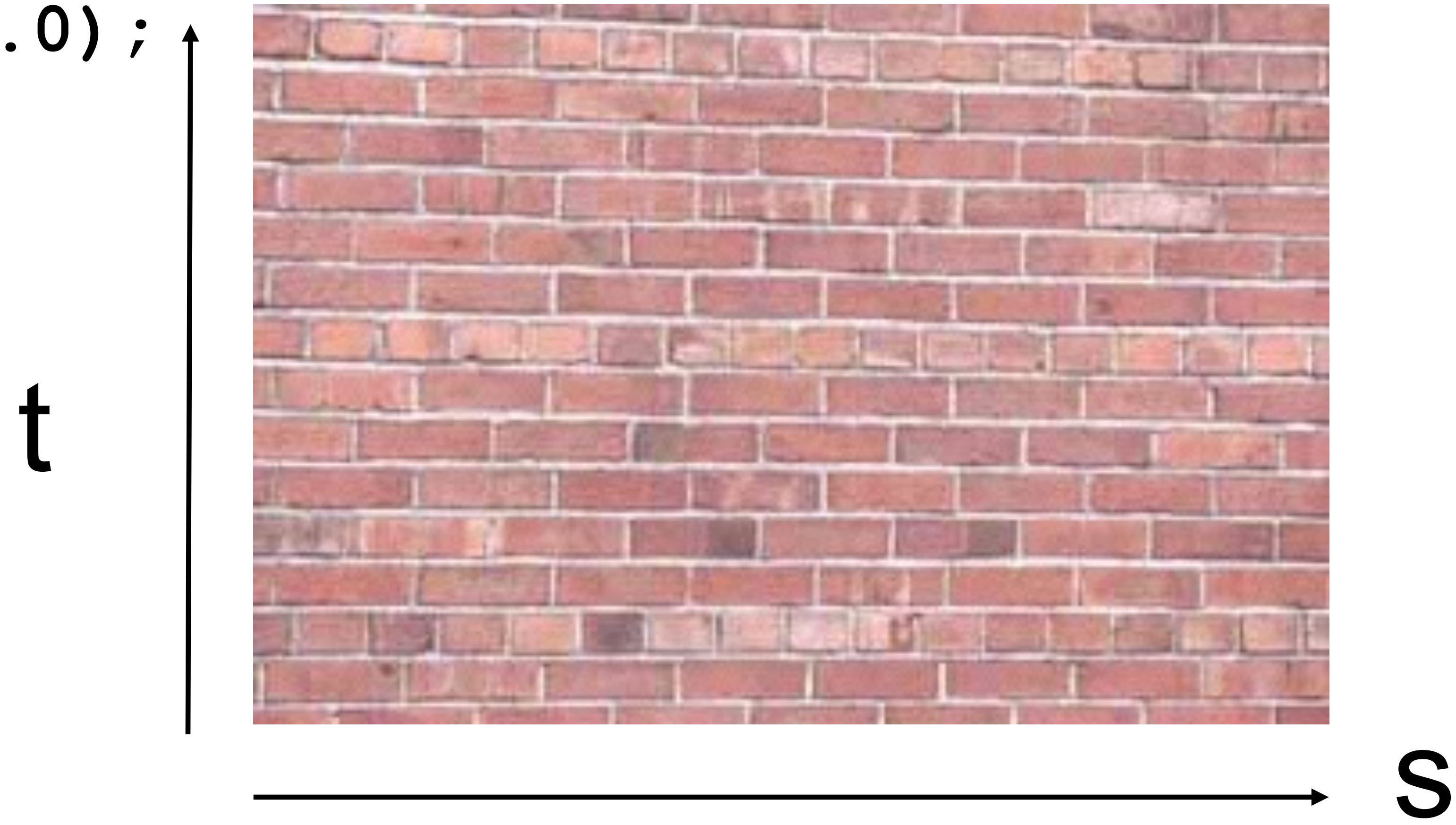


# 2D Texture

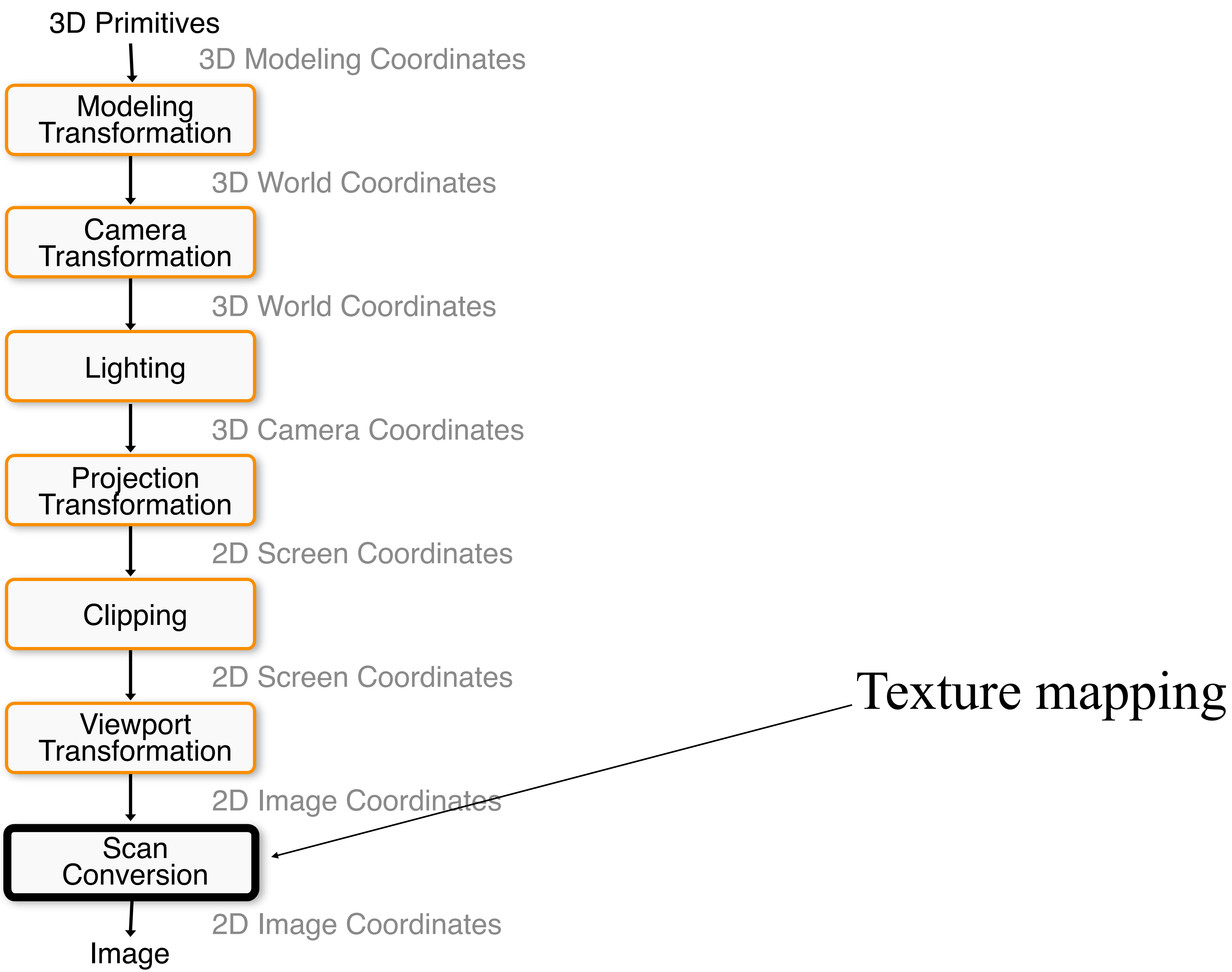
```
glBegin(GL_TRIANGLE);
glTexCoord2f(0.0, 0.0);
glVertex3f(0.0, 0.0, 0.0);

glTexCoord2f(1.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);

glTexCoord2f(1.0, 1.0);
glVertex3f(1.0, 1.0, 0.0);
glEnd();
```



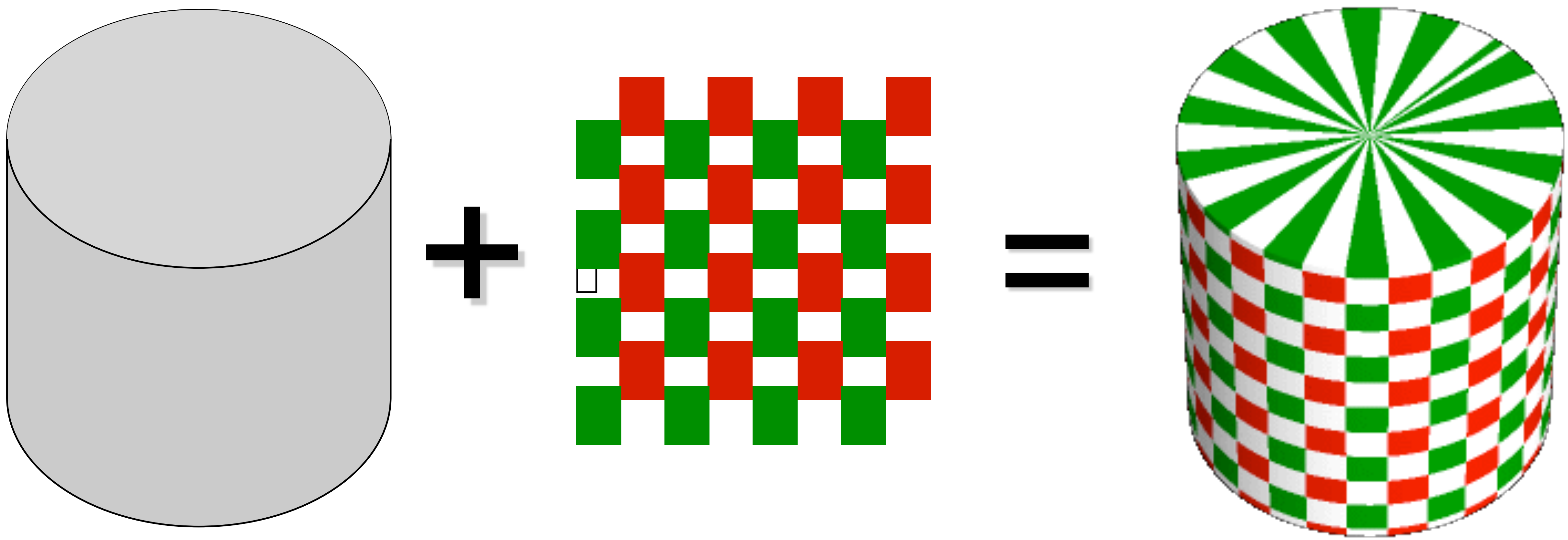
# 3D Rendering Pipeline (for direct illumination)



# Overview

- Texture mapping methods
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Shadow maps
  - Volume Textures

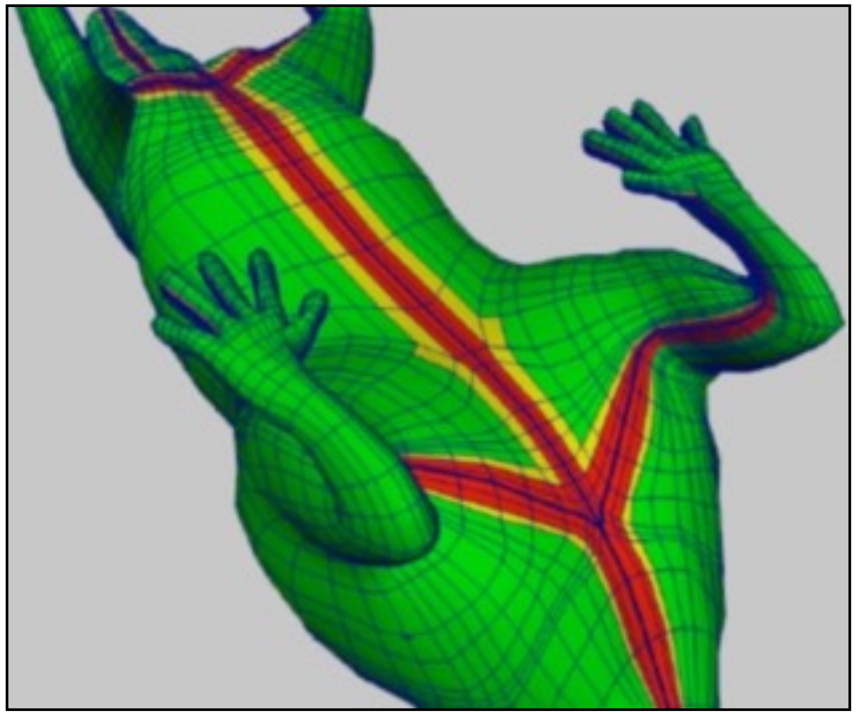
# Parameterization

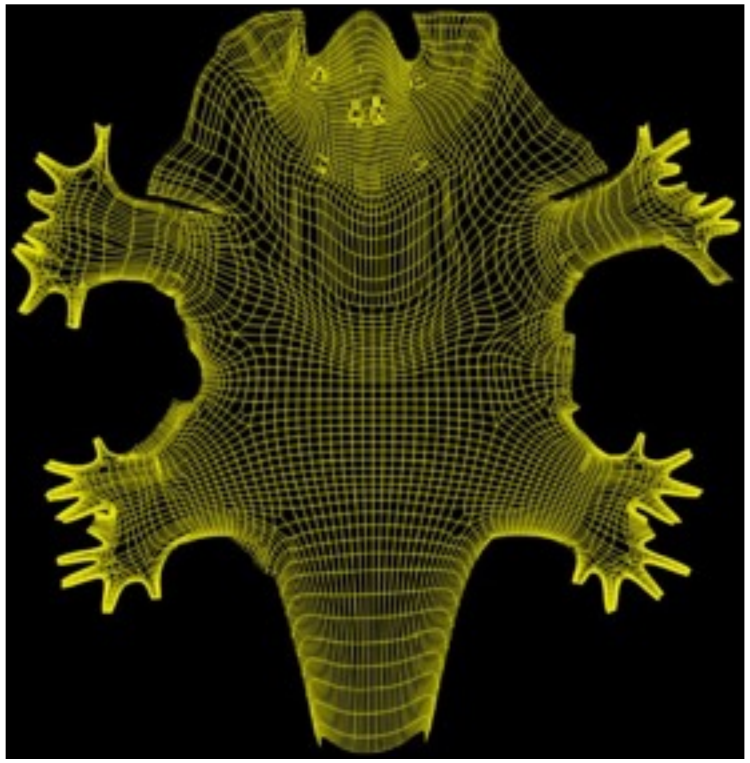



geometry — image — texture map

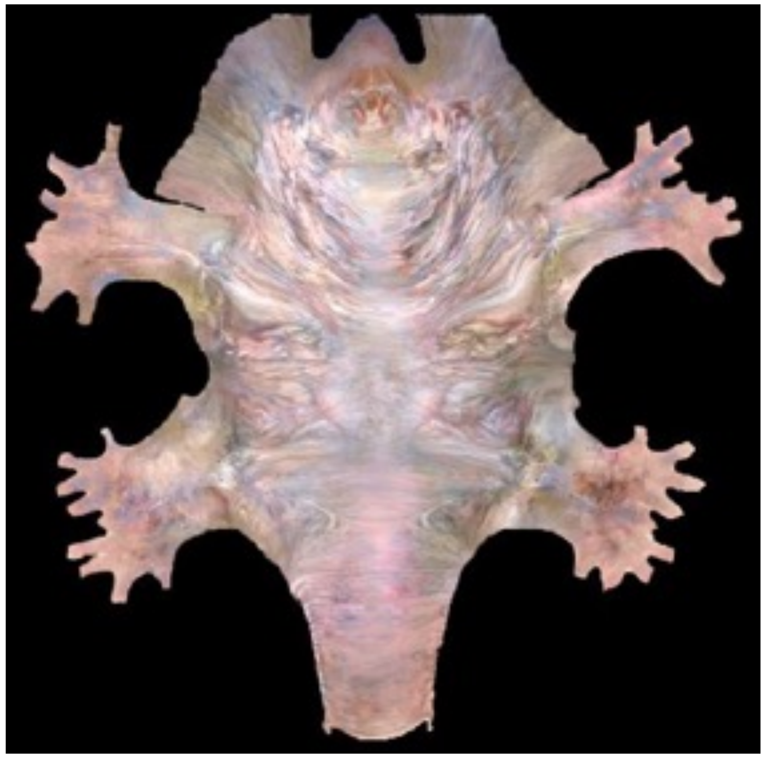
- Q: How do we decide *where* on the geometry each color from the image should go?

# Option: Unfold/Map Entire Surface



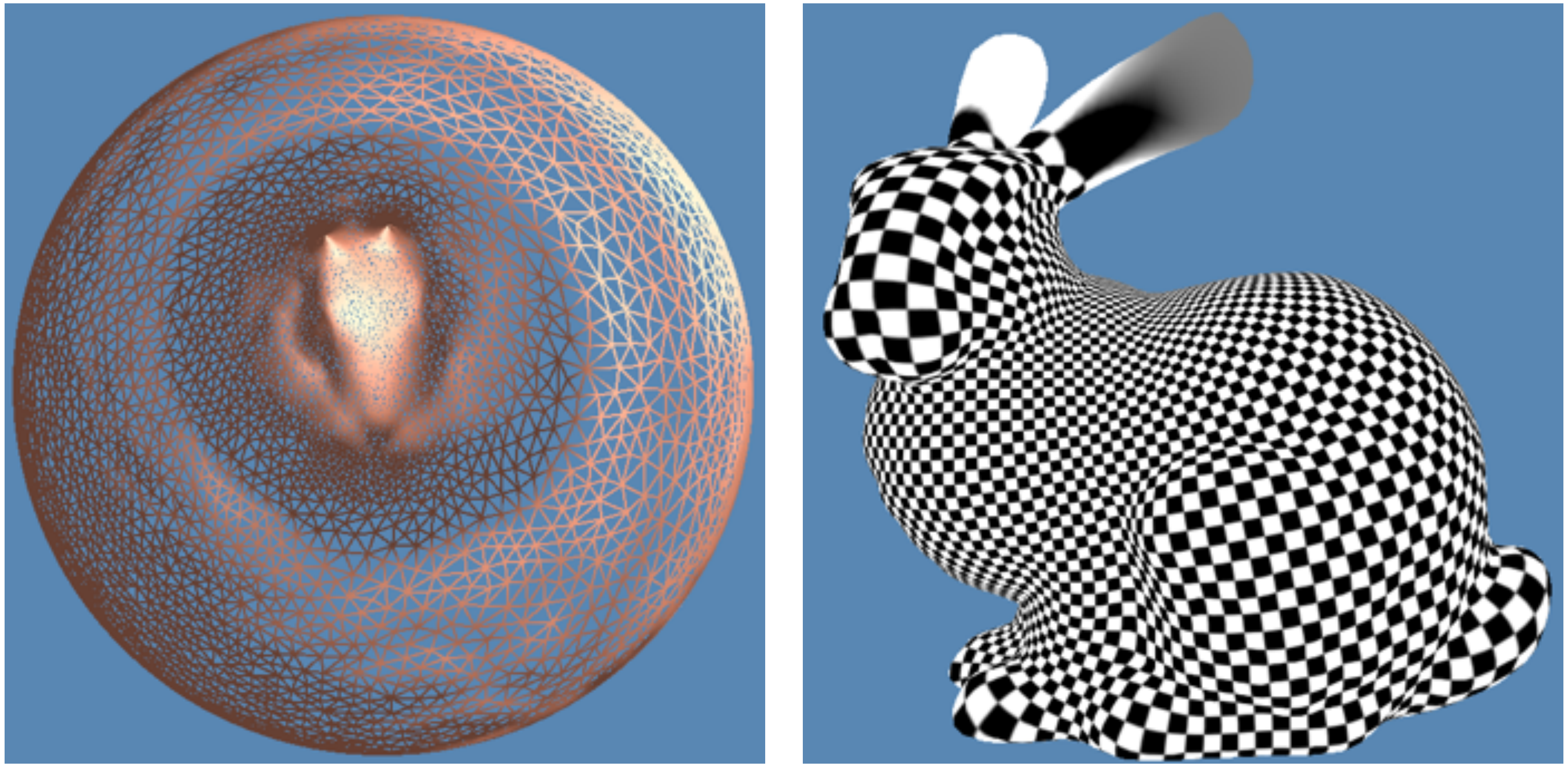


[Piponi2000]



# Option: Unfold/Map Entire Surface

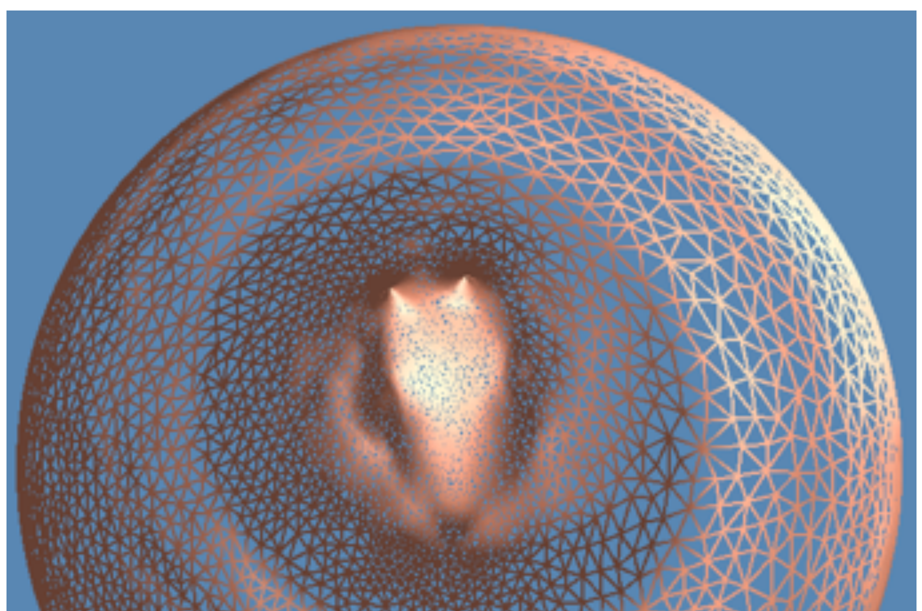
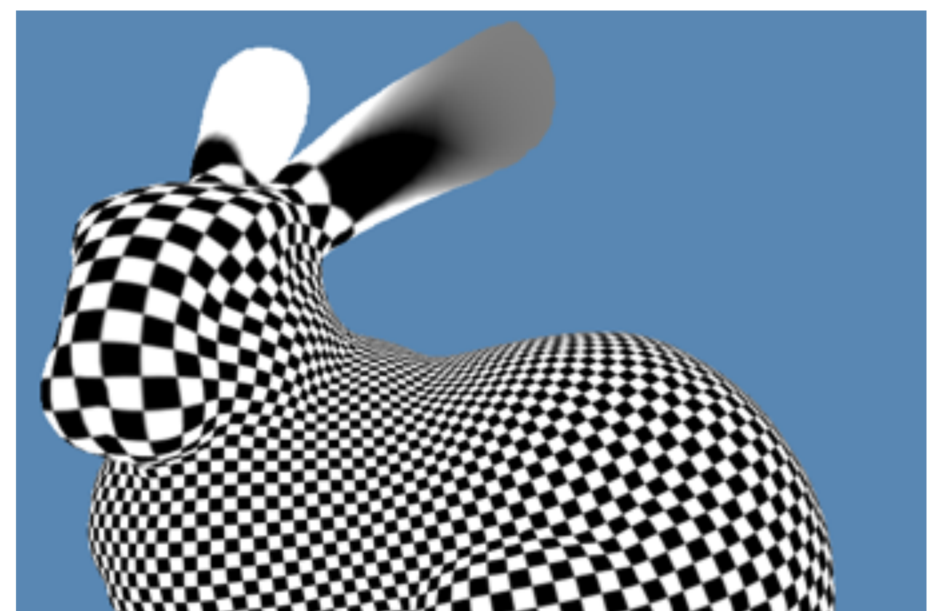
- Tricky, because mapped surface may have severe distortions
- However, because texture is continuous, may be easier to think about



Gu *et al*. 2003

# Option: Unfold/Map Entire Surface

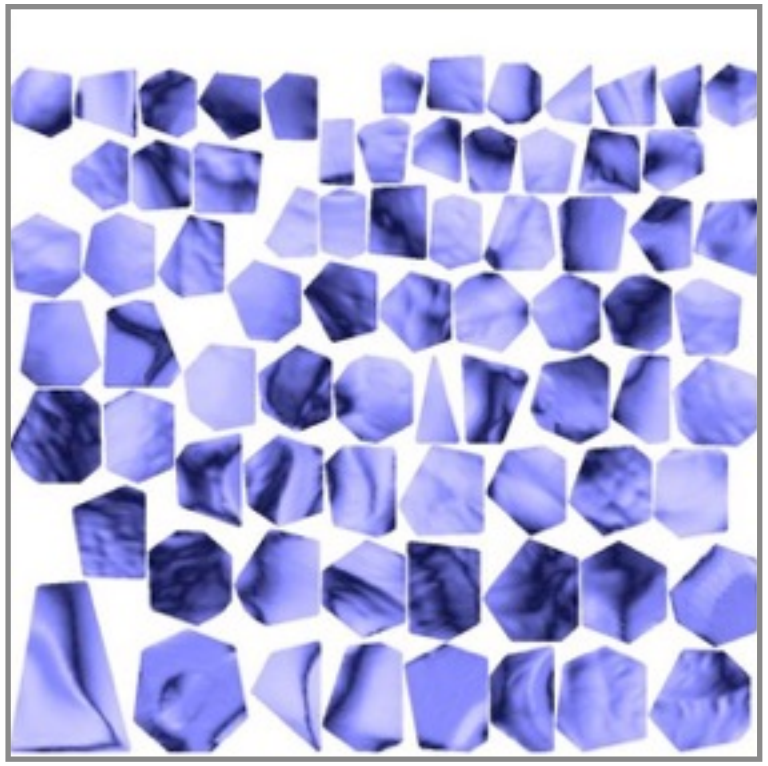
- Tricky, because mapped surface may have severe distortions
- However, because texture is continuous, may be easier to think about

In general, it is impossible to parameterize a complex shape to a simple base domain so that both angles and areas are preserved

# Option: Atlas

charts



atlas

surface

Can be produced automatically by software such as MeshLab

[Sander2001]

# Option: Atlas

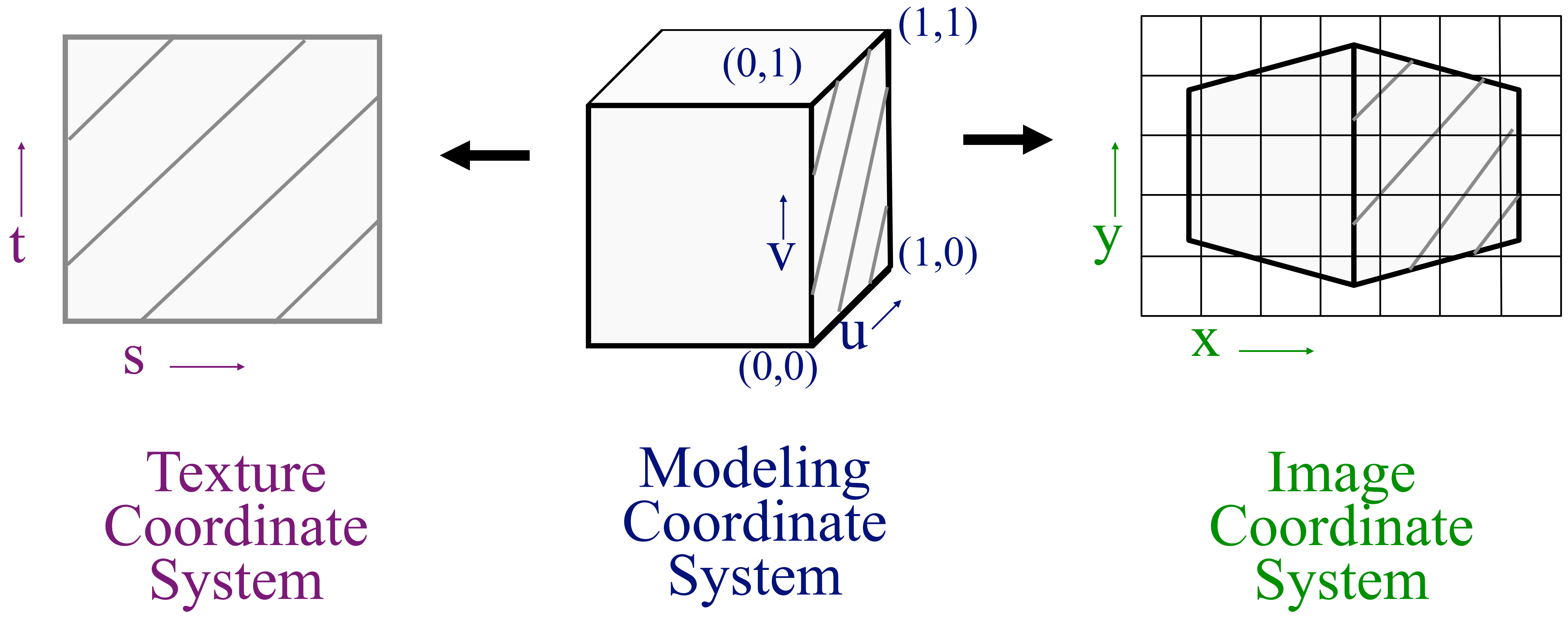
- Less distortion on each little piece of atlas
- Need to pack patches to reduce wasted space in texture image
- May be more difficult to think about the relationships between the different pieces

# Overview

- Texture mapping methods
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Shadow Maps
  - Volume textures

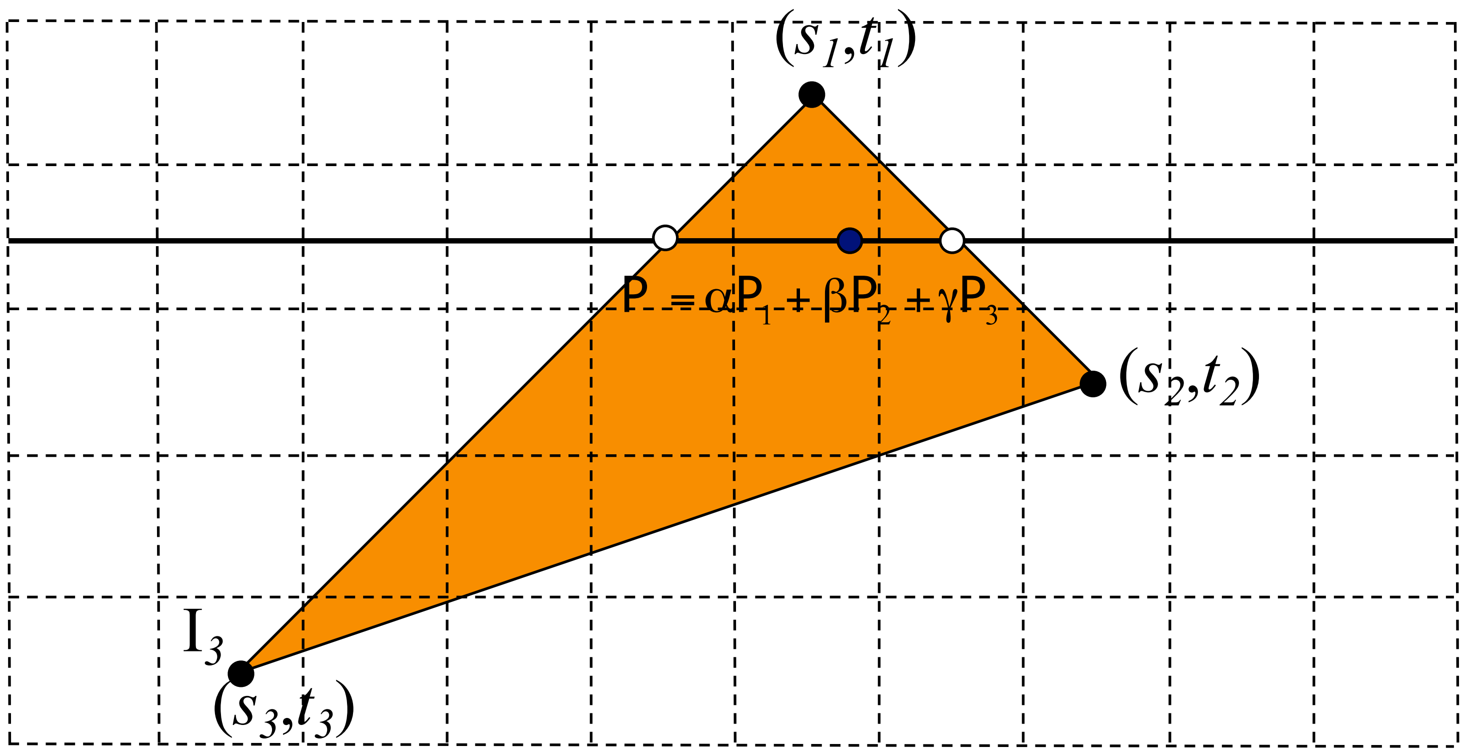
# Texture Mapping

- Steps:
  - Define texture
  - Specify mapping from surface to texture
  - Lookup texture values during scan conversion



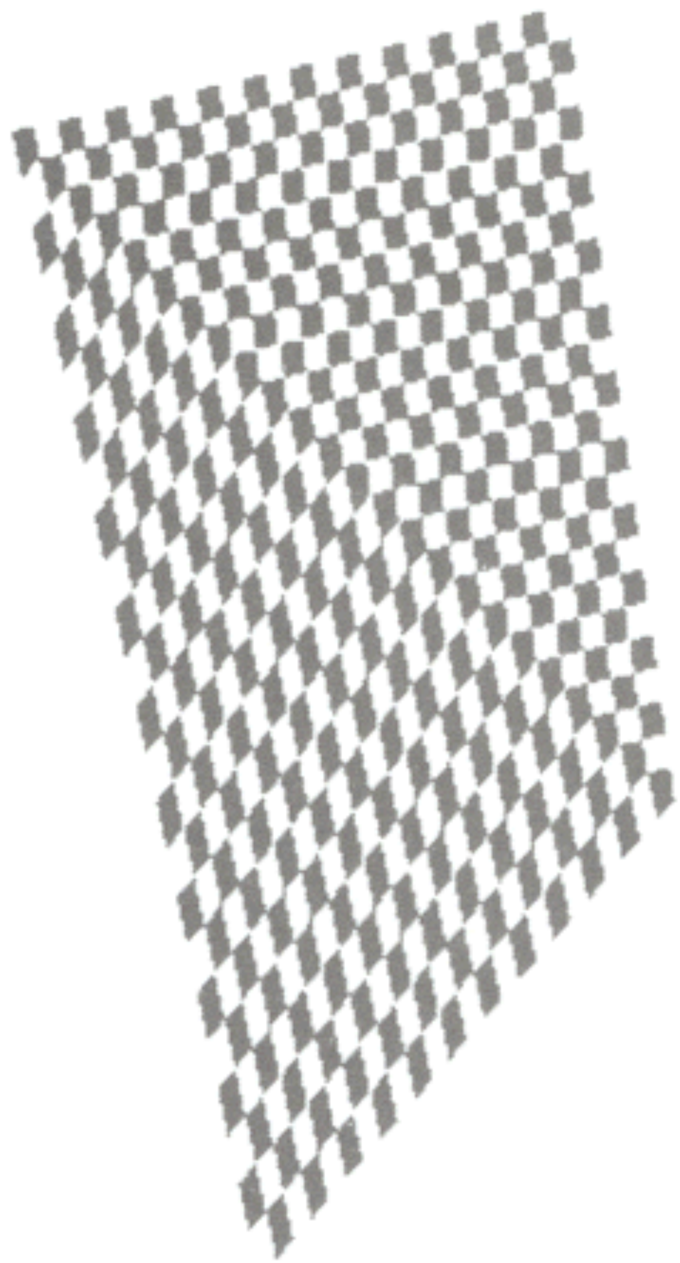
# Texture Mapping

- Scan conversion:
  - Interpolate texture coordinates down/across scan lines
  - Do perspective divide at each pixel based on mapping from screen space to 3-space

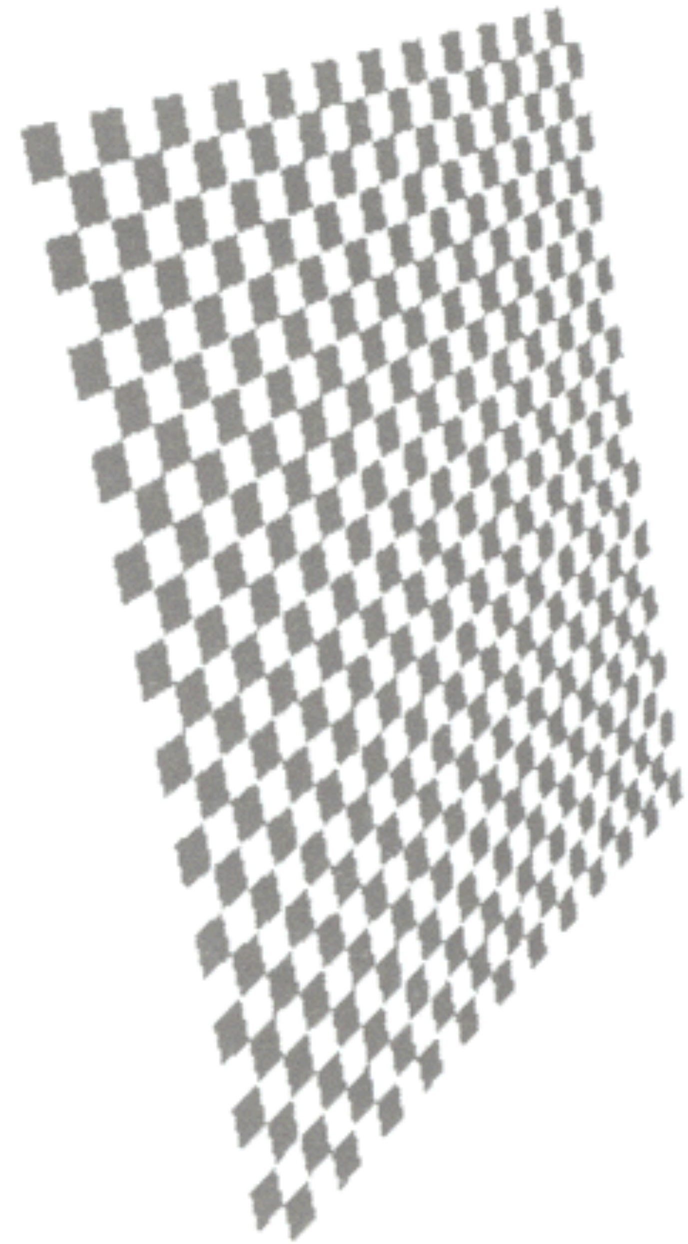



$$(s,t)=\alpha(s_1,t_1)+\beta(s_2,t_2)+\gamma(s_3,t_3)$$

# Texture Mapping



Linear interpolation
of texture coordinates
in screen space



Correct interpolation
with perspective divide

Hill Figure 8.42

# Perspective Correct Texture Mapping

From Wikipedia:

Perspective correct mapping interpolates after dividing by depth z, then uses its interpolated reciprocal to recover the correct coordinate:

$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

Here alpha is the interpolation parameter working in 2D screen space coordinates.

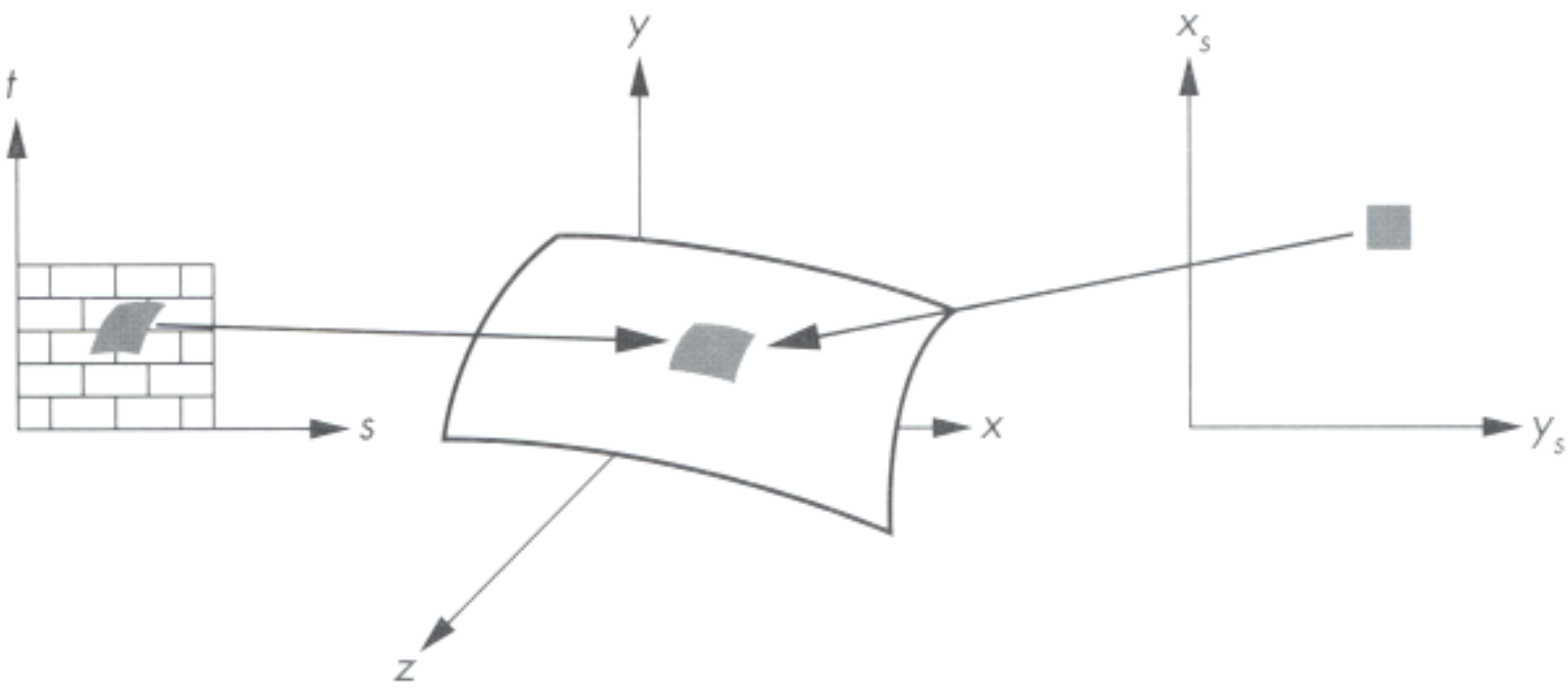
Hill Figure 8.42

# Overview

- Texture mapping methods
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Shadow maps
  - Volume Textures

# Texture Filtering

Must sample texture to determine color at each pixel in image

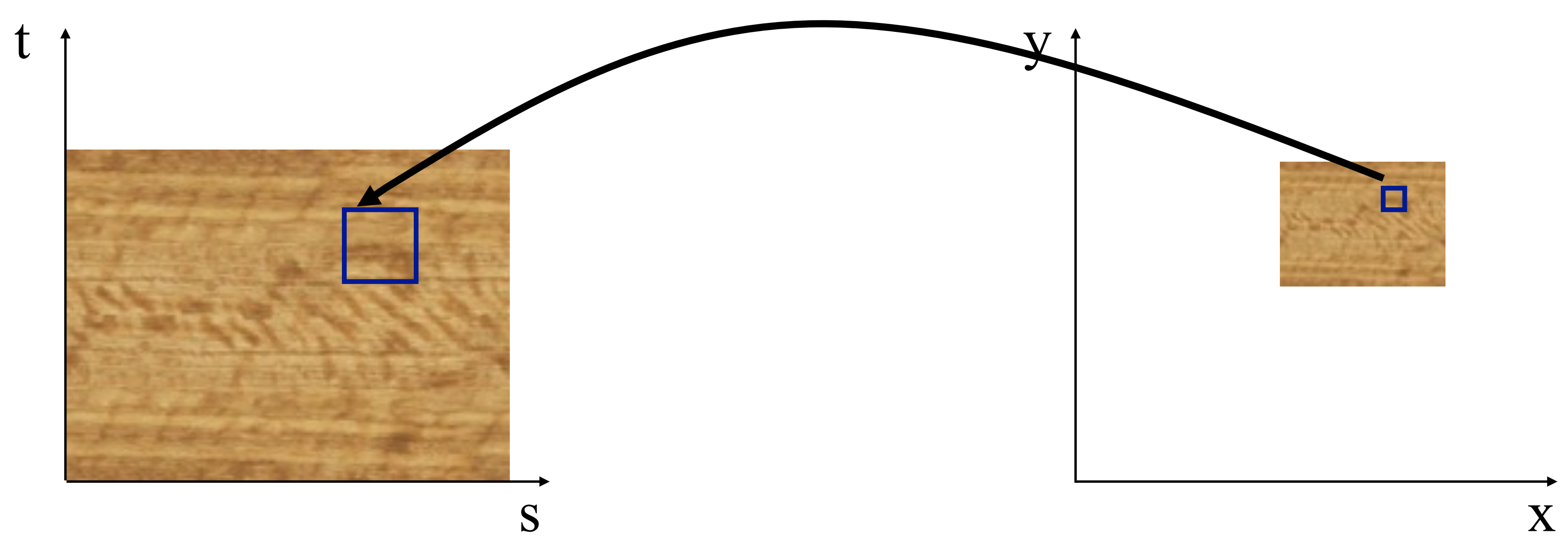



Angel Figure 9.4

# Texture Filtering

Must sample texture to determine color at each pixel in image

- In general, the transformation from screen space to texture space does not preserve area

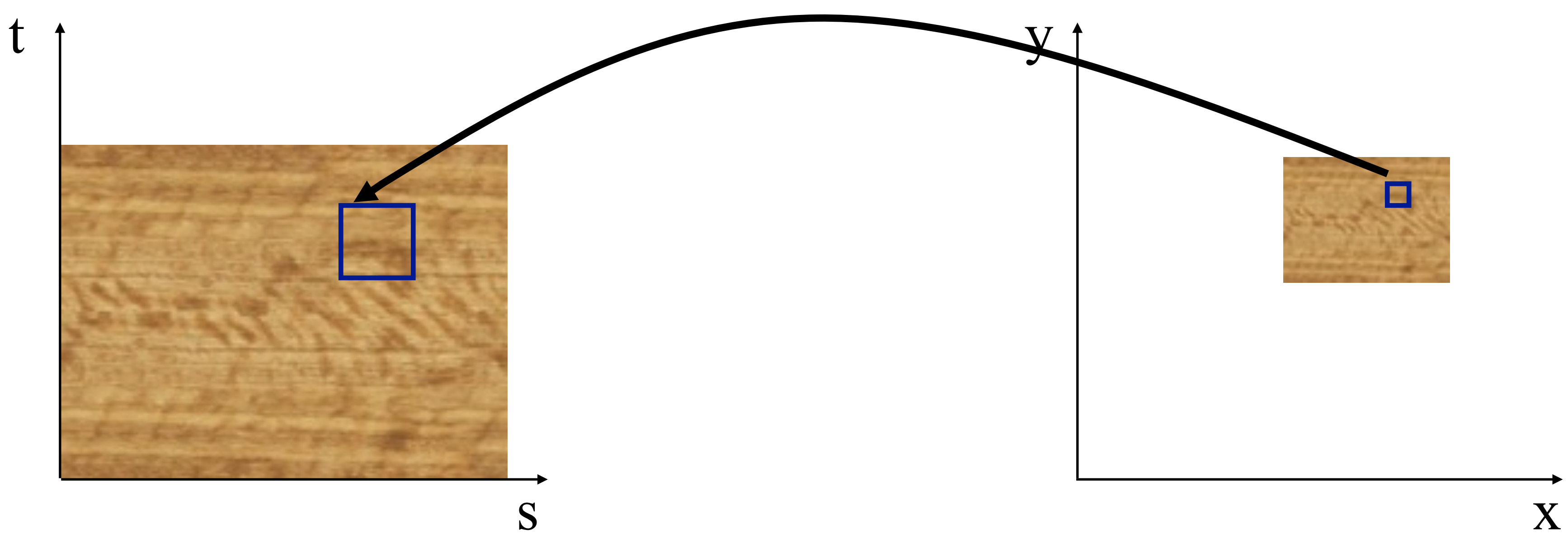


Minification

# Texture Filtering

Must sample texture to determine color at each pixel in image

- In general, the transformation from screen space to texture space does not preserve area
- Need to compute the average of the pixels in texture space to get the color for screen space

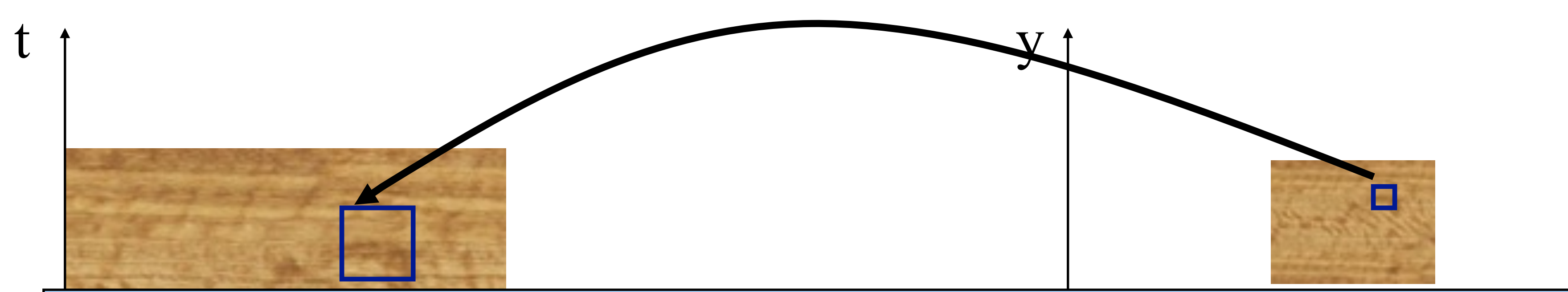



Minification

# Texture Filtering

Must sample texture to determine color at each pixel in image

- In general, the transformation from screen space to texture space does not preserver area
- Need to compute the average of the pixels in texture space to get the color for screen space



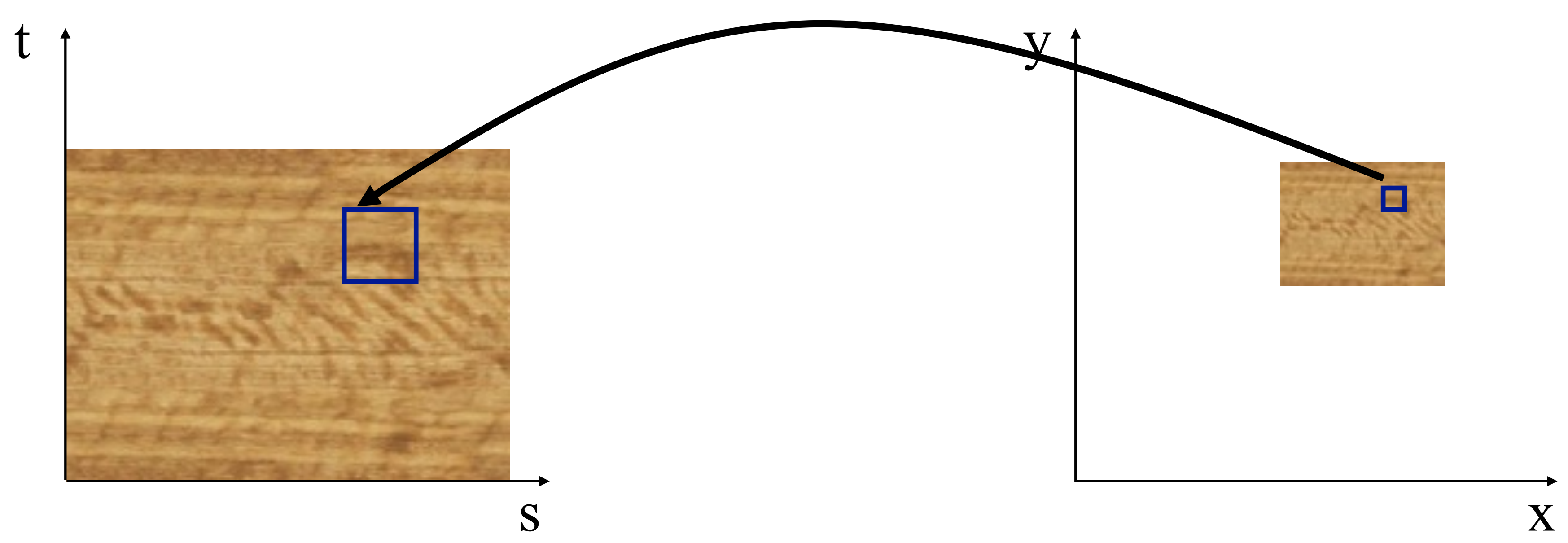

If the distortion is very large, this will require a lot of texture look-ups/adds.

Minification

# Texture Filtering

Size of filter depends on the projective deformation

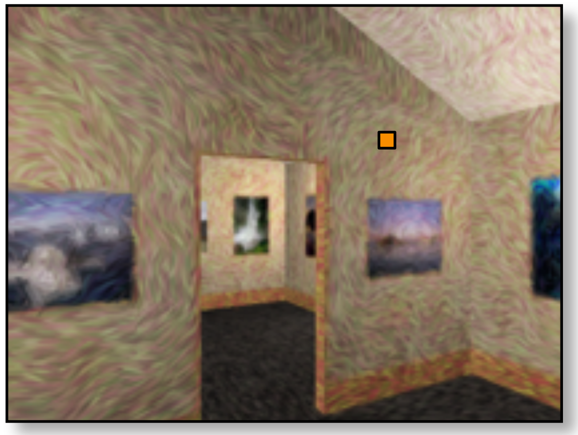
- Can prefilter images for better performance
  - Mip maps
  - Summed area tables

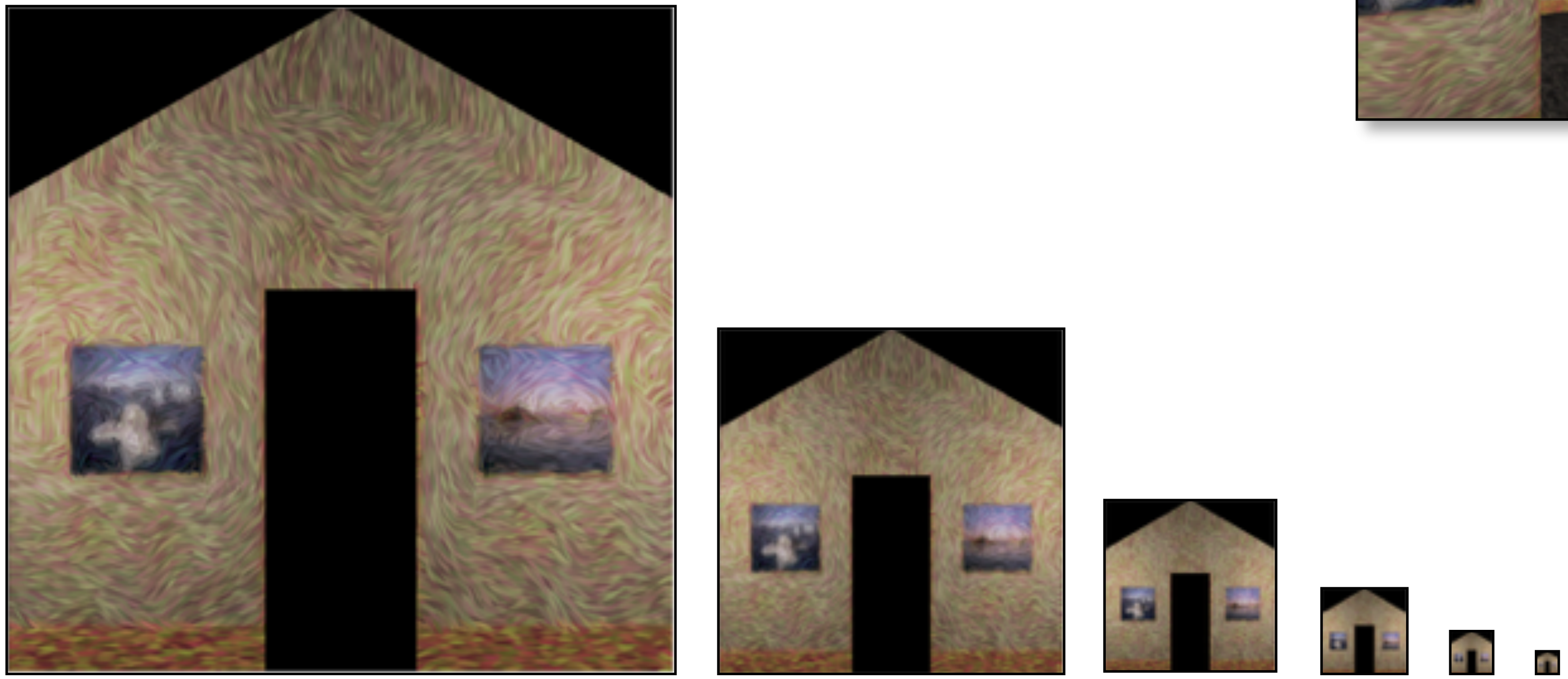



Minification

# Mip Maps

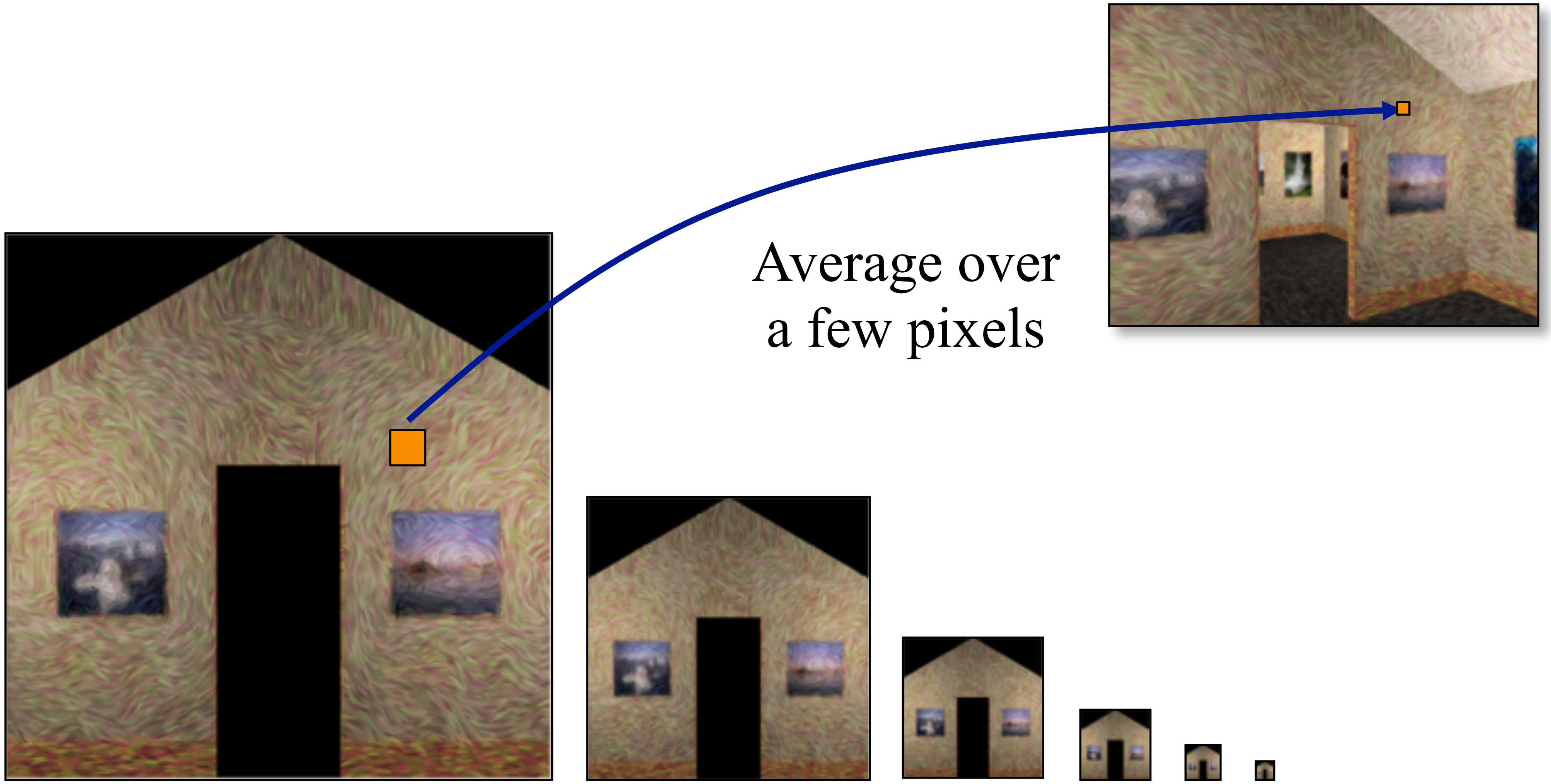
- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware
  - Similar to “Gaussian pyramid”





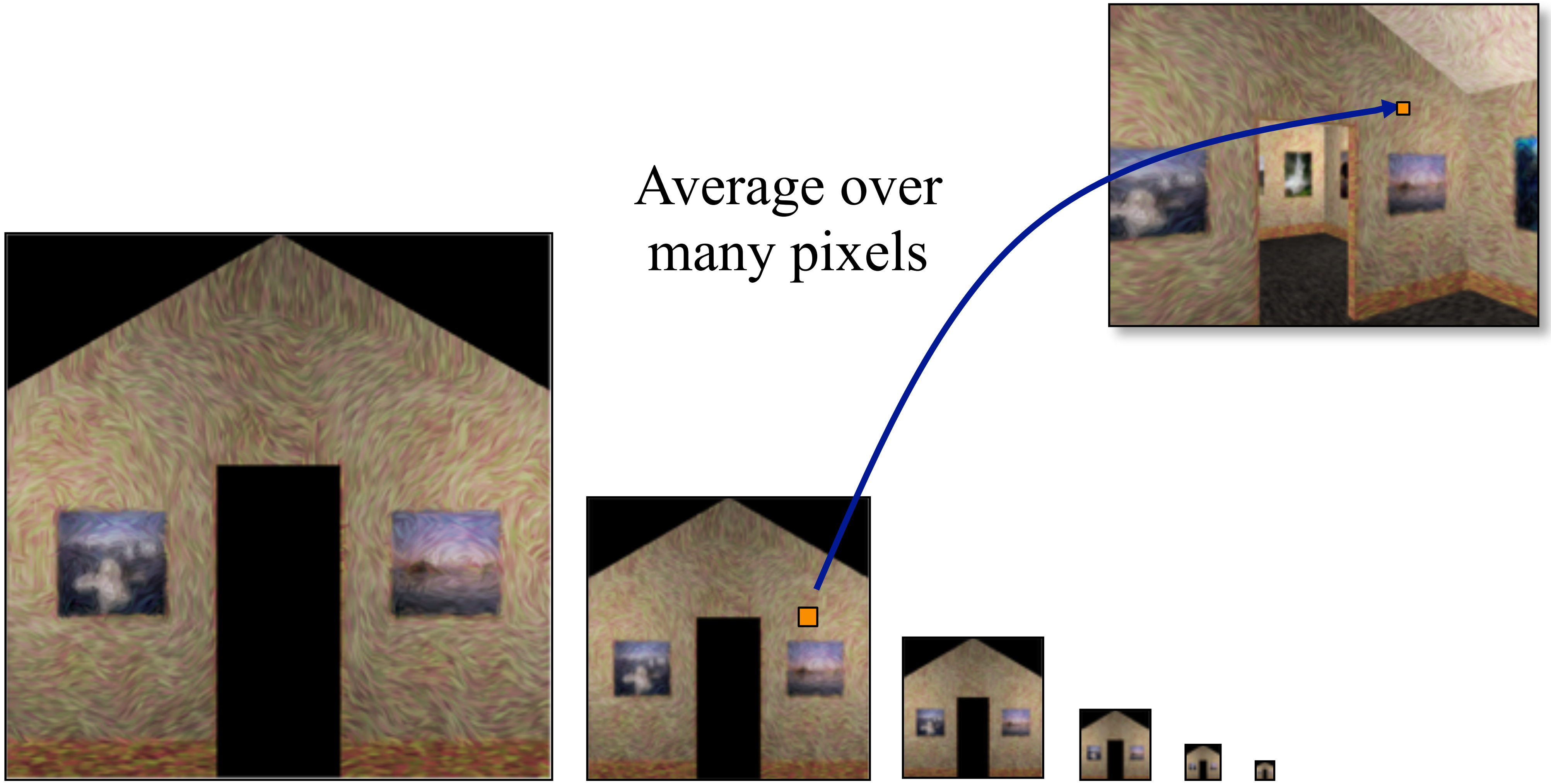
# Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware



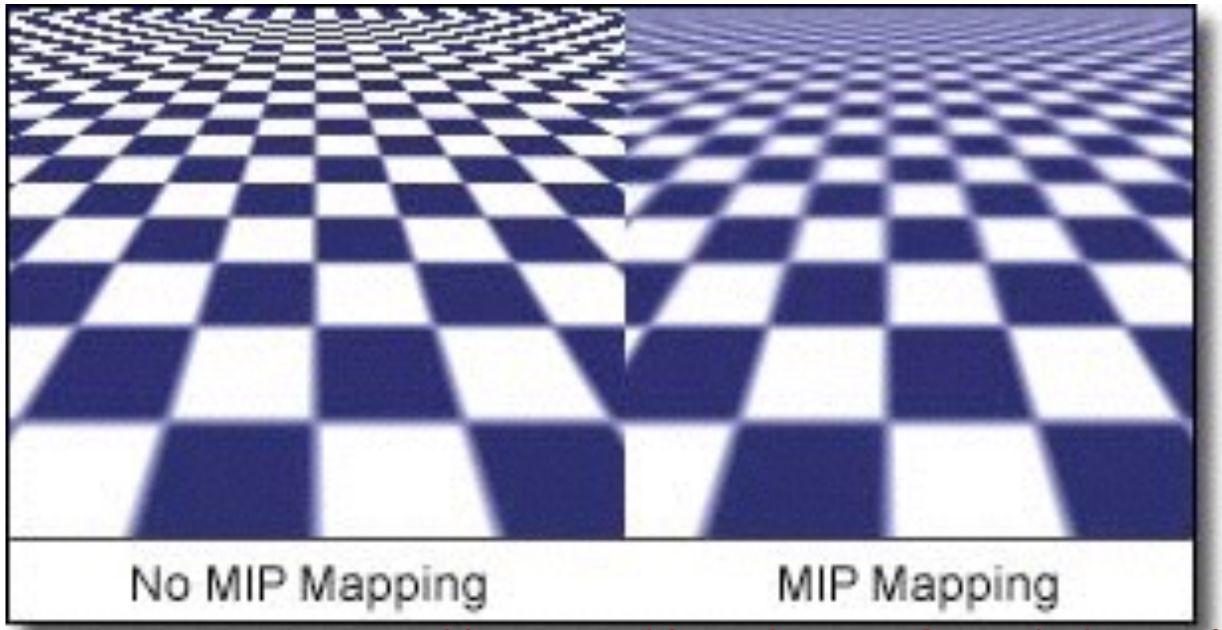
# Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware



# Mip Maps

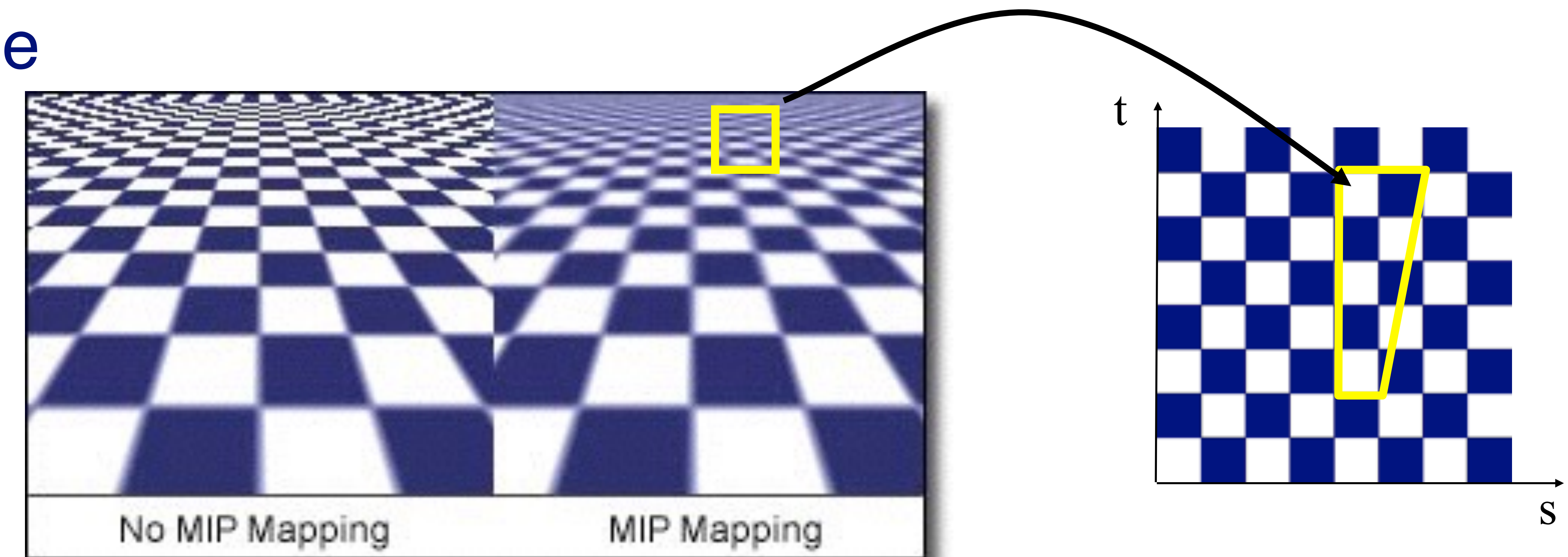
- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware




Again: we're trading aliasing for blurring!

# Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, use the mip-map closest level
  - Fast, easy for hardware
- This type of filtering is isotropic:
  - It doesn't take into account that there is more compression in the vertical direction than in the horizontal one



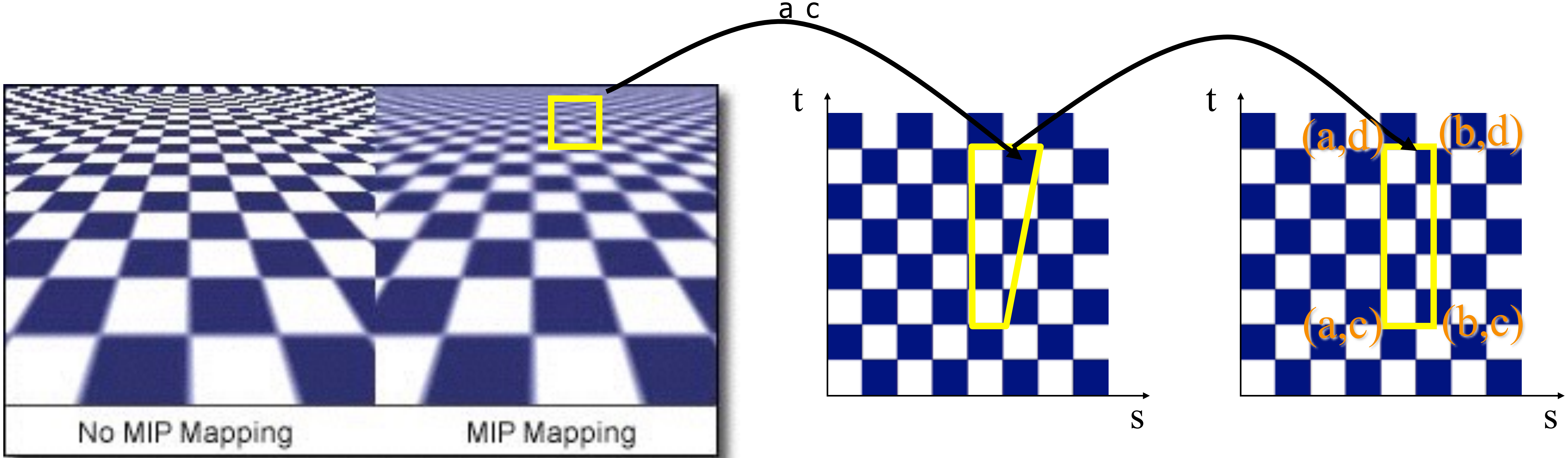

Again: we're trading aliasing for blurring!

# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

$$\mathrm{Sum}\big([a,b]\times[c,d]\big)=\int_a^b\int_c^d f(x,y)\,dy\,dx$$



# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.
- Perform the integration quickly by pre-computing integrals and leveraging the formula

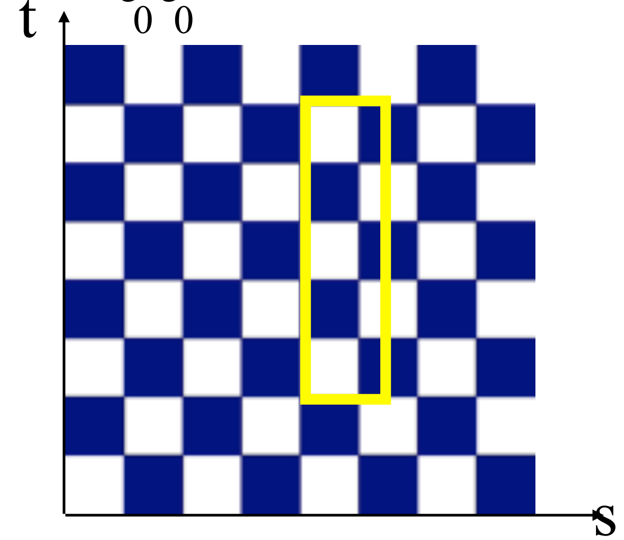
$$\int_a^b\int_c^d f(x,y)\,dy\,dx = \int_0^b\int_0^d f(x,y)\,dy\,dx - \int_0^b\int_0^c f(x,y)\,dy\,dx - \int_0^a\int_0^d f(x,y)\,dy\,dx + \int_0^a\int_0^c f(x,y)\,dy\,dx$$

# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.
- Perform the integration quickly by pre-computing integrals and leveraging the formula

$$\boxed{\int_a^b\int_c^d f(x,y)\,dy\,dx} = \int_0^b\int_0^d f(x,y)\,dy\,dx - \int_0^b\int_0^c f(x,y)\,dy\,dx - \int_0^a\int_0^d f(x,y)\,dy\,dx + \int_0^a\int_0^c f(x,y)\,dy\,dx$$

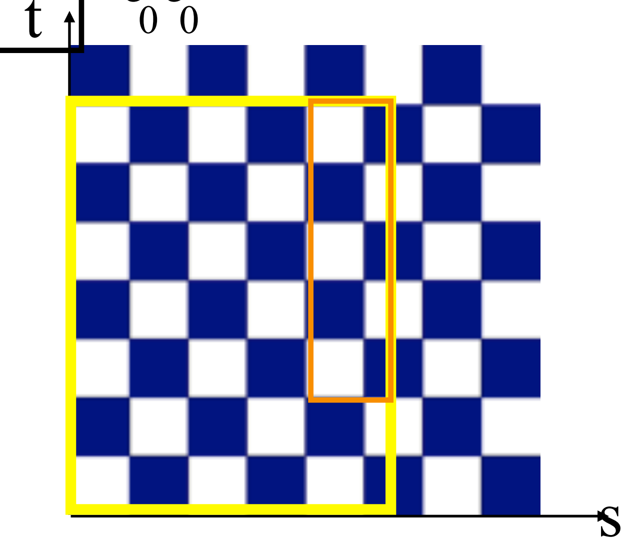


# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.
- Perform the integration quickly by pre-computing integrals and leveraging the formula

$$\int_a^b\int_c^d f(x,y)\,dy\,dx = \boxed{\int_0^b\int_0^d f(x,y)\,dy\,dx} - \int_0^b\int_0^c f(x,y)\,dy\,dx - \int_0^a\int_0^d f(x,y)\,dy\,dx + \int_0^a\int_0^c f(x,y)\,dy\,dx$$



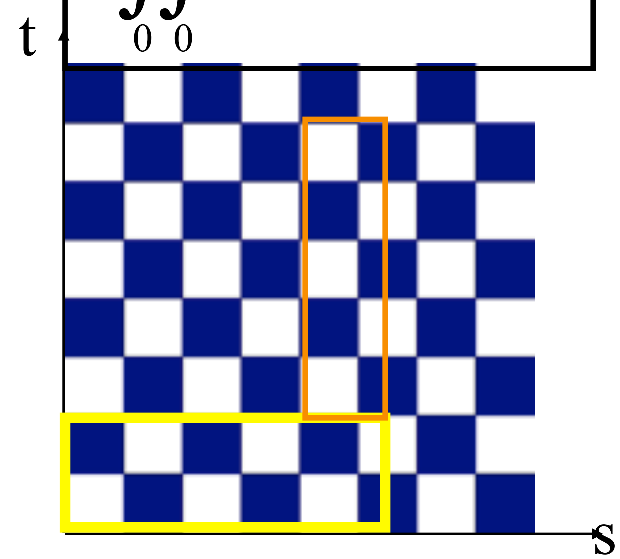
# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

- Perform the integration quickly by pre-computing integrals and leveraging the formula

$$\int_a^b\int_c^d f(x,y)\,dy\,dx = \int_0^b\int_0^d f(x,y)\,dy\,dx \boxed{-\int_0^b\int_0^c f(x,y)\,dy\,dx} - \int_0^a\int_0^d f(x,y)\,dy\,dx + \int_0^a\int_0^c f(x,y)\,dy\,dx$$



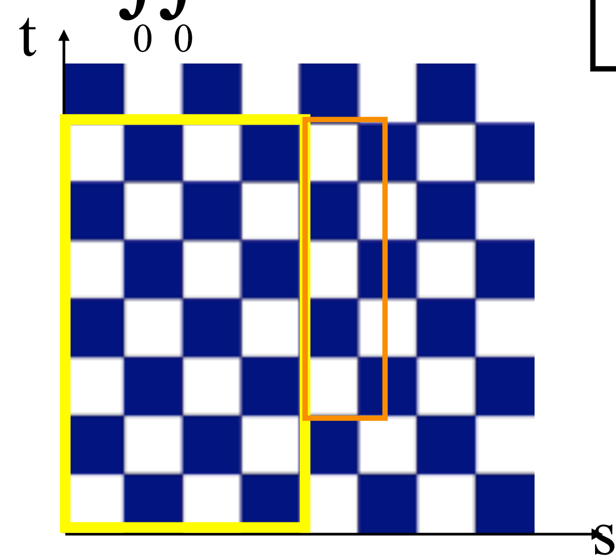
# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.

- Perform the integration quickly by pre-computing integrals and leveraging the formula

$$\int_a^b\int_c^d f(x,y)\,dy\,dx = \int_0^b\int_0^d f(x,y)\,dy\,dx - \int_0^b\int_0^c f(x,y)\,dy\,dx \boxed{-\int_0^a\int_0^d f(x,y)\,dy\,dx} + \int_0^a\int_0^c f(x,y)\,dy\,dx$$

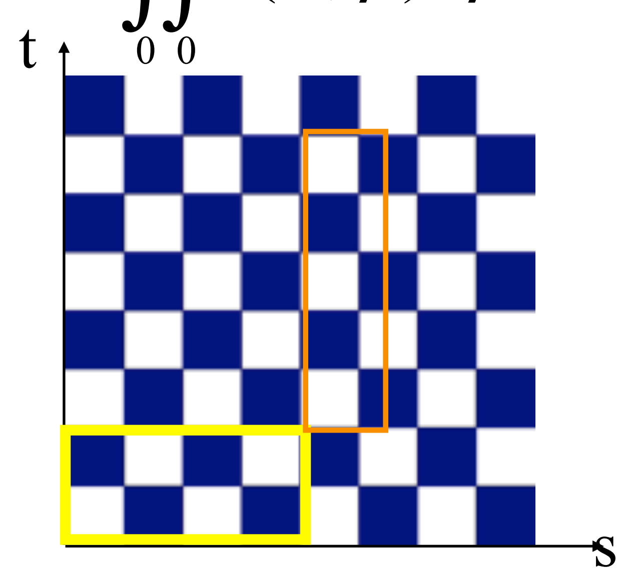


# Summed-area tables

Key Idea:

- Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle.
- Perform the integration quickly by pre-computing integrals and leveraging the formula

$$\int_a^b\int_c^d f(x,y)\,dy\,dx = \int_0^b\int_0^d f(x,y)\,dy\,dx - \int_0^b\int_0^c f(x,y)\,dy\,dx - \int_0^a\int_0^d f(x,y)\,dy\,dx + \boxed{\int_0^a\int_0^c f(x,y)\,dy\,dx}$$



# Summed-area tables

- Precompute the values of the integral:

$$S(a,b) = \int_0^a \int_0^b f(x,y)\,dy\,dx$$

- Each texel is the sum of all texels below and to the left of it

**Original image**

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

→

**Summed area table**

| | | | |
|---|---|---|---|
| 4 | 8 | 12 | 16 |
| 3 | 6 | 9 | 12 |
| 2 | 4 | 6 | 8 |
| 1 | 2 | 3 | 4 |

Courtesy Simon Green

# Summed-area tables

- Now, suppose I have some pixel *on screen* that maps to these pixels in my texture. What to do?
    - Explicitly computing the average (applying a box filter) is too slow!

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

**Original image**

# Summed-area tables

- Now, suppose I have some pixel *on screen* that maps to these pixels in my texture. What to do?
  - o Explicitly computing the average (applying a box filter) is too slow!
  - o Use summed-area table formula

$$\mathrm{Sum}([0,1]\times[3,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1)$$
$$= 16 - 8 - 4 + 2 = 6$$

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

**Original image**

| | | | |
|---|---|---|---|
| 4 | 8 | 12 | 16 |
| 3 | 6 | 9 | 12 |
| 2 | 4 | 6 | 8 |
| 1 | 2 | 3 | 4 |

**Summed-area table**

# Summed-area tables

- Now, suppose I have some pixel *on screen* that maps to these pixels in my texture. What to do?
  - Explicitly computing the average (applying a box filter) is too slow!
  - Use summed-area table formula

$$\mathrm{Sum}([0,1]\times[3,3]) = S(3,3) - S(0,3) - S(3,1) + S(0,1)$$
$$= 16 - 8 - 4 + 2 = 6$$
$$\mathrm{Average}([0,1]\times[3,3]) = \mathrm{Sum}([0,1]\times[3,3])/\mathrm{Area}([0,1]\times[3,3]) = 6/6 = 1$$

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

**Original image**

| | | | |
|---|---|---|---|
| 4 | 8 | 12 | 16 |
| 3 | 6 | 9 | 12 |
| 2 | 4 | 6 | 8 |
| 1 | 2 | 3 | 4 |

**Summed-area table**

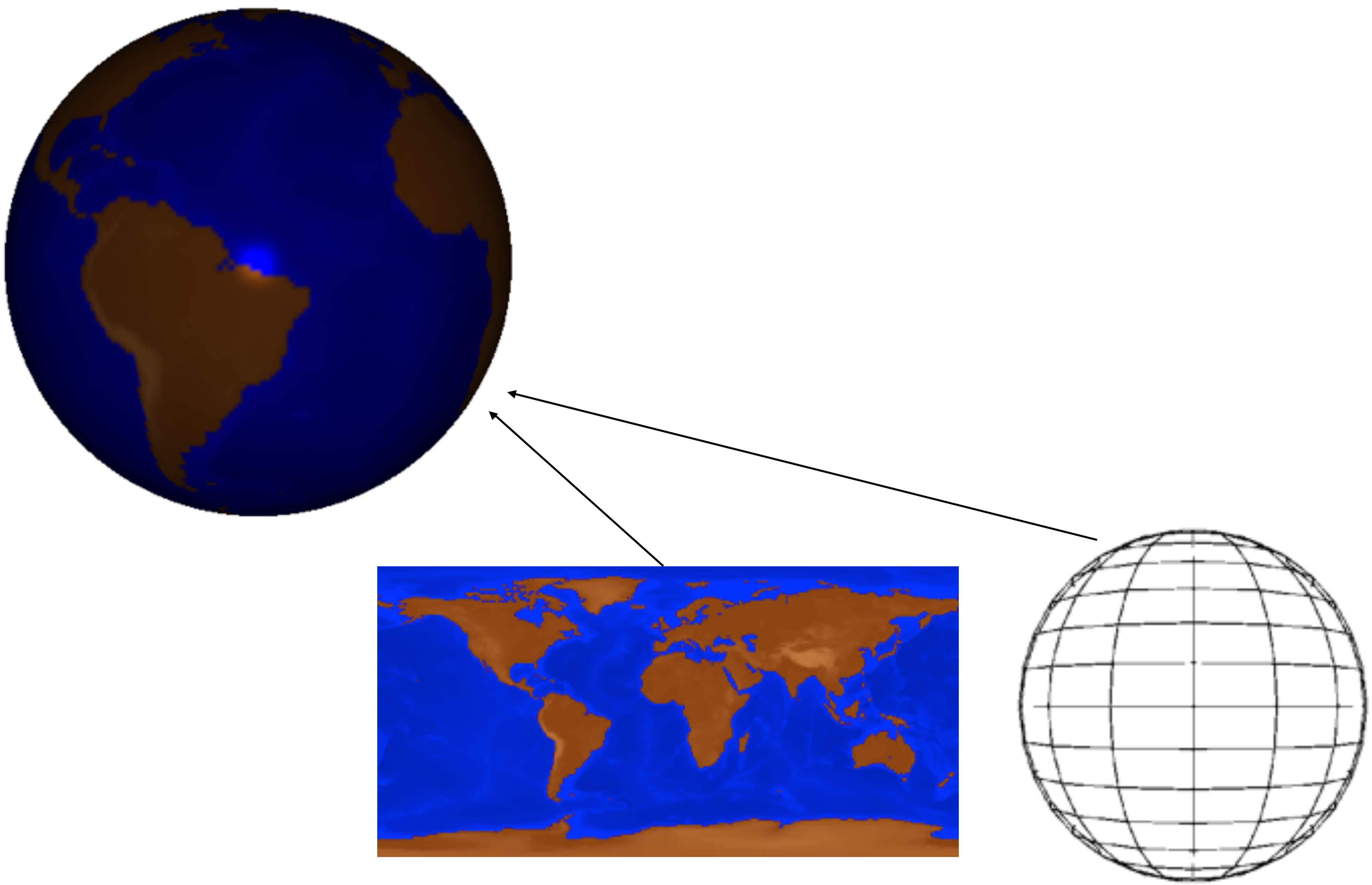
# Overview

- Texture mapping methods
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Volume Textures

# Modulation textures

Map texture values to scale factor

Modulation

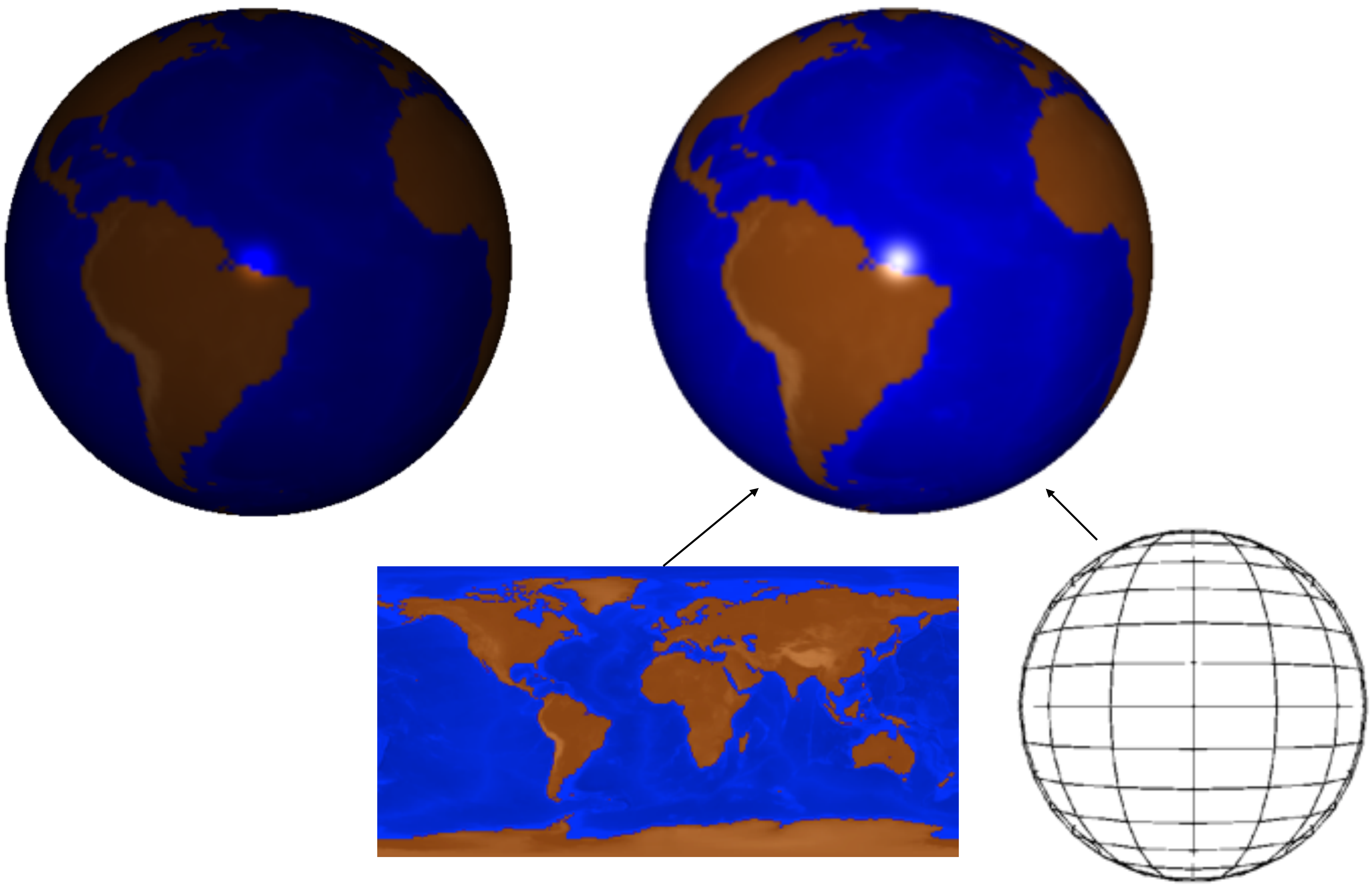


$$I = \boxed{T(s,t)}\left(I_E + K_A I_A + \sum_L (K_D(N \bullet L) + K_S(V \bullet R)^n) S_L I_L + K_T I_T + K_S I_S\right)$$

# Illumination Mapping

Map texture values to any material parameter

Modulation Diffuse

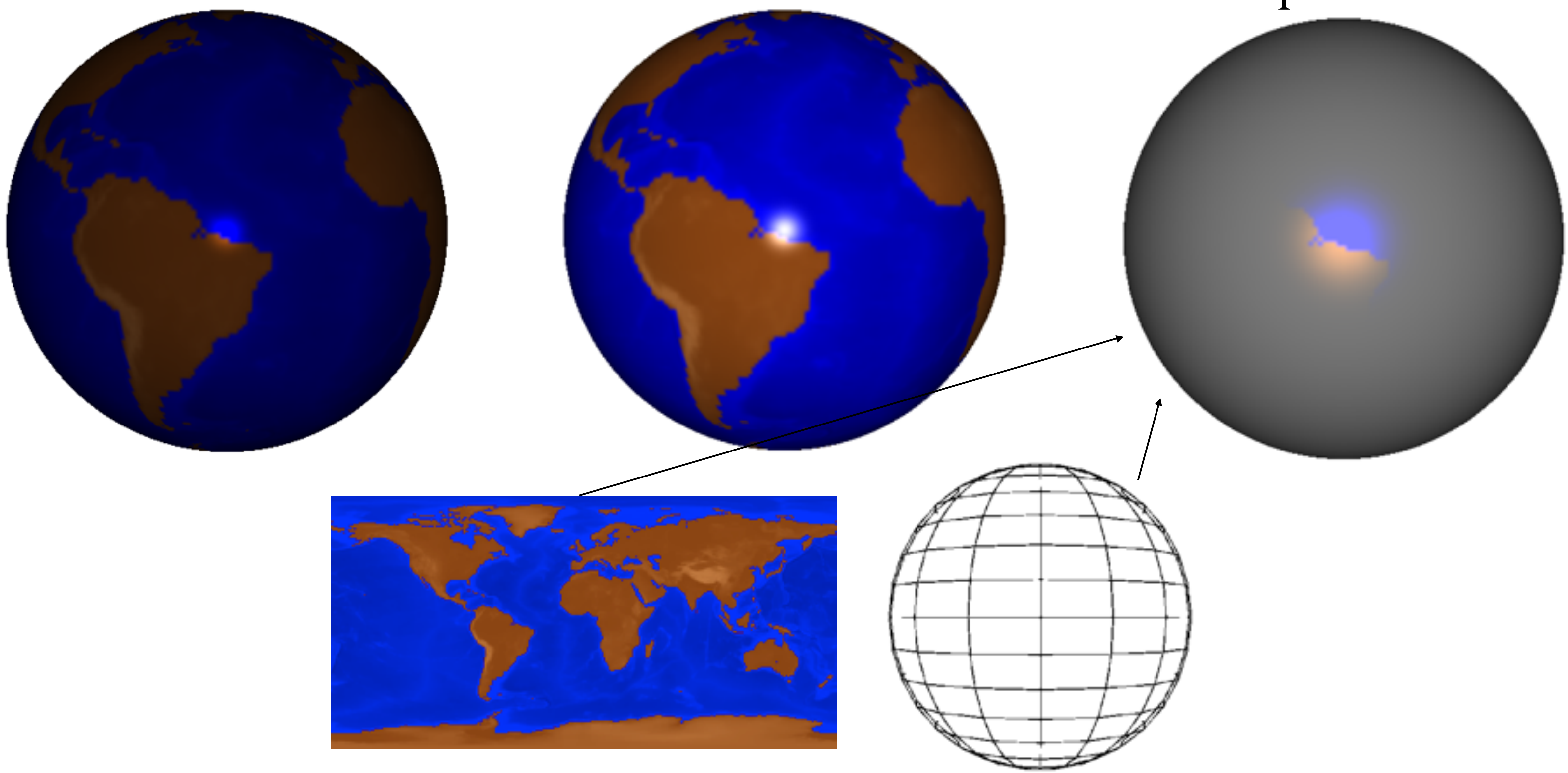


$$I = I_E + K_A I_A + \sum_L \left(\boxed{T(s,t)}(N \bullet L) + K_S (V \bullet R)^n\right) S_L I_L + K_T I_T + K_S I_S$$

# Illumination Mapping

Map texture values to any material parameter

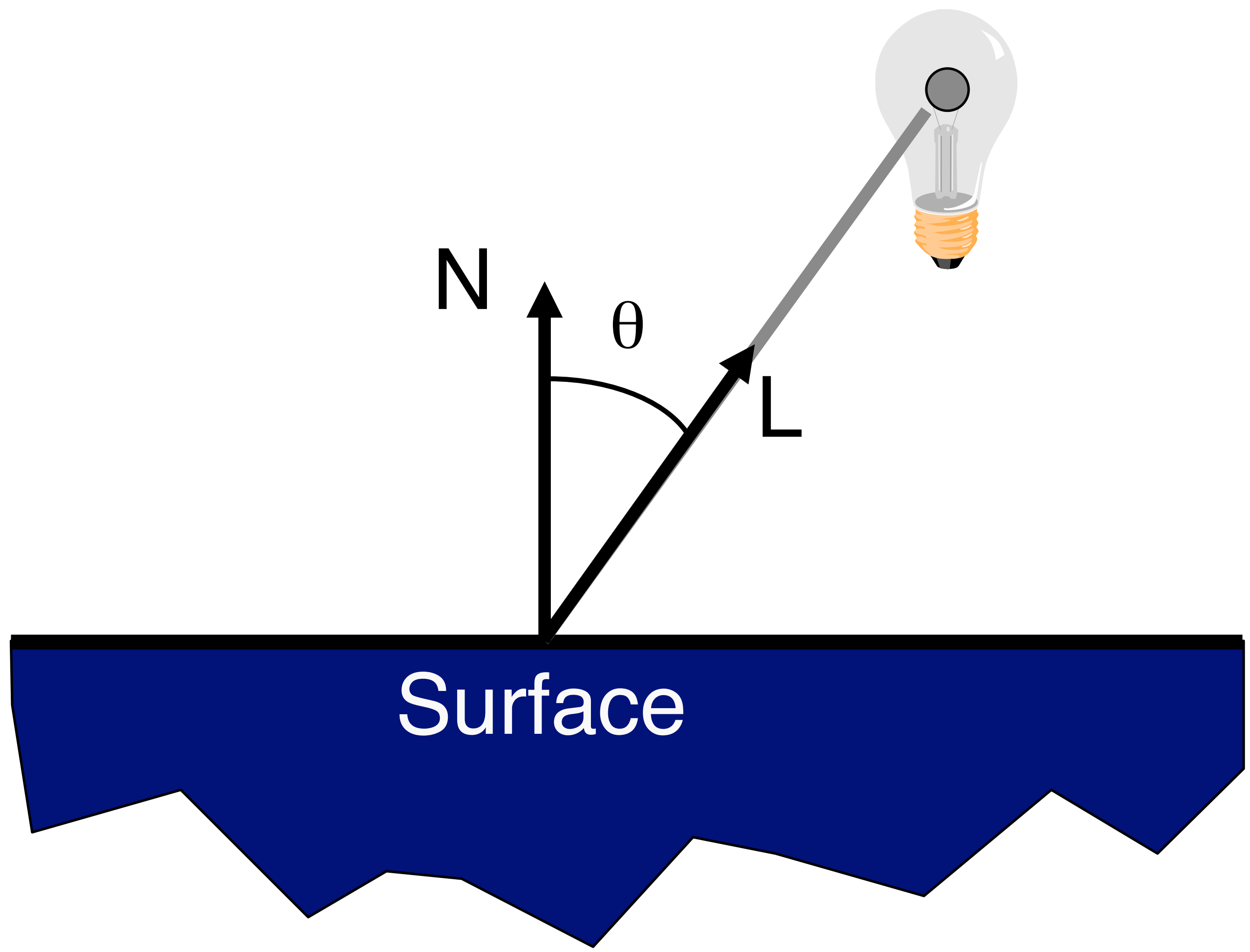
Modulation Diffuse Specular



$$I = I_E + K_A I_A + \sum_L \left( K_D (N \bullet L) + \boxed{T(s,t)} (V \bullet R)^n \right) S_L I_L + K_T I_T + K_S I_S$$

# Bump Mapping

- Recall that many parts of our lighting calculation depend on surface normals



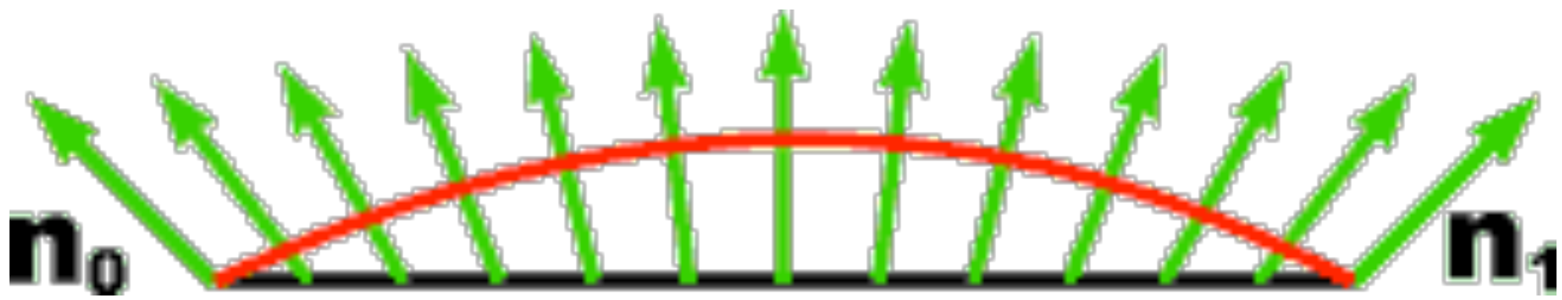

$$I = I_E + K_A I_A + \sum_L \left( K_D (N \bullet L) + K_S (V \bullet R)^n \right) S_L I_L + K_T I_T + K_S I_S$$

# Bump Mapping



P. Rheingans

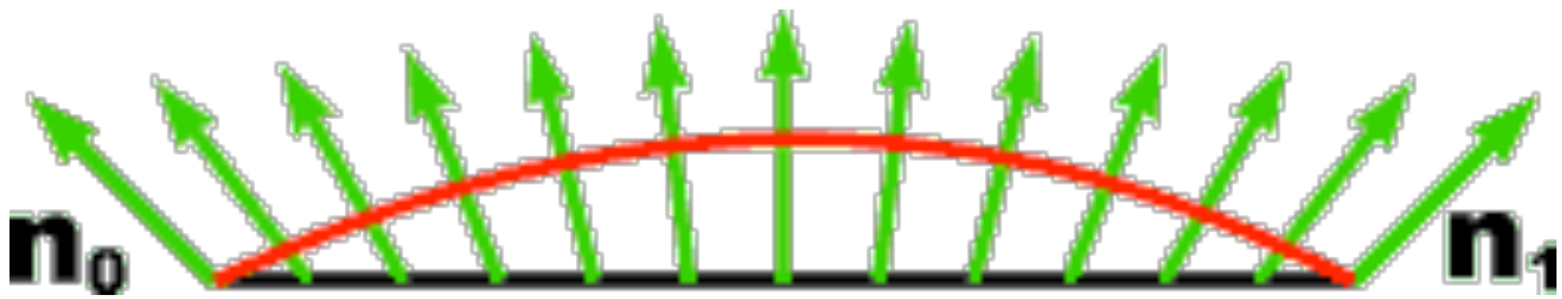
# Bump Mapping



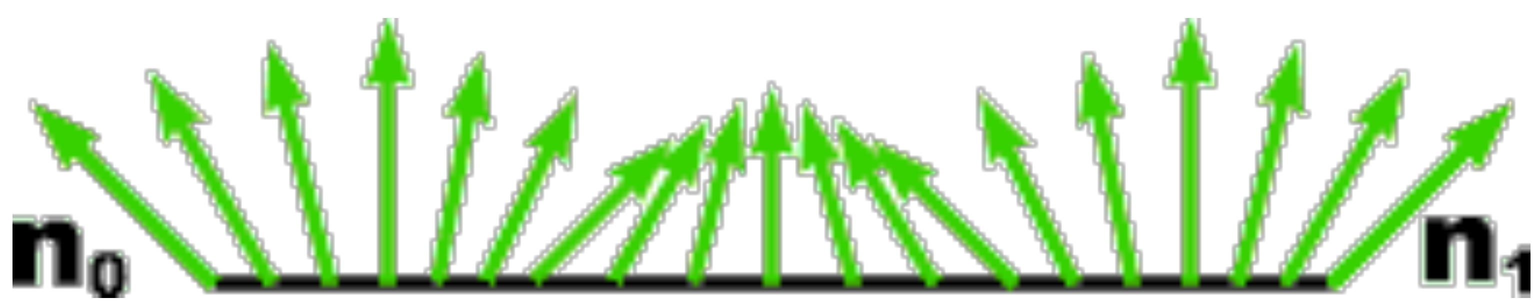

Phong shading approximates smoothly curved surface


P. Rheingans

# Bump Mapping



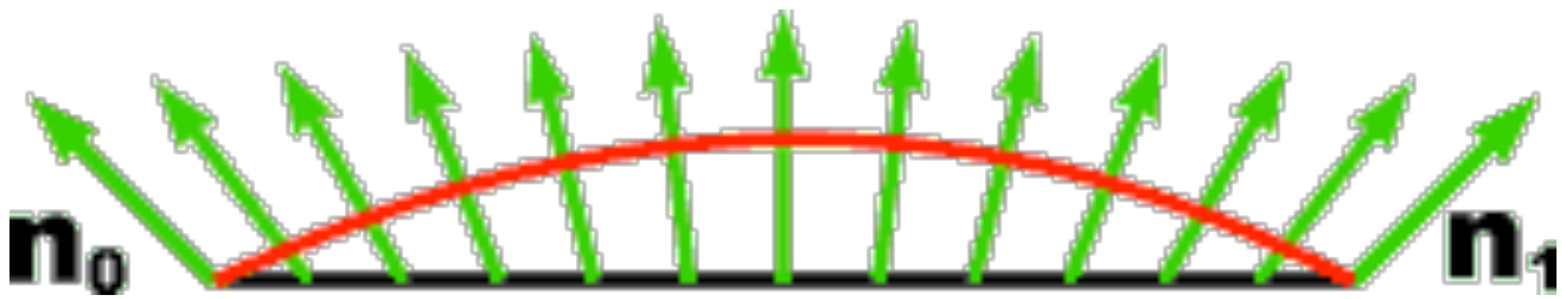

Phong shading approximates smoothly curved surface



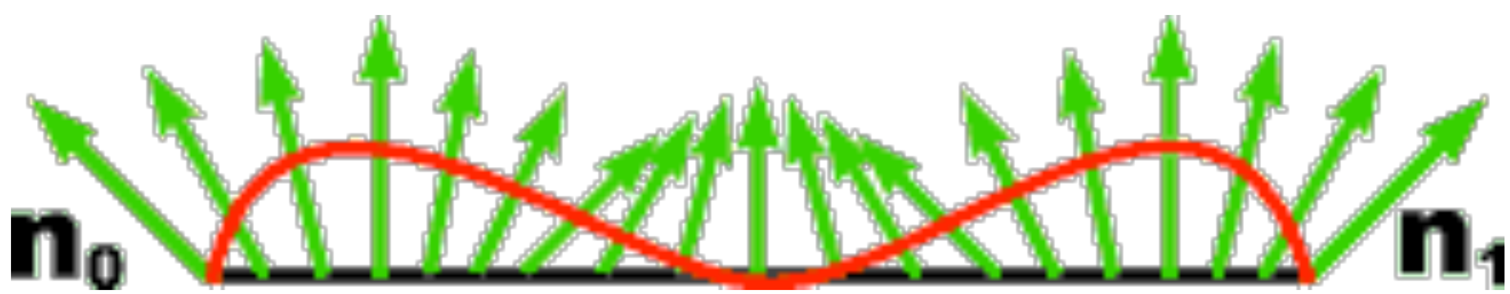

We can store perturbations to normals in a texture map

P. Rheingans

# Bump Mapping



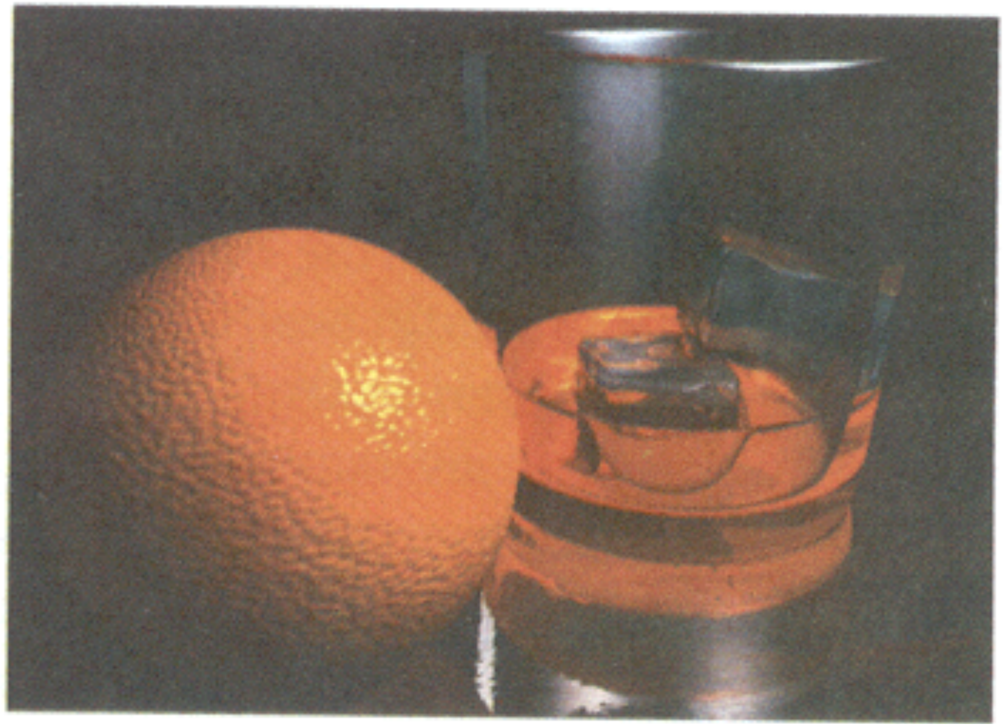

Phong shading approximates smoothly curved surface




Now Phong shading gives the appearance of a bumpy surface

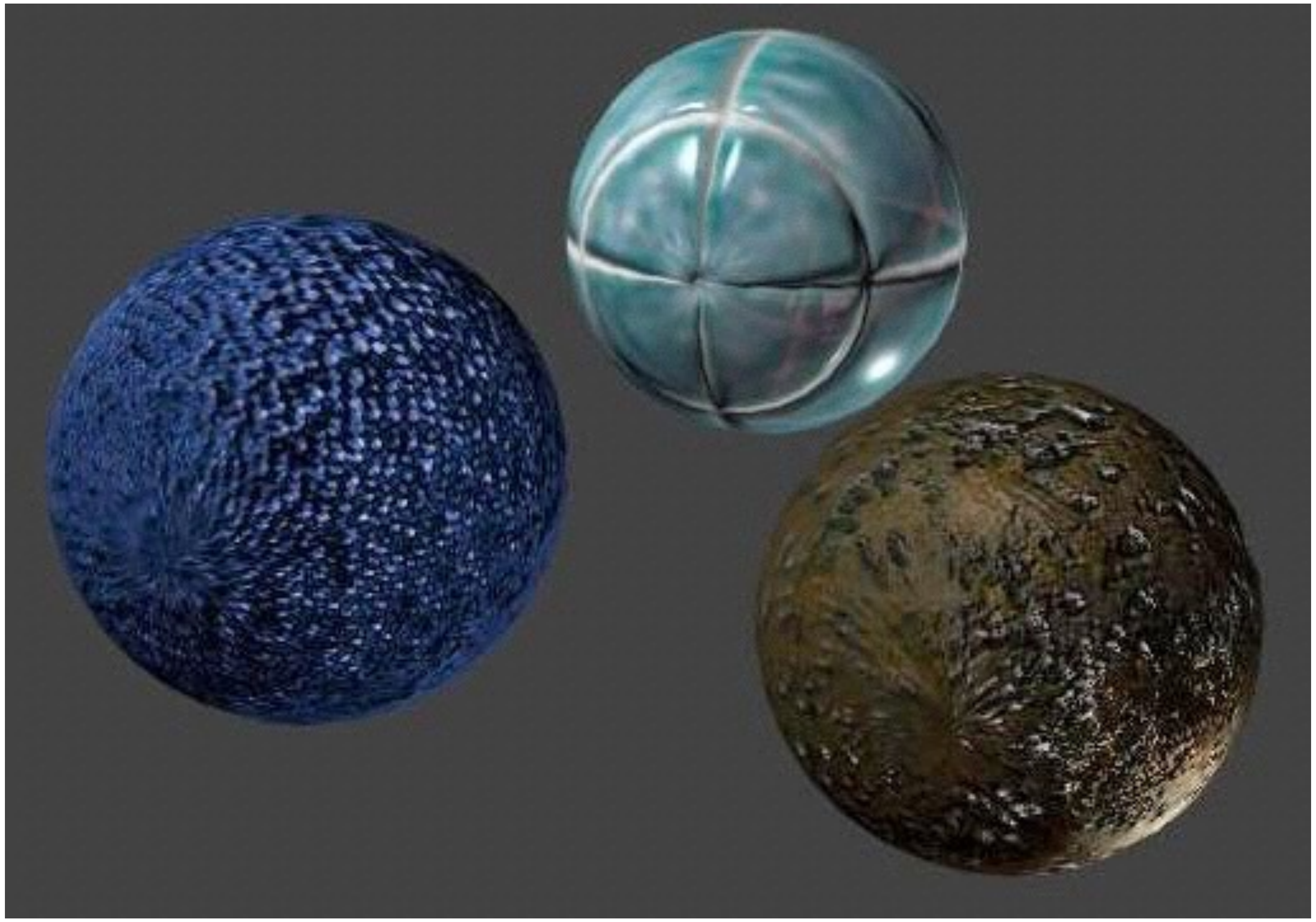
P. Rheingans

# Bump Mapping



H&B Figure 14.100

# Bump Mapping

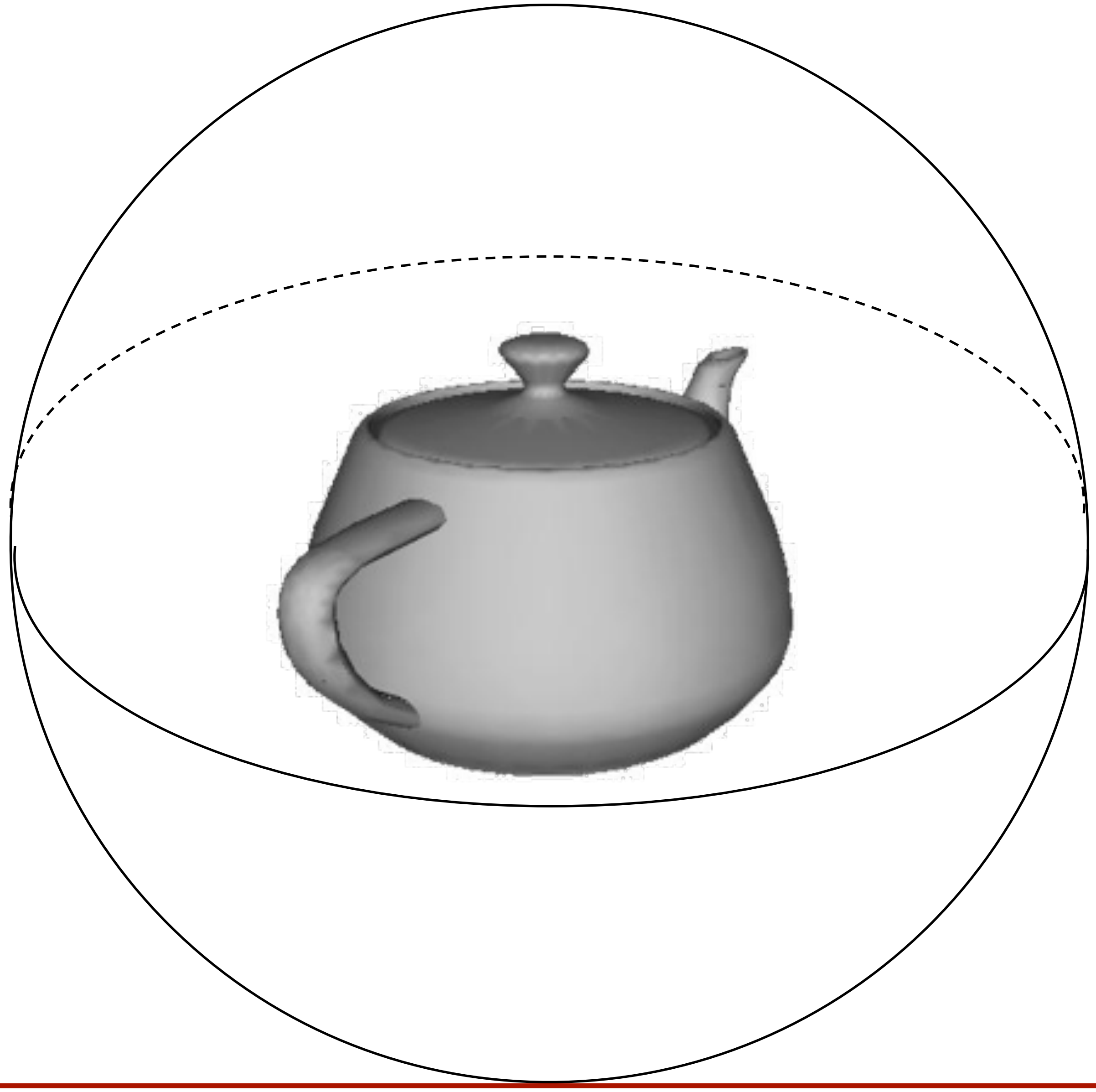


Note that bump mapping
does not change object silhouette

Siggraph.org

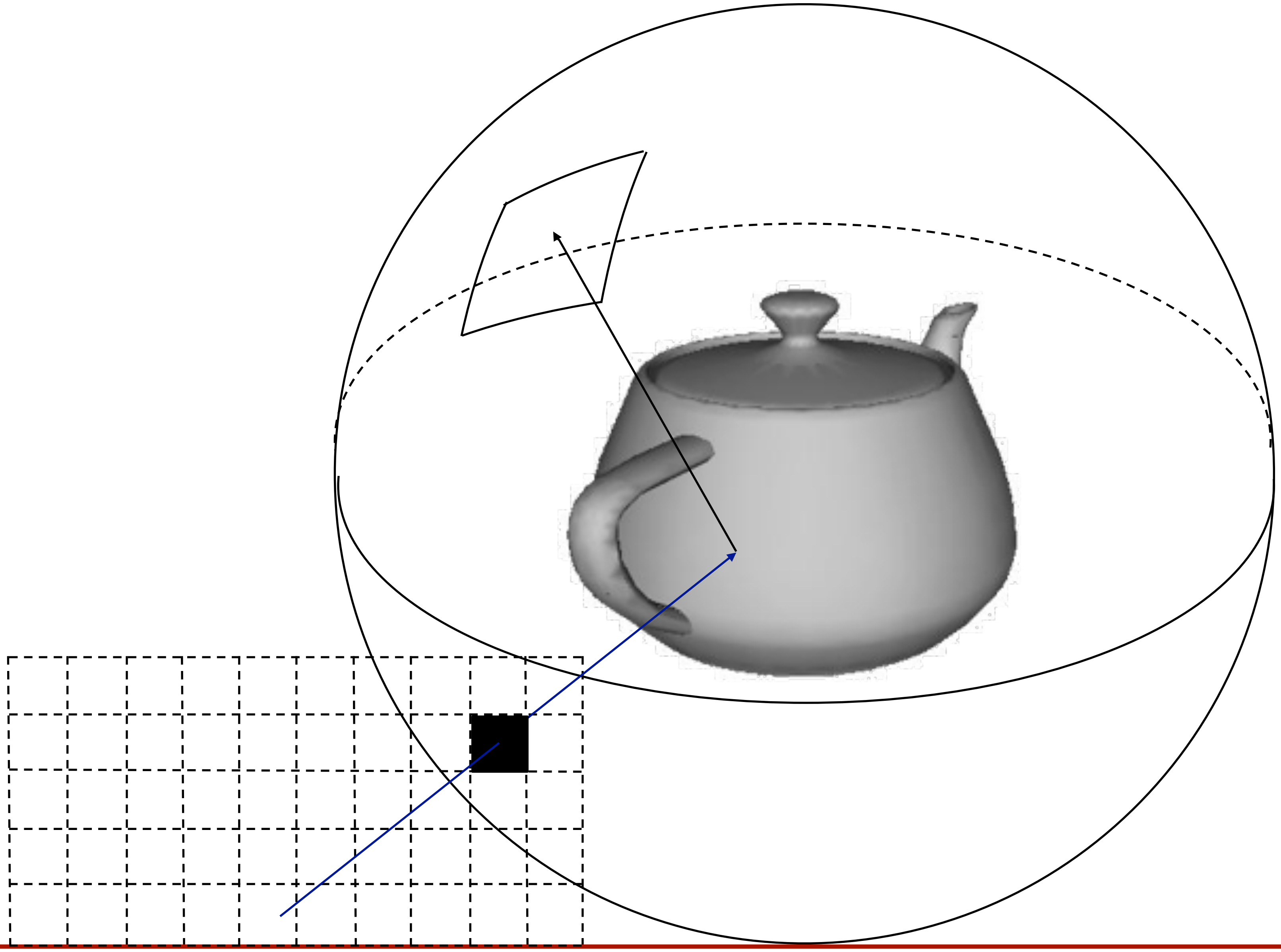
# Environment Mapping

- Generate a spherical/cubic map of the environment around the model.



# Environment Mapping

- Generate a spherical/cubic map of the environment around the model.
- Texture values are reflected off surface patch

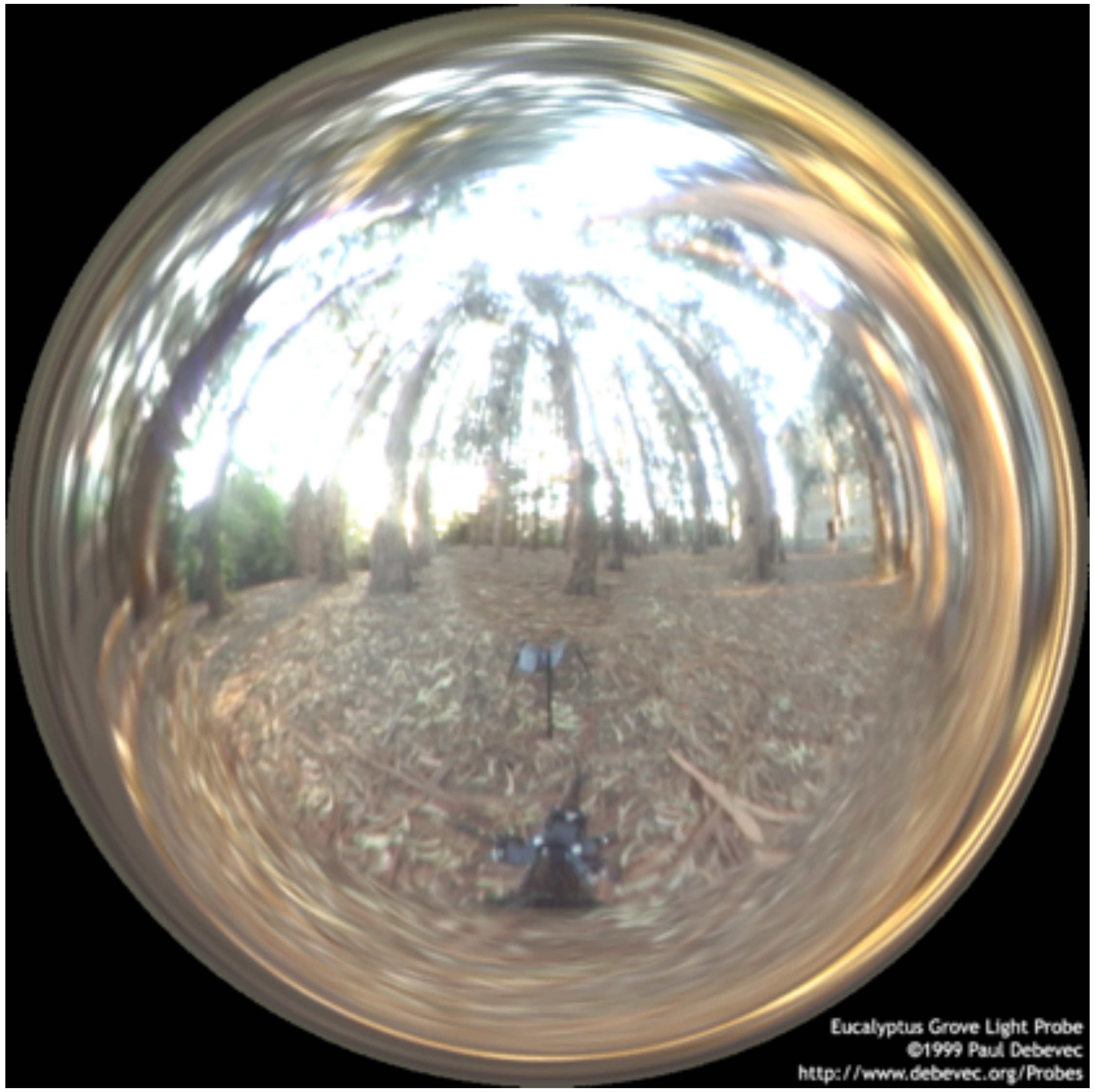


# Environment Mapping

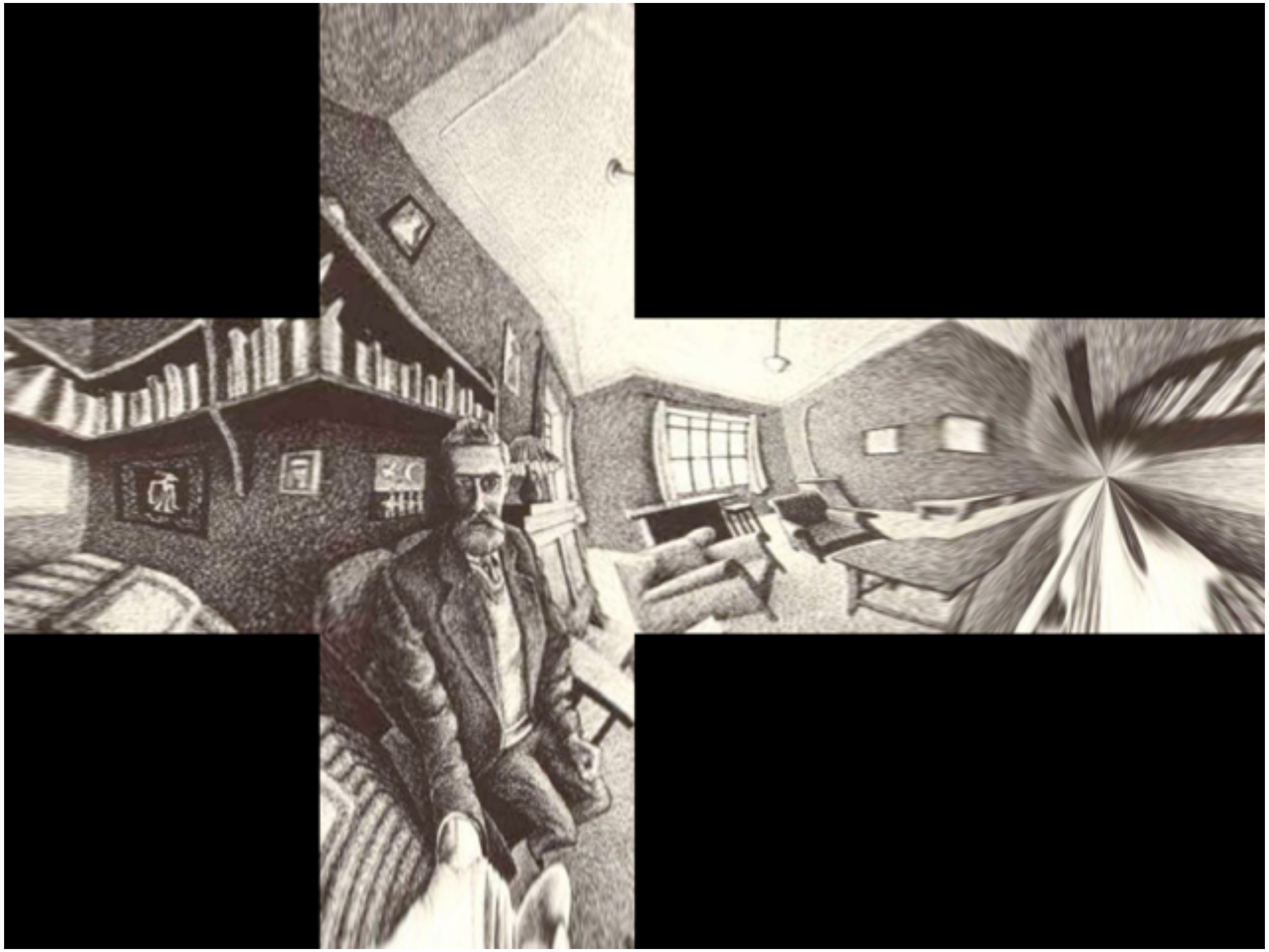
Texture values are reflected off surface patch

P. Debevec

# Environment Maps / Light Probes



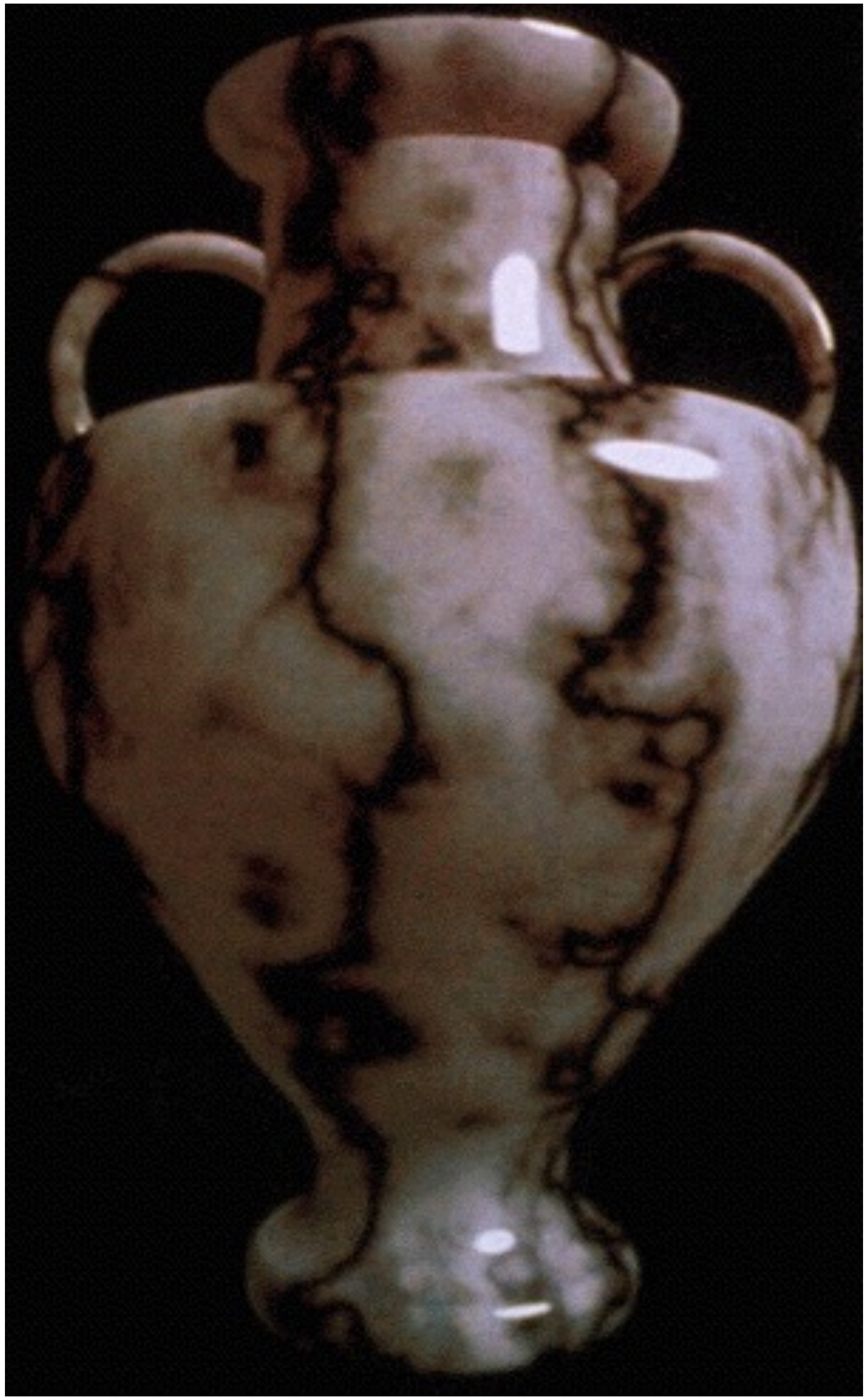
# Cube Maps



# Solid textures

Texture values indexed by 3D location (x,y,z)

- Expensive storage, or
- Compute on the fly, e.g. Perlin noise →



# 3D Rendering Pipeline (for direct illumination)