Topics

Buffers

Texture mapping

Buffers

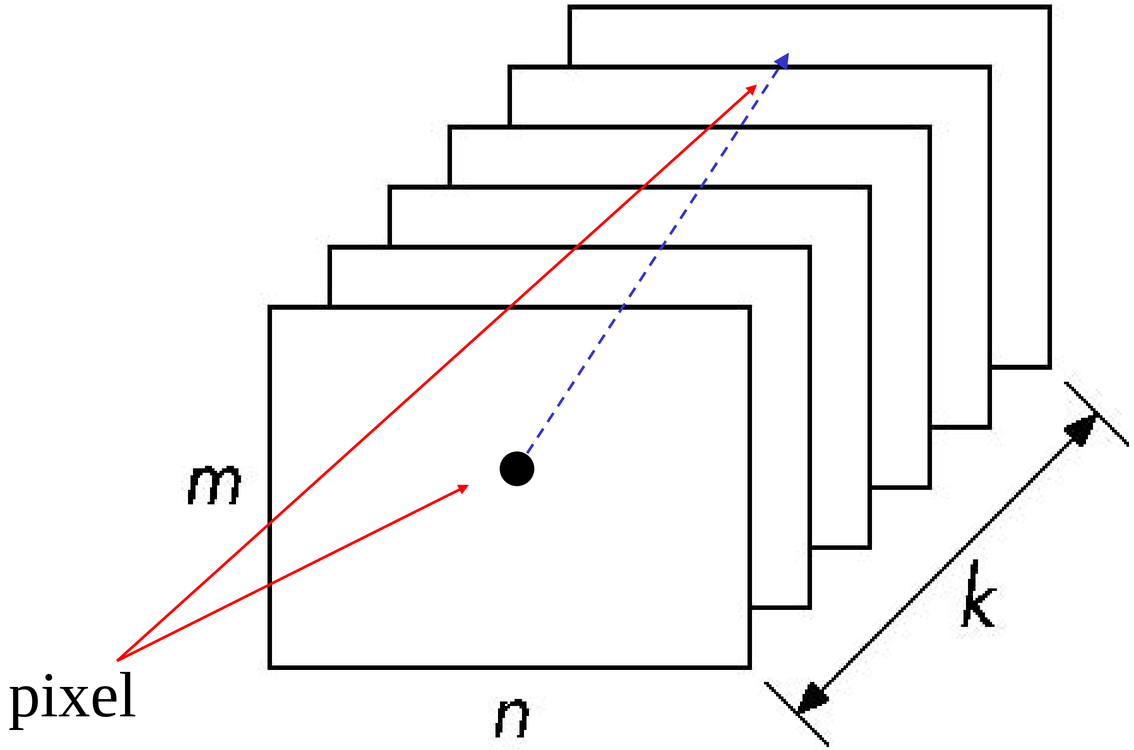
Introduce additional OpenGL buffers

Learn to read from buffers

Learn to use blending

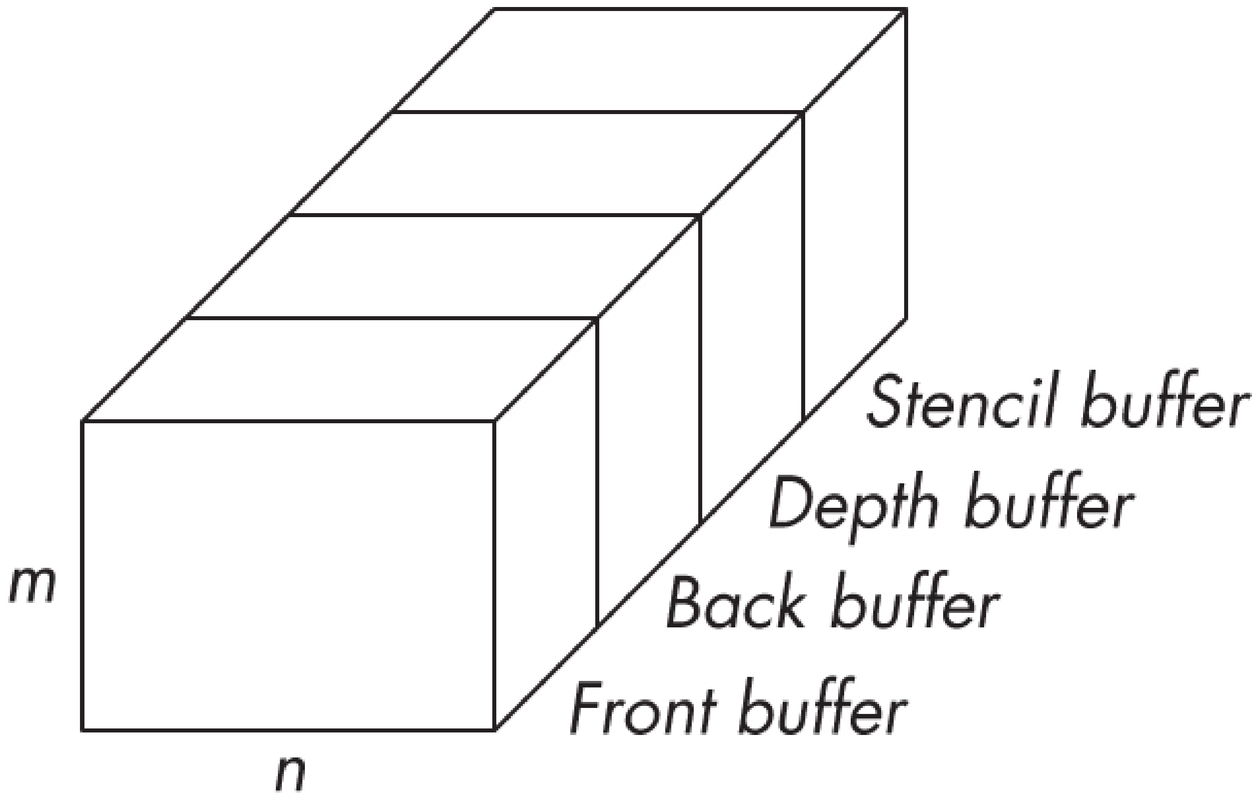
Buffer

Define a buffer by its spatial resolution (n x m) and its depth (or precision) k, the number of bits/pixel



OpenGL Frame Buffer

64 bits for front and back buffers



OpenGL Buffers

Color buffers can be displayed

- Front
- Back
- Stereo

Depth

Stencil

- Holds masks (per-pixel integers) to control rendering

Most RGBA buffers 8 bits per component

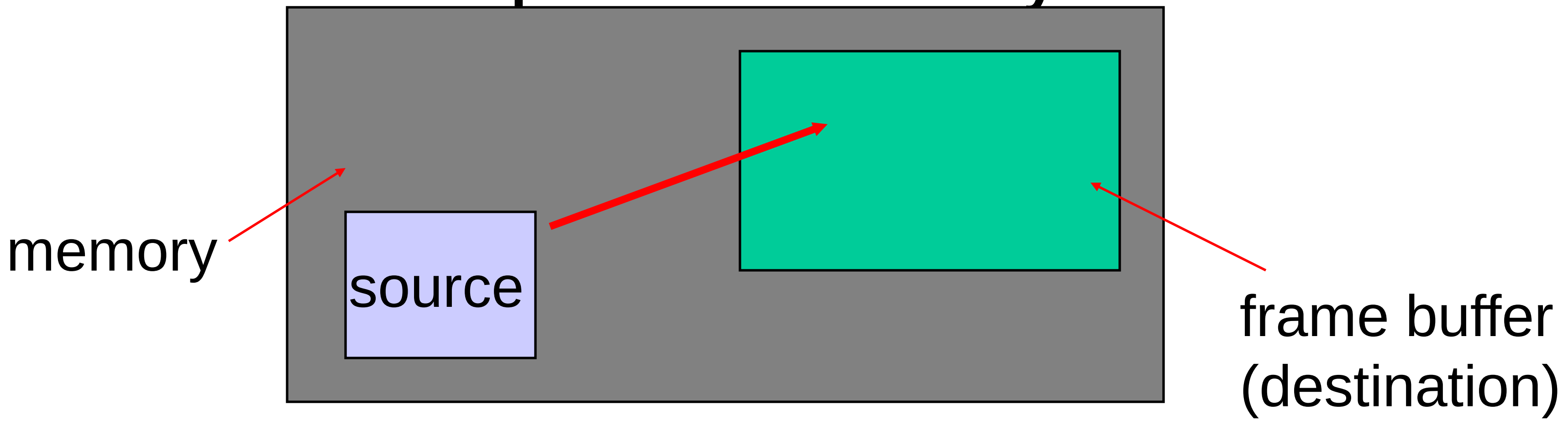
Writing in Buffers

Conceptually, we can consider all of memory as a large two-dimensional array of pixels

In practice, we read and write rectangular blocks of pixels

- *Bit block transfer* (*bitblt*) *operations*

The frame buffer is part of this memory



writing into frame buffer

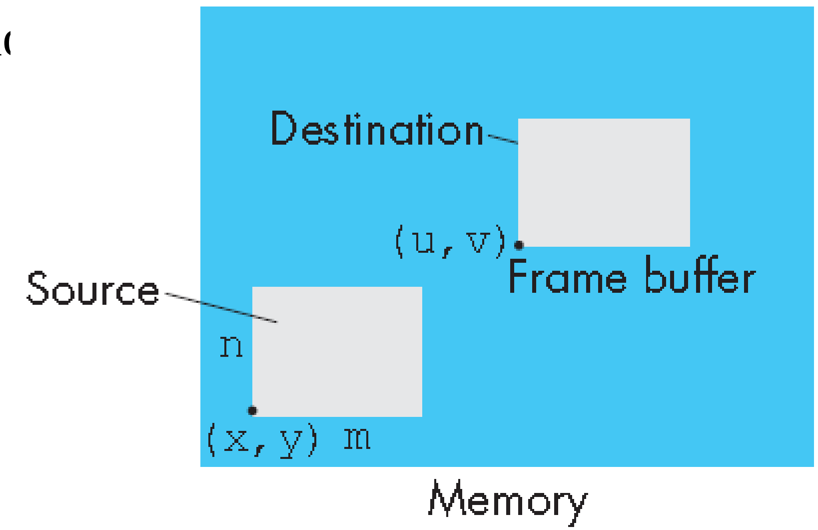
Writing in Buffers

Write an nxm source block with

Lower-left corner of destination block

```
write_block(source, n, m, x, y, destination, u, v);
```

Lower-left corner
of source bl

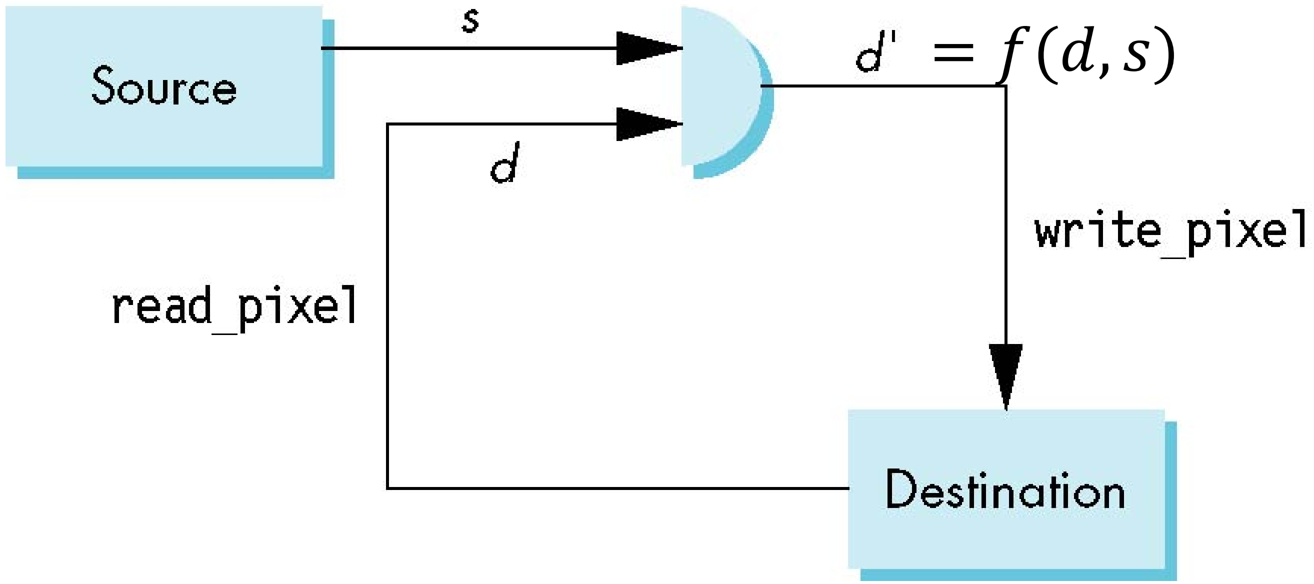


Writing Model

s: source bit

d: destination bit

Read destination pixel before writing source



Bit Writing Modes

Source and destination bits are combined bitwise

16 possible functions (one per column in table)

0 and 15: clear mode; 3 and 7: write mode

replace XOR OR

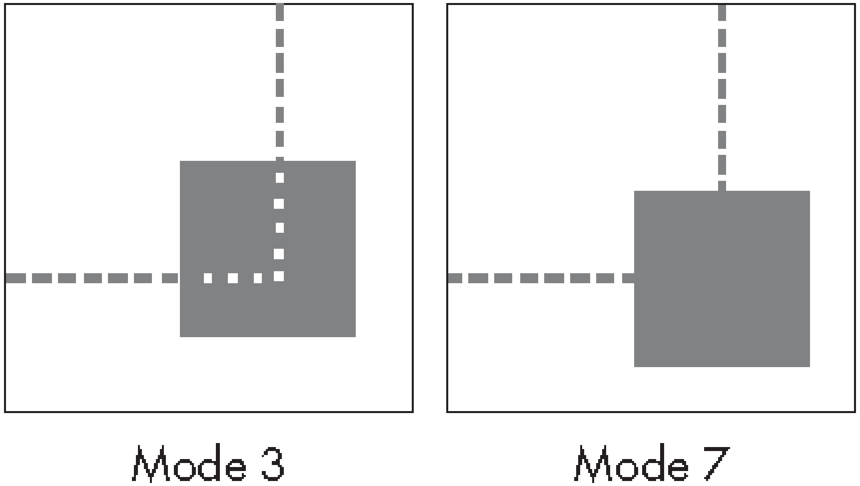
s	*d*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Bit Writing Modes

Background color: white

Foreground color: black

replace OR



XOR (Exclusive OR) Mode

Property of XOR: return the original value if apply XOR twice

$$d = (d \oplus s) \oplus s$$

XOR is especially useful for swapping blocks of memory such as menus that are stored off screen (***backing store***)

If S represents screen and M represents a menu, the sequence

$S \leftarrow S \oplus M$

$M \leftarrow S \oplus M$

$S \leftarrow S \oplus M$

swaps S and M

For example, S=1010, M=1100

S=S $\oplus$ M=0110

M=S $\oplus$ M=1010

S=S $\oplus$ M=1100

Buffer Selection

OpenGL can read from any of the buffers (front, back, depth)

Default to the back buffer

Change with `glReadBuffer`

Drawing through texture functions

Limits of Geometric Modeling

Although graphics cards can render over 10 million polygons per second, that number is insufficient for many phenomena

- Clouds
- Grass
- Terrain
- Skin

Mapping

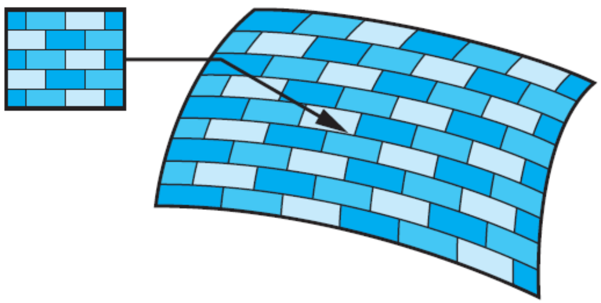
Modify color in fragment processing after rasterization

Three Major Mapping Methods

- **Texture Mapping**
  - Uses images to fill inside of polygons
- **Environment (reflection mapping)**
  - Uses a picture of the environment for texture maps of reflection surface
  - Allows simulation of highly specular surfaces
- **Bump mapping**
  - Emulates altering normal vectors during the rendering process

Examples of Mapping

Texture mapping



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Reflection mapping

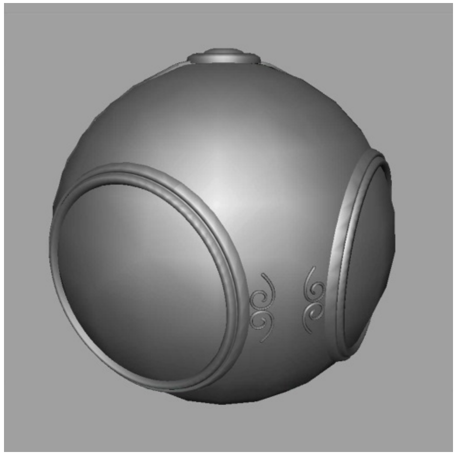
https://en.wikipedia.org/wiki/Reflection_mapping

Bump mapping

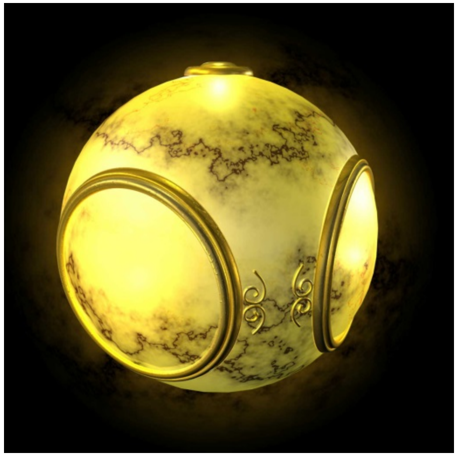
http://memim.com/bumpmapping.html

Examples of Mapping

Geometric model

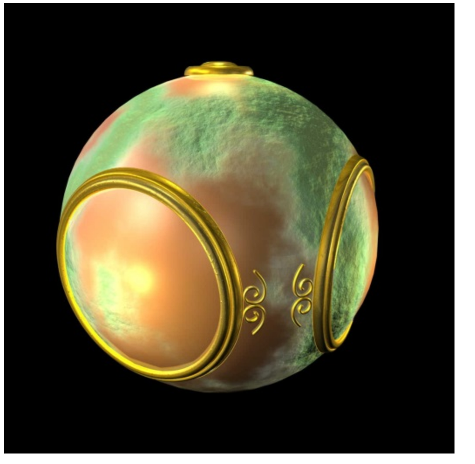


Texture mapping



Reflection mapping

Bump mapping



Modeling an Orange

Consider the problem of modeling an orange (the fruit)

Start with an orange-colored sphere

- Too simple

Replace sphere with a more complex shape

- Does not capture surface characteristics (small dimples)
- Takes too many polygons to model all the dimples

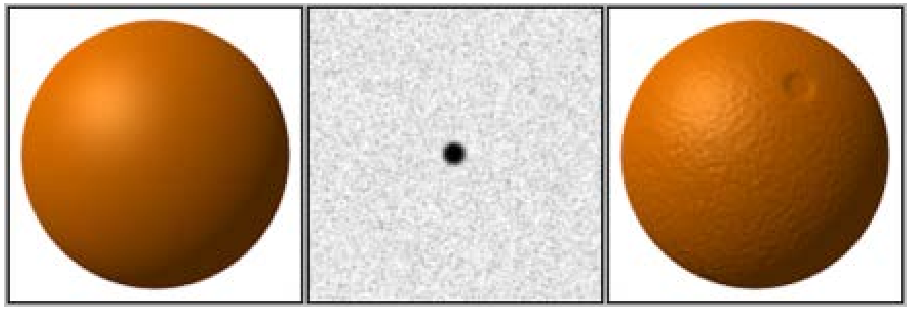
Modeling an Orange (2)

Take a picture of a real orange, scan it, and "paste" onto simple geometric model

- This process is known as texture mapping

Still might not be sufficient because resulting surface will be smooth

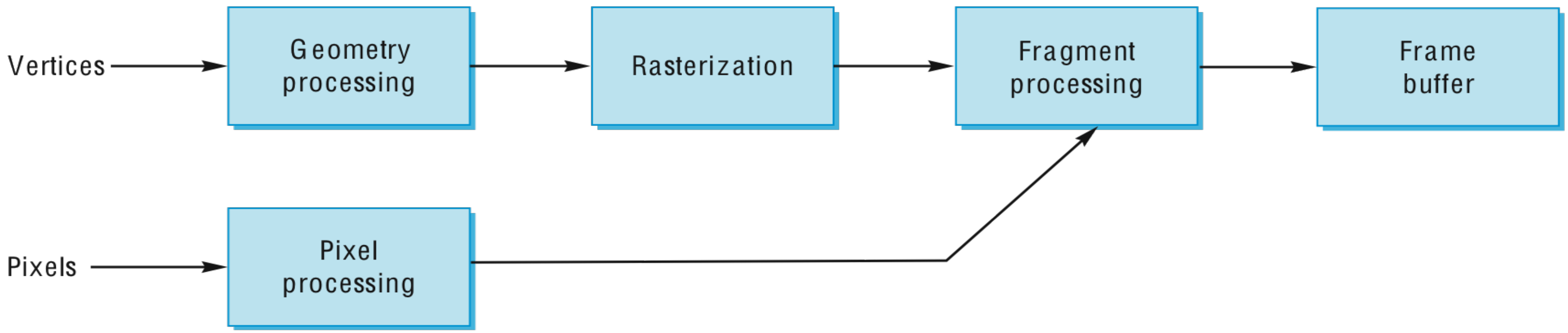
- Need to change local shape
- Bump mapping



Where does mapping take place?

Mapping techniques are implemented at the end of the rendering pipeline

- Very efficient because few polygons make it past the clipper



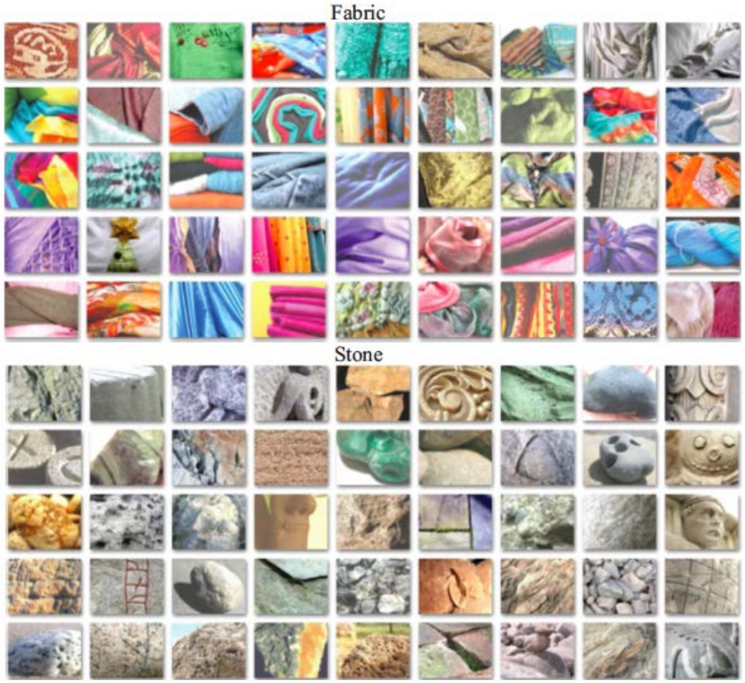
Texture Mapping

Textures are patterns including

- Regular patterns, e.g.,
  - Stripes, checkerboards
- Complex patterns
  - Natural materials

Textures can be

- 1D – coloring a curve
- 2D – coloring surfaces
- 3D – coloring a solid block
- 4D – space-time texture



Forsyth and Ponce, “Computer Vision – A Modern Approach 2e”
Images are from Flickr Material Database,
https://people.csail.mit.edu/lavanya/fmd.html

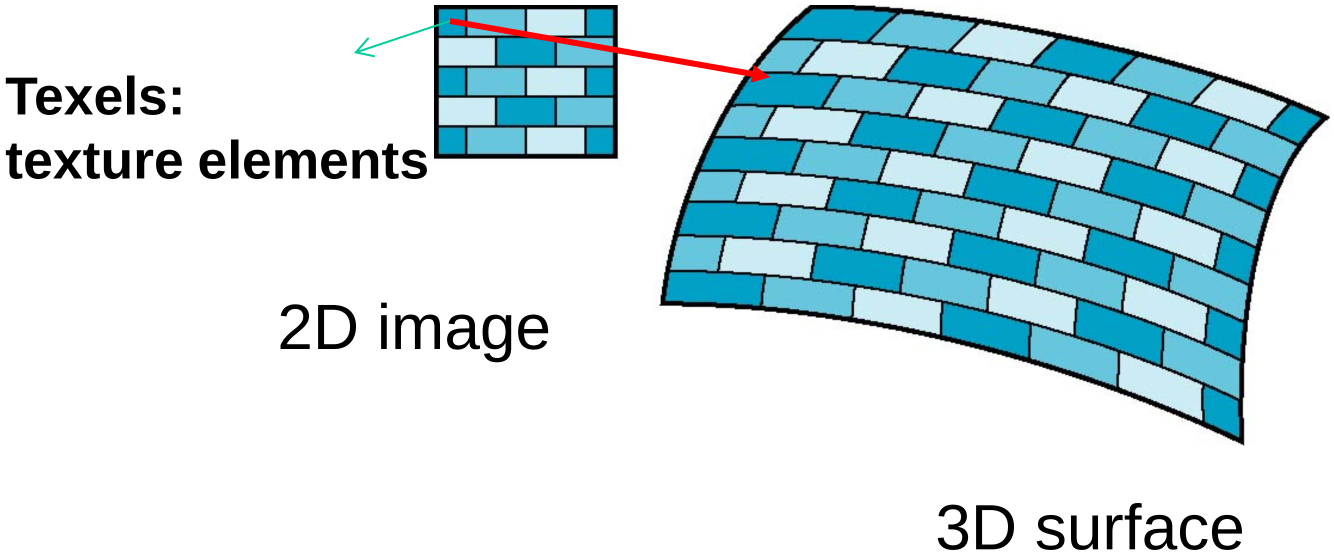
Texture Mapping

Textures are stored in images - 2D arrays.

Each element is called a ***texel***

The idea is simple---map an image to a surface or map every texel to a point on a geometric object

However, there are 3 or 4 coordinate systems involved



Coordinate Systems

Parametric coordinates

- May be used to model curves and curved surfaces

Texture coordinates

- Used to identify points in the image to be mapped

Object or World Coordinates

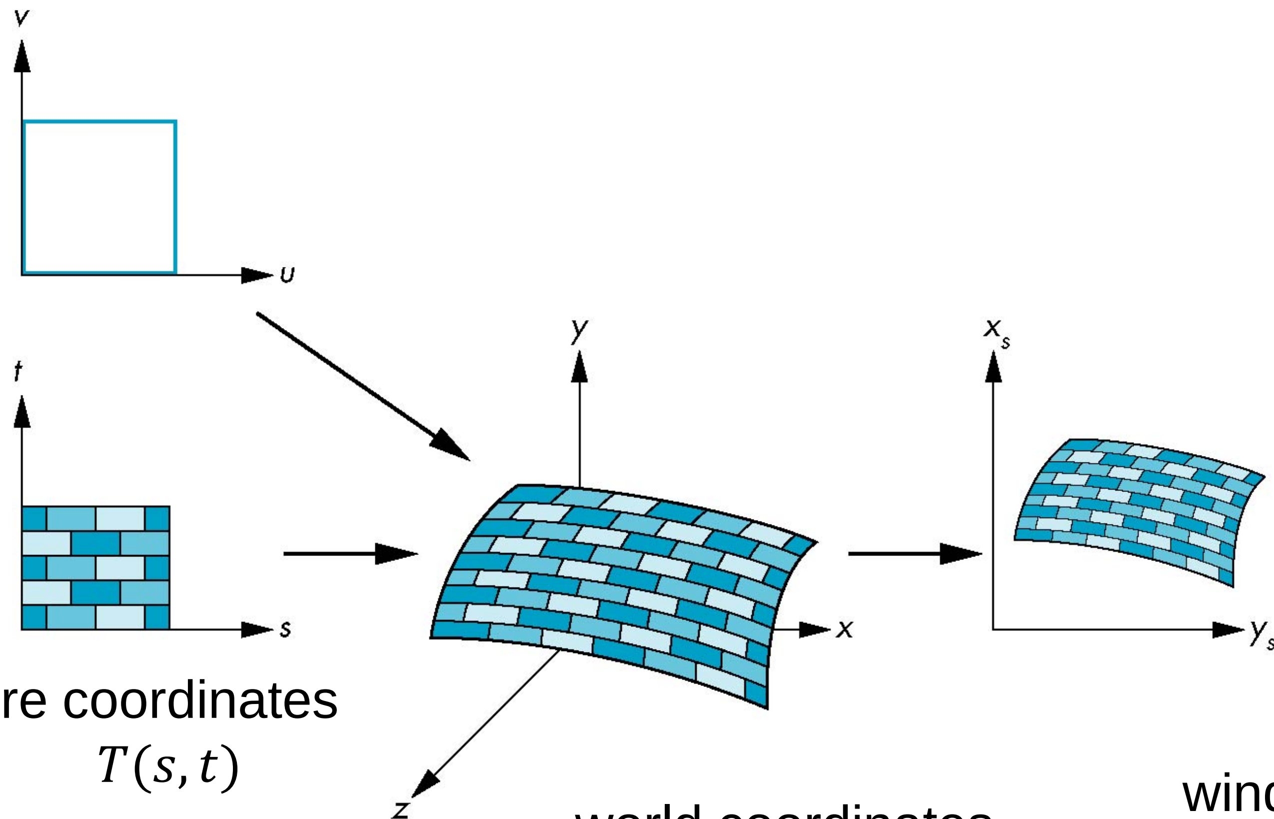
- Conceptually, where the mapping takes place

Window/screen Coordinates

- Where the final image is really produced

Texture Mapping

parametric coordinates



texture coordinates

$T(s,t)$

world coordinates

window coordinates

Mapping Functions

Intuitively, consider mapping from texture coordinates to a point on a surface – forward mapping

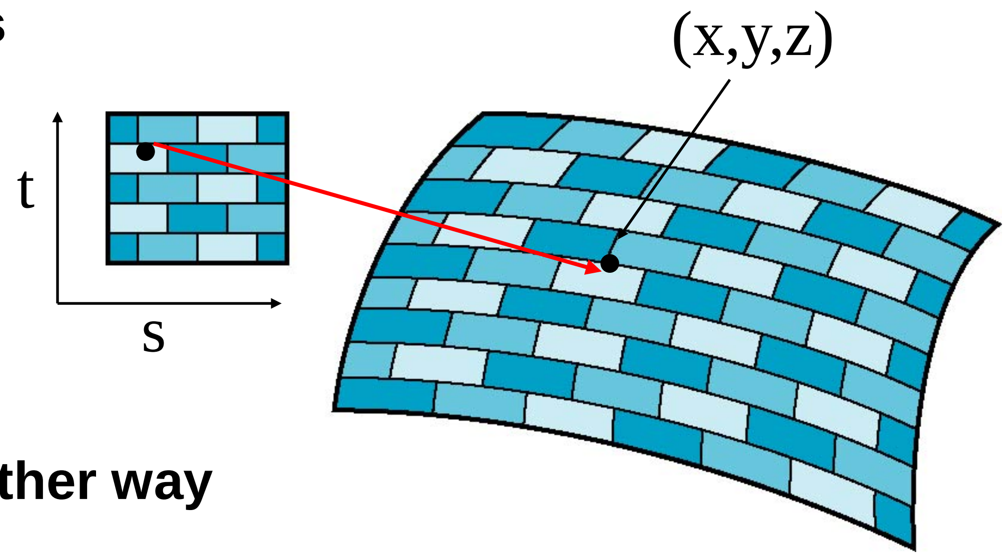
Appear to need four functions

$x = x(s,t)$

$y = y(s,t)$

$z = z(s,t)$

$w = w(s,t)$



But we really want to go the other way

Why?

One severe problem- may result in holes

Backward Mapping

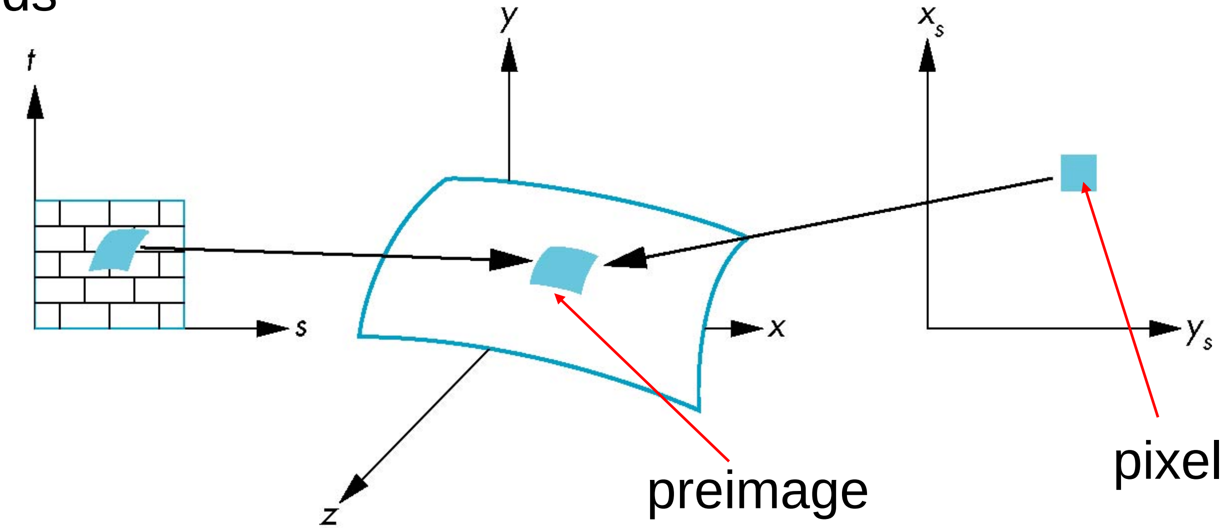
We really want to go backwards

- Given a pixel, we want to know to which point on an object it corresponds, the preimage (inverse) of a pixel
- Given a point on an object, we want to know to which point in the texture it corresponds

Backward mapping

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

$$t = t(x,y,z,w)$$

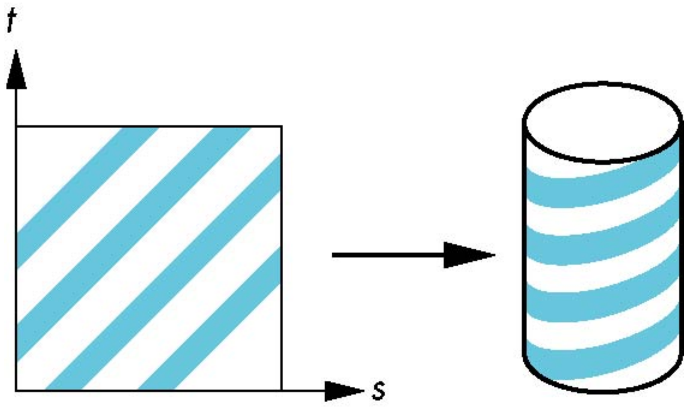


Such functions are difficult to find in general

Two-part mapping

One solution to the mapping problem is to first map the texture to a simple intermediate surface, e.g., a sphere, cylinder, or cube

Example: map to cylinder and then map to the target surface



First Mapping: (1) Cylindrical Mapping

Parametric function of a *cylinder*

$$x = r\cos(2\pi u)$$
$$y = r\sin(2\pi u)$$
$$z = v/h$$

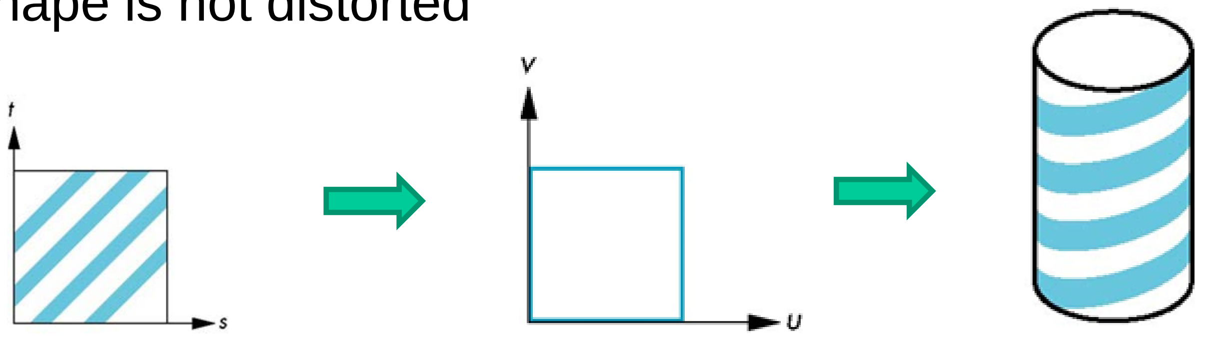
maps rectangle in u,v space to the curved part of a cylinder of radius r and height h in world coordinates

Parametric function of texture map

$$s = u$$
$$t = v$$

maps to texture space

The shape is not distorted

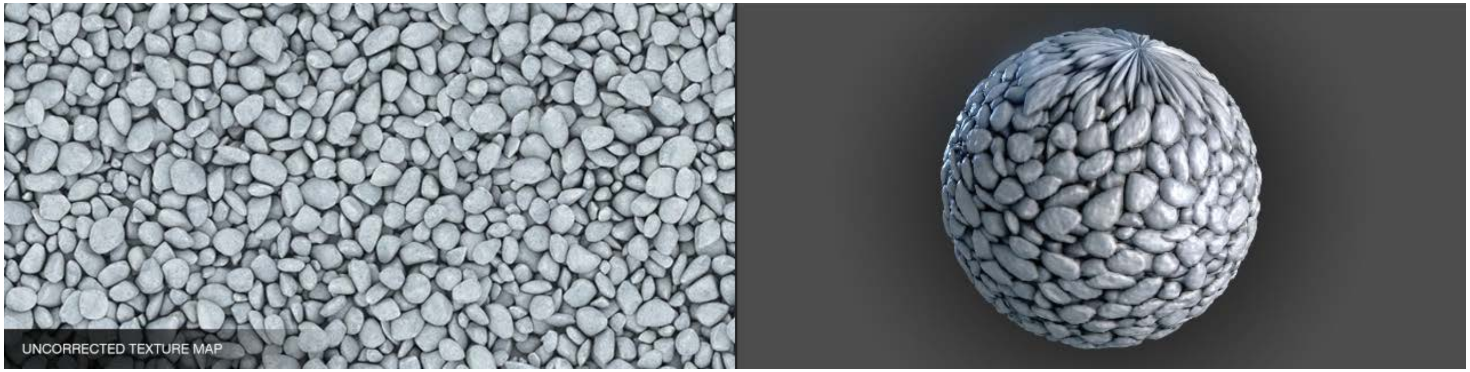


First Mapping: (2) Spherical Map

We can use a parametric sphere

$$x = r\cos 2\pi u$$
$$y = r\sin 2\pi u \cos 2\pi v$$
$$z = r\sin 2\pi u \sin 2\pi v$$

in a similar manner to the cylinder but have to decide where to put the distortion, e.g., at the poles



http://richardrosenman.com/shop/spherical-mapping-corrector/

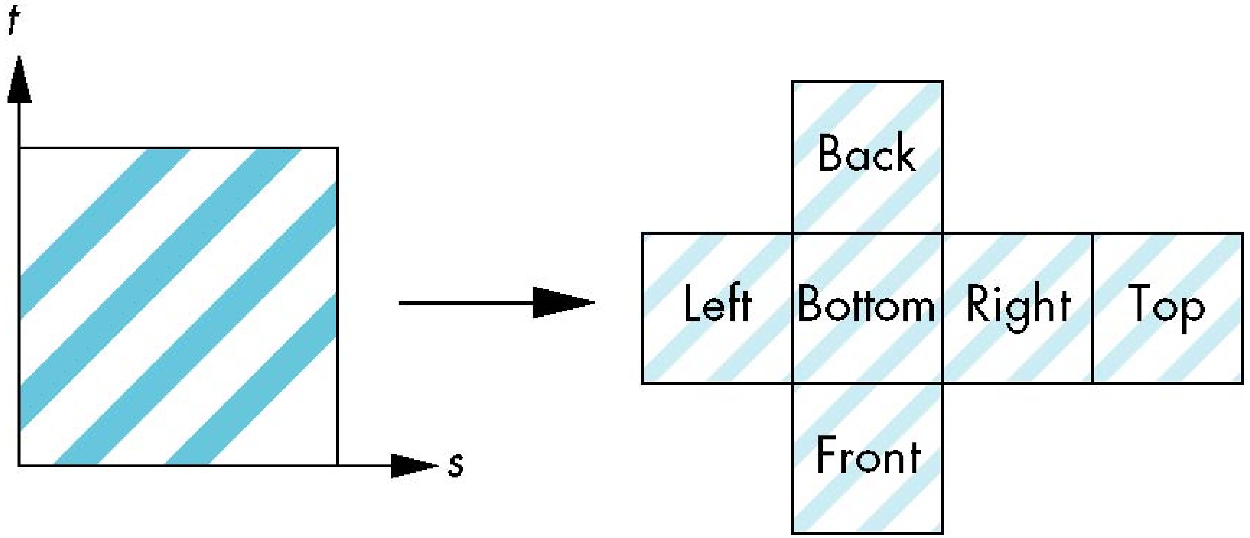
Spheres are often used in environmental maps

First Mapping: (3) Box Mapping

Map the texture to a unraveled box

Easy to use with simple orthographic projection

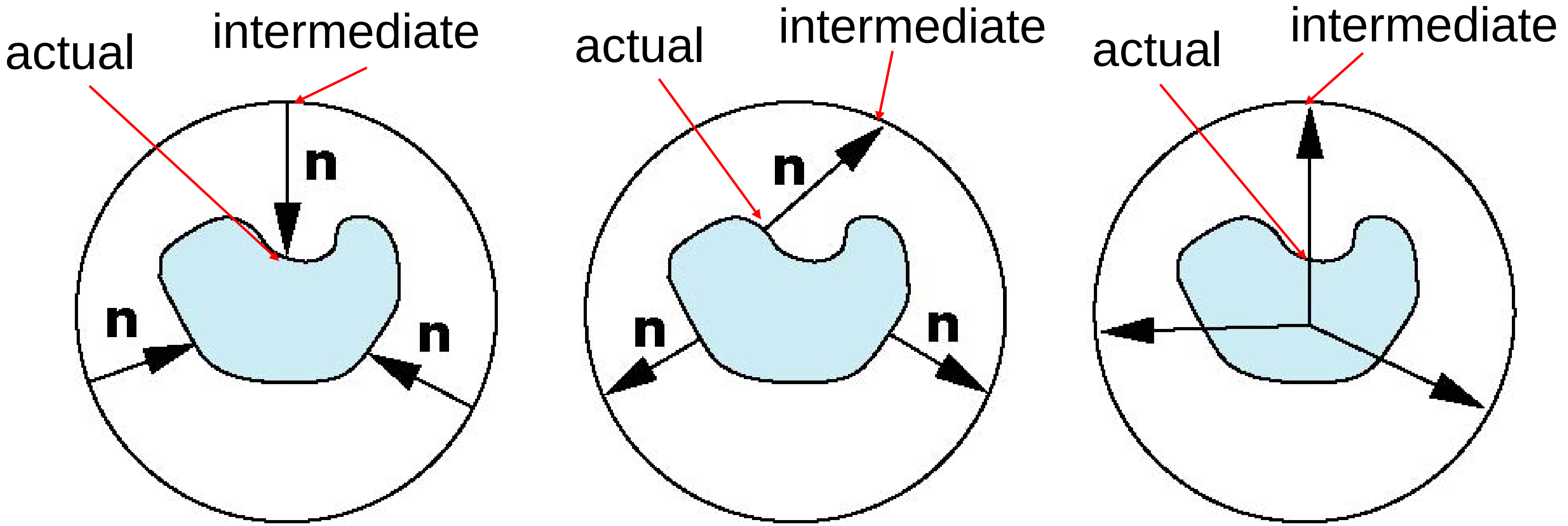
Also used in environment maps



Second Mapping: From Intermediate Object to Actual Object

Three strategies:

- from intermediate, along normal of intermediate until intersect with the object
- from object, along normal of the object until intersect with the intermediate
- Vectors from center of intermediate/object, intersect the object and the intermediate



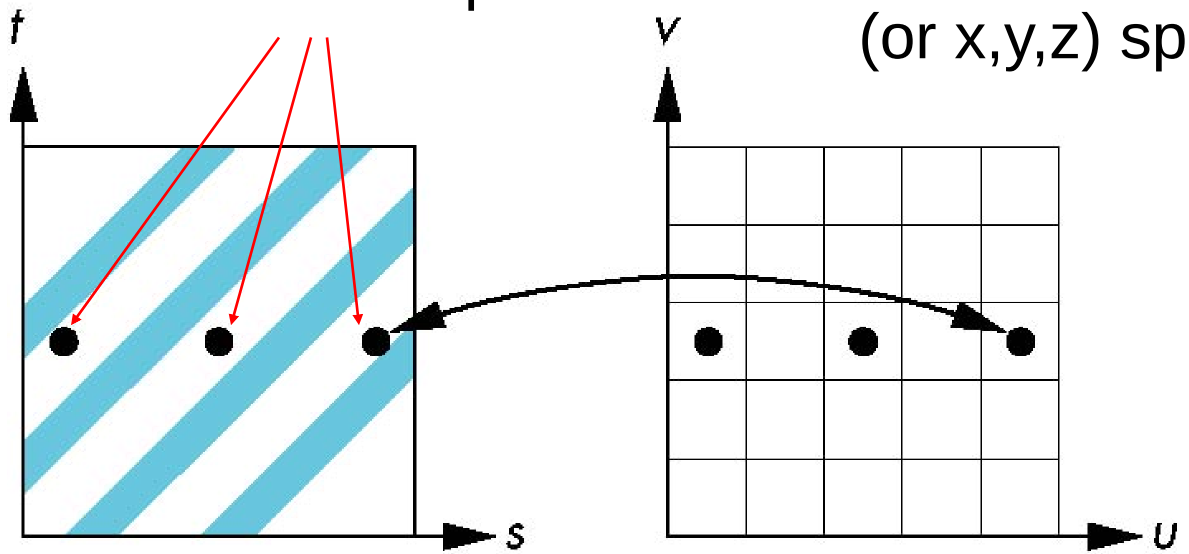
Aliasing

Backward mapping for the centers of pixels

Point sampling of the texture/object can lead to aliasing errors

miss blue stripes

point samples in u,v
(or x,y,z) space



Area Averaging

A better but slower option is to use ***area averaging*** of the texture map over the preimage

Cannot handle high-frequency components, e.g., the stripe pattern – sampling at higher frequencies