CSE 167:
Introduction to Computer Graphics
Lecture #8: Textures

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2017

Announcements

UCSD

Lecture Overview

▸ Texture Mapping

  ▸ Overview

  ▸ Wrapping
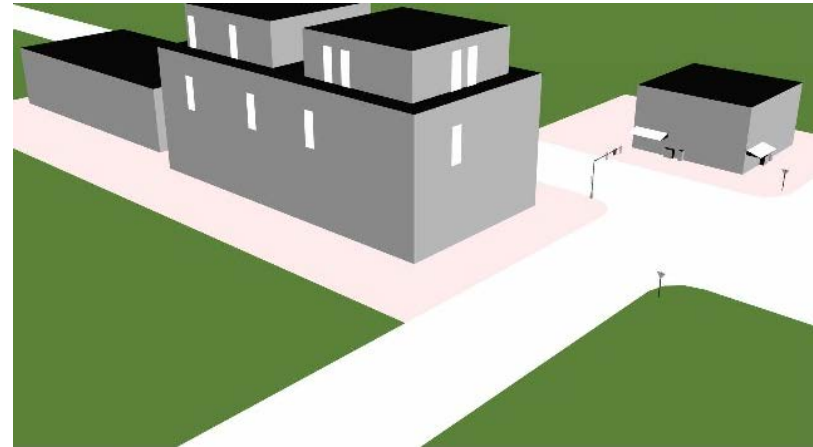
  ▸ Texture coordinates

  ▸ Anti-aliasing

UCSD

Large Triangles

Pros:

- Often sufficient for simple geometry
- Fast to render

Cons:

- Per vertex colors look boring and computer-generated

UCSD

Texture Mapping

- Map textures (images) onto surface polygons
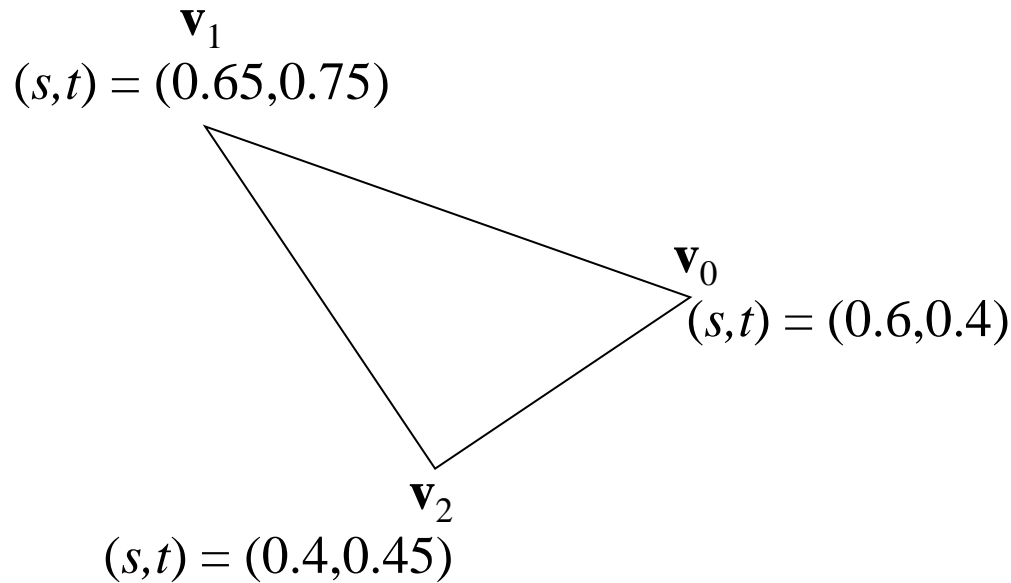- Same triangle count, much more realistic appearance

UCSD

Texture Mapping

- Goal: map locations in texture to locations on 3D geometry

- Texture coordinate space

  - Texture pixels (texels) have texture coordinates (s,t)

- Convention

  - Bottom left corner of texture is at $(s,t) = (0,0)$
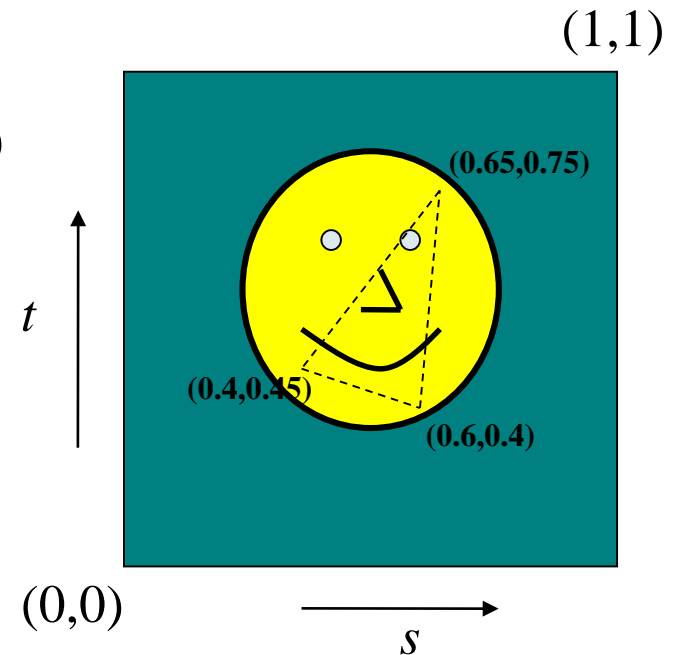
  - Top right corner is at $(s,t) = (1,1)$

$(1,1)$

t

$(0,0)$

s

Texture coordinates
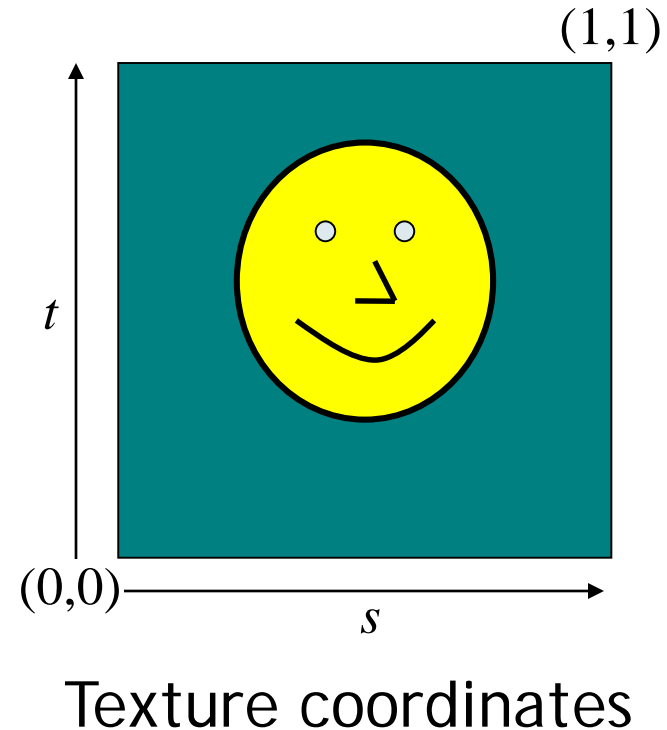
Texture Mapping

▸ Store 2D texture coordinates s,t with each triangle vertex

$\mathbf{v}_1$
$(s,t) = (0.65, 0.75)$

$\mathbf{v}_0$
$(s,t) = (0.6, 0.4)$

$\mathbf{v}_2$
$(s,t) = (0.4, 0.45)$

Triangle in any space before projection

(1,1)

(0.65,0.75)

t

(0.4,0.45)

(0.6,0.4)

(0,0)

s

Texture coordinates

UCSD

Texture Mapping

▶ Each point on triangle gets color from its corresponding point in texture

$\mathbf{v}_1$
$(s,t) = (0.65, 0.75)$

$\mathbf{v}_0$
$(s,t) = (0.6, 0.4)$

$\mathbf{v}_2$
$(s,t) = (0.4, 0.45)$

Triangle in any space before projection

$(1,1)$

(0.65,0.75)

t

(0.4,0.45)

(0.6,0.4)

$(0,0)$

s

Texture coordinates

UCSD

Texture Mapping

Primitives

Modeling and viewing transformation

Shading

Projection

Rasterization
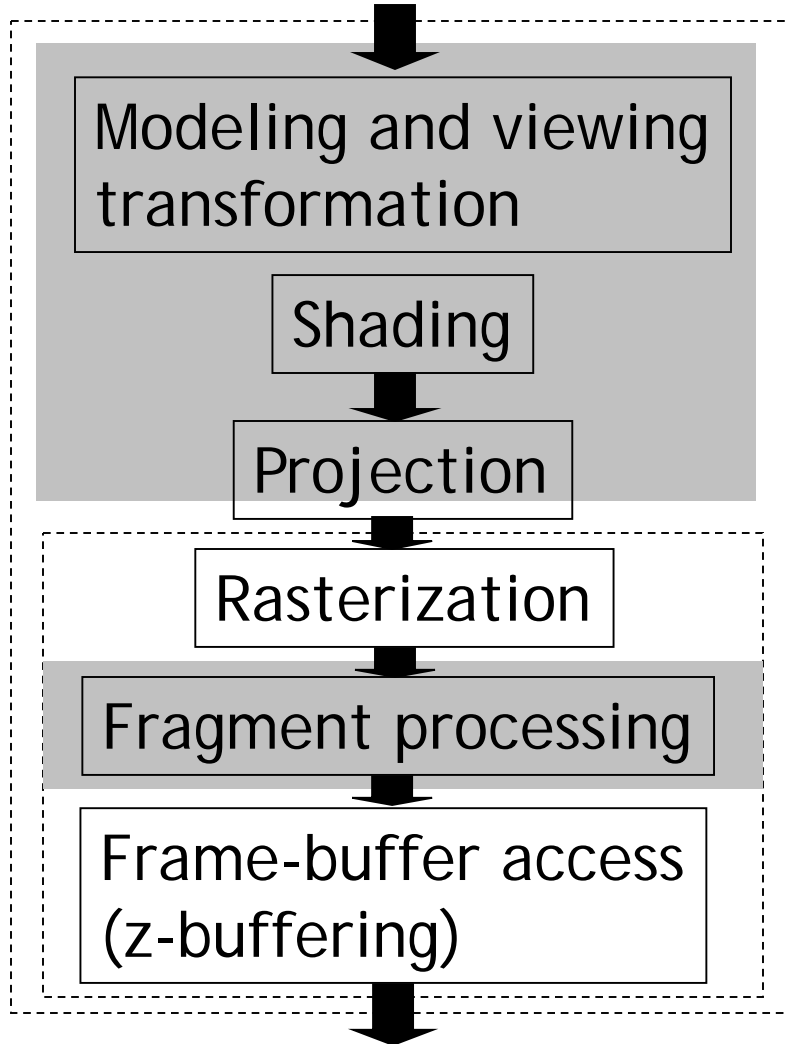
Fragment processing → Includes texture mapping

Frame-buffer access (z-buffering)

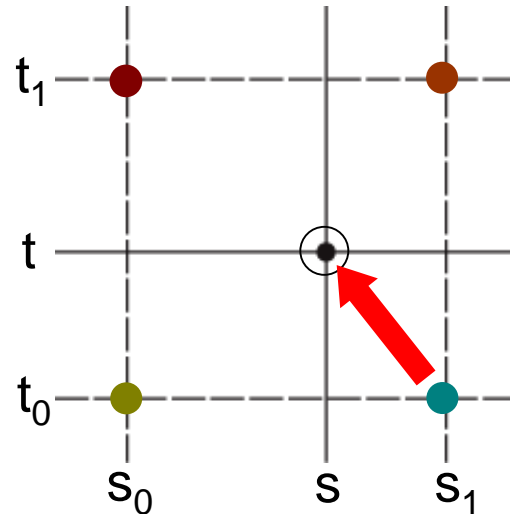Image

UCSD

Texture Look-Up

▸ Given interpolated texture coordinates (s, t) at current pixel

▸ Closest four texels in texture space are at

$(s_0, t_0), (s_1, t_0), (s_0, t_1), (s_1, t_1)$

▸ How to compute pixel color?
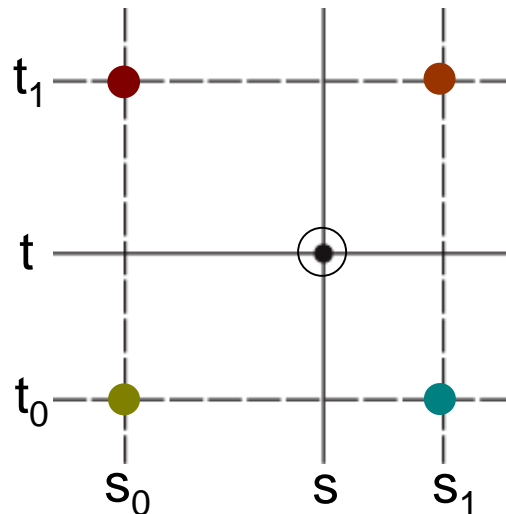
Nearest-Neighbor Interpolation

‣ Use color of closest texel

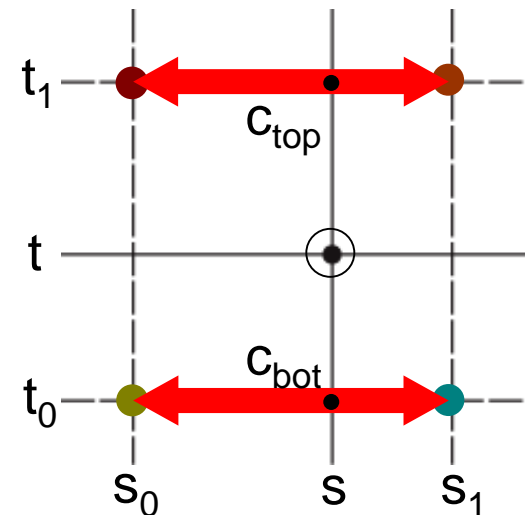‣ Simple, but low quality and aliasing

UCSD

Bilinear Interpolation

1. Linear interpolation horizontally:

   Ratio in s direction r_s:

   $$r_s = \frac{s - s_0}{s_1 - s_0}$$

   c_{top} = tex(s_0,t_1) ($1-r_s$) + tex(s_1,t_1) r_s

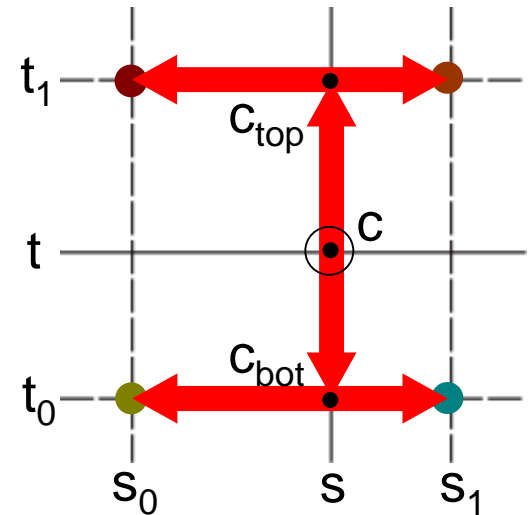 c_{bot} = tex(s_0, t_0) ($1-r_s$) + tex(s_1,t_0) r_s

Bilinear Interpolation

2. Linear interpolation vertically

Ratio in t direction r_t:

$$r_t = \frac{t - t_0}{t_1 - t_0}$$

$c = c_{bot}\,(1-r_t) + c_{top}\,r_t$

UCSD

Texture Filtering in OpenGL

▸ GL_NEAREST: Nearest-Neighbor interpolation

▸ GL_LINEAR: Bilinear interpolation

▸ Example:

  ▸ glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

  ▸ glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

GL_NEAREST   GL_LINEAR

*Source: https://open.gl/textures*

Lecture Overview

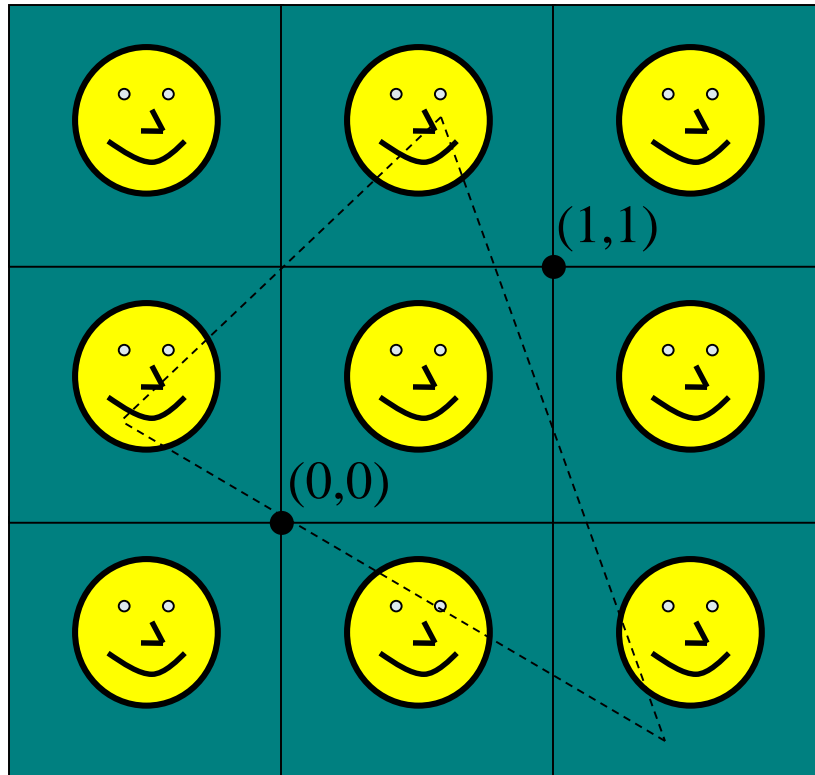- Texture Mapping
  - Wrapping
  - Texture coordinates
  - Anti-aliasing

UCSD

Wrap Modes

▸ **Texture image extends from $[0,0]$ to $[1,1]$ in texture space**

 ▸ What if (s,t) texture coordinates are beyond that range?

▸ **→ Texture wrap modes**

UCSD

Repeat

▸ Repeat the texture

  ▸ Creates discontinuities at edges

    ▸ unless texture is designed to line up

Texture Space

t

s

$(1,1)$

$(0,0)$
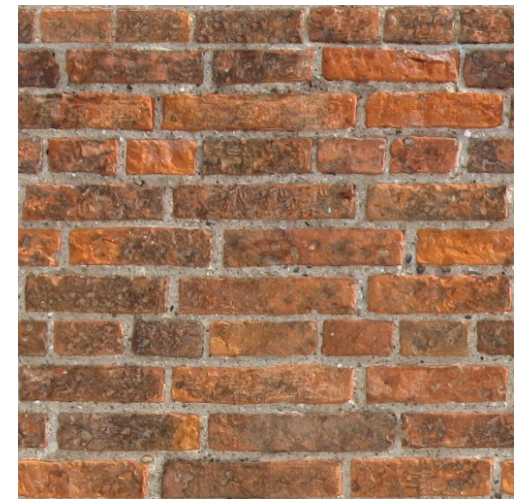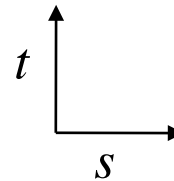
Seamless brick wall texture
(by Christopher Revoir)

Clamp

▸ Use edge value everywhere outside data range [0..1]
▸ Or use specified border color outside of range [0..1]

Texture Space
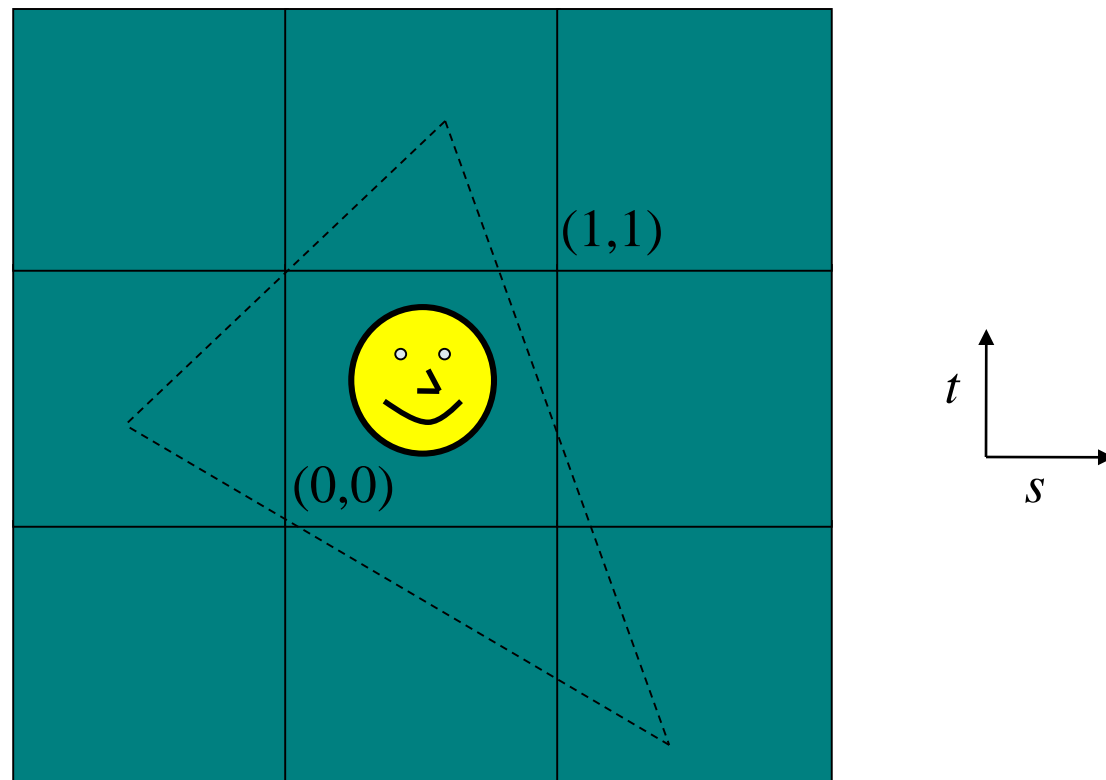
Wrap Modes in OpenGL

▸ Default:

  ▸ glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);

  ▸ glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

▸ Options for wrap mode:

  ▸ GL_REPEAT

  ▸ GL_MIRRORED_REPEAT

  ▸ GL_CLAMP_TO_EDGE: repeats last pixel in the texture

  ▸ GL_CLAMP_TO_BORDER: requires border color to be set

| GL_REPEAT | GL_MIRRORED_REPEAT | GL_CLAMP_TO_EDGE | GL_CLAMP_TO_BORDER |

Source: https://open.gl/textures

Lecture Overview

- Texture Mapping
  - Wrapping
  - Texture coordinates
  - Anti-aliasing

UCSD

Texture Coordinates

**What if texture extends across multiple polygons?**

→ Surface parameterization

▸ Mapping between 3D positions on surface and 2D texture coordinates

  ▸ Defined by texture coordinates of triangle vertices

▸ Options for mapping:

  ▸ Parametric

  ▸ Orthographic

  ▸ Projective

  ▸ Spherical

  ▸ Cylindrical

  ▸ Skin

UCSD

Cylindrical Mapping

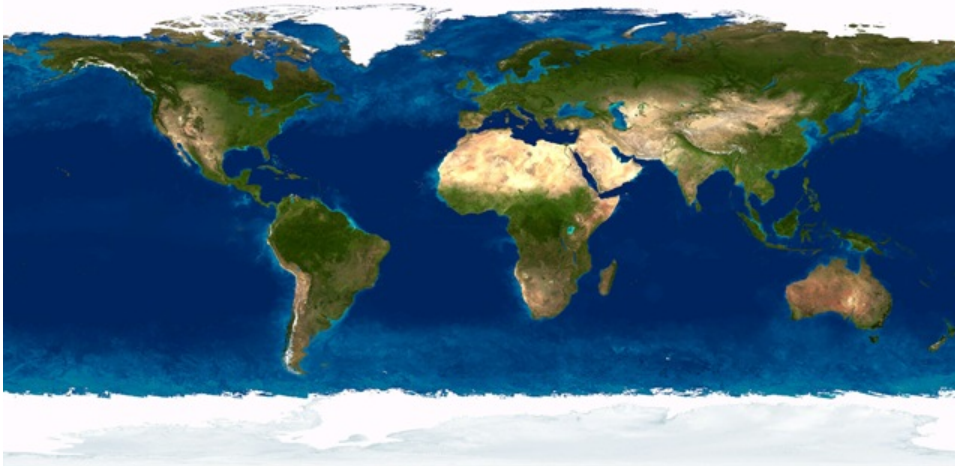▸ Similar to spherical mapping, but with cylindrical coordinates
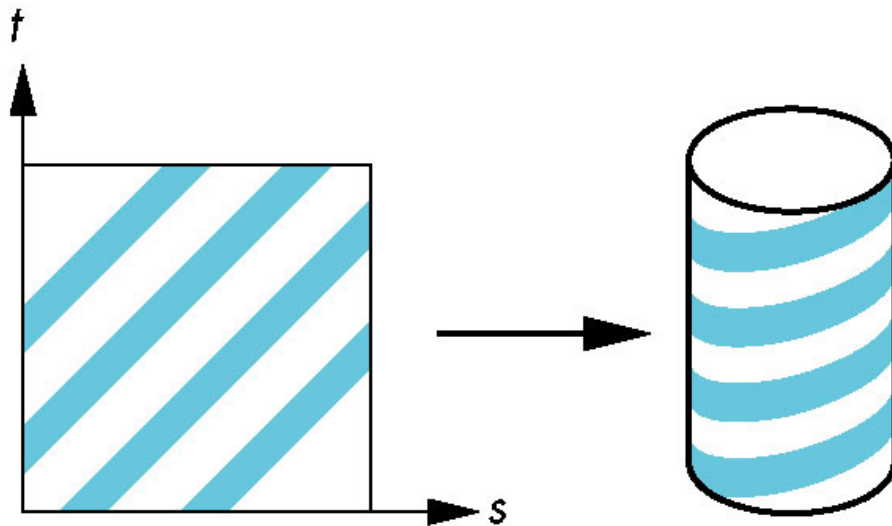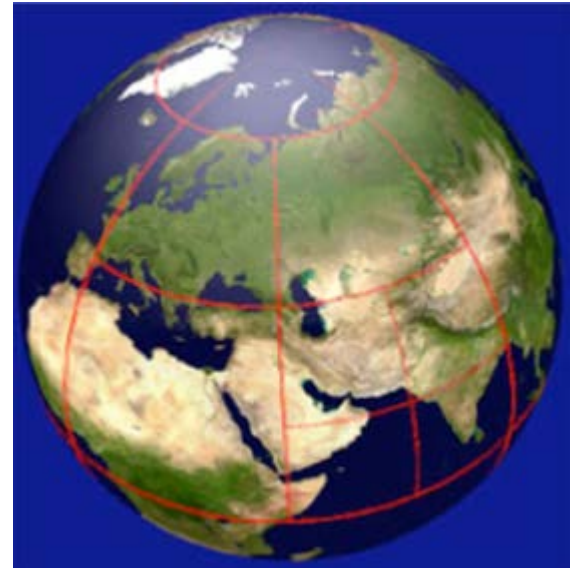

Cylinder Sides
Texture Coordinates

UCSD
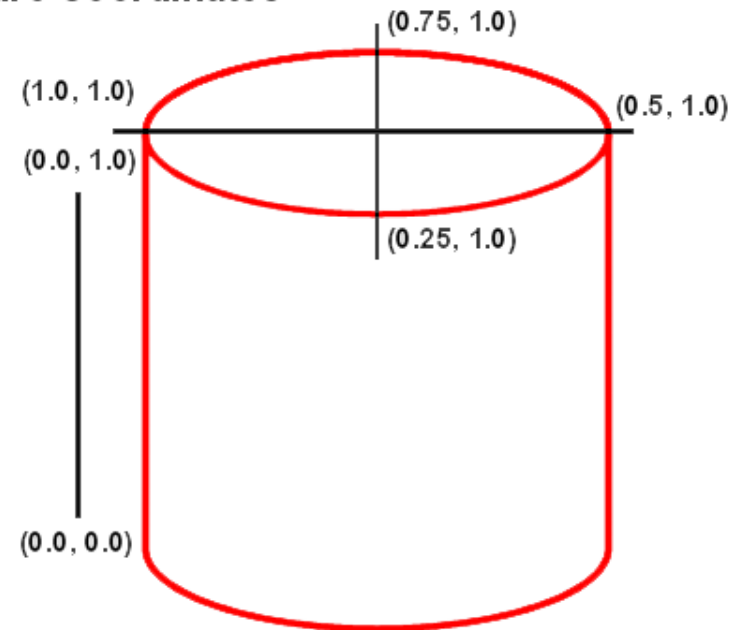
Spherical Mapping

▸ Use spherical coordinates

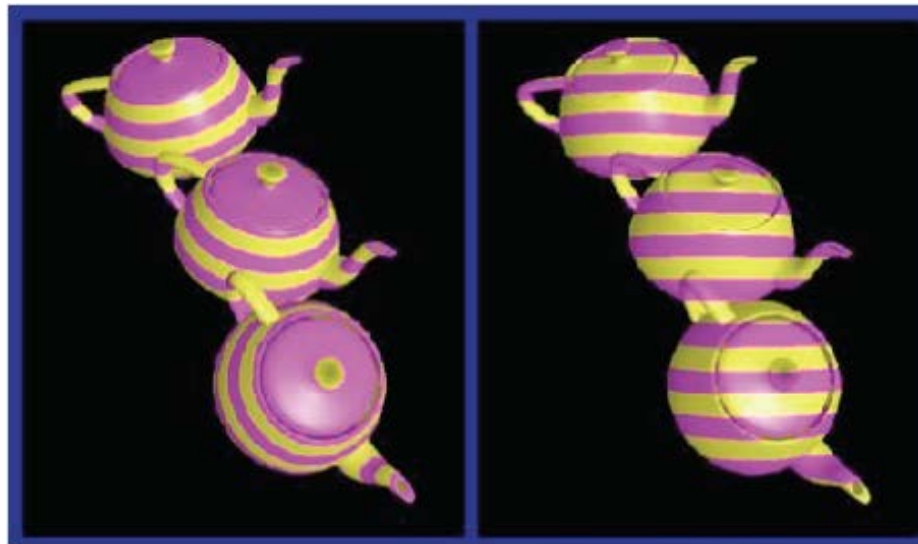▸ "Shrink-wrap" sphere to object

Texture map

Mapping result

UCSD

Orthographic Mapping

▸ Use linear transformation of object's xyz coordinates

▸ Example:

$$
\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}
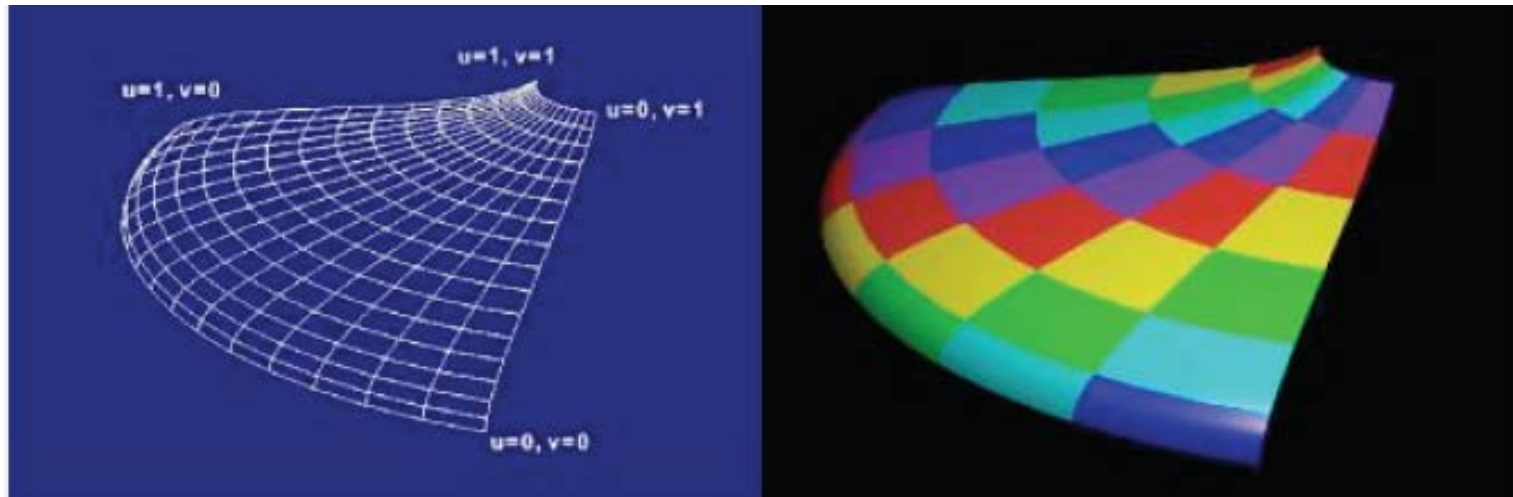$$

xyz in object space   xyz in camera space

UCSD

Parametric Mapping

▶ Surface given by parametric functions
$$x = f(u, v) \quad y = f(u, v) \quad z = f(u, v)$$

▶ Very common in CAD

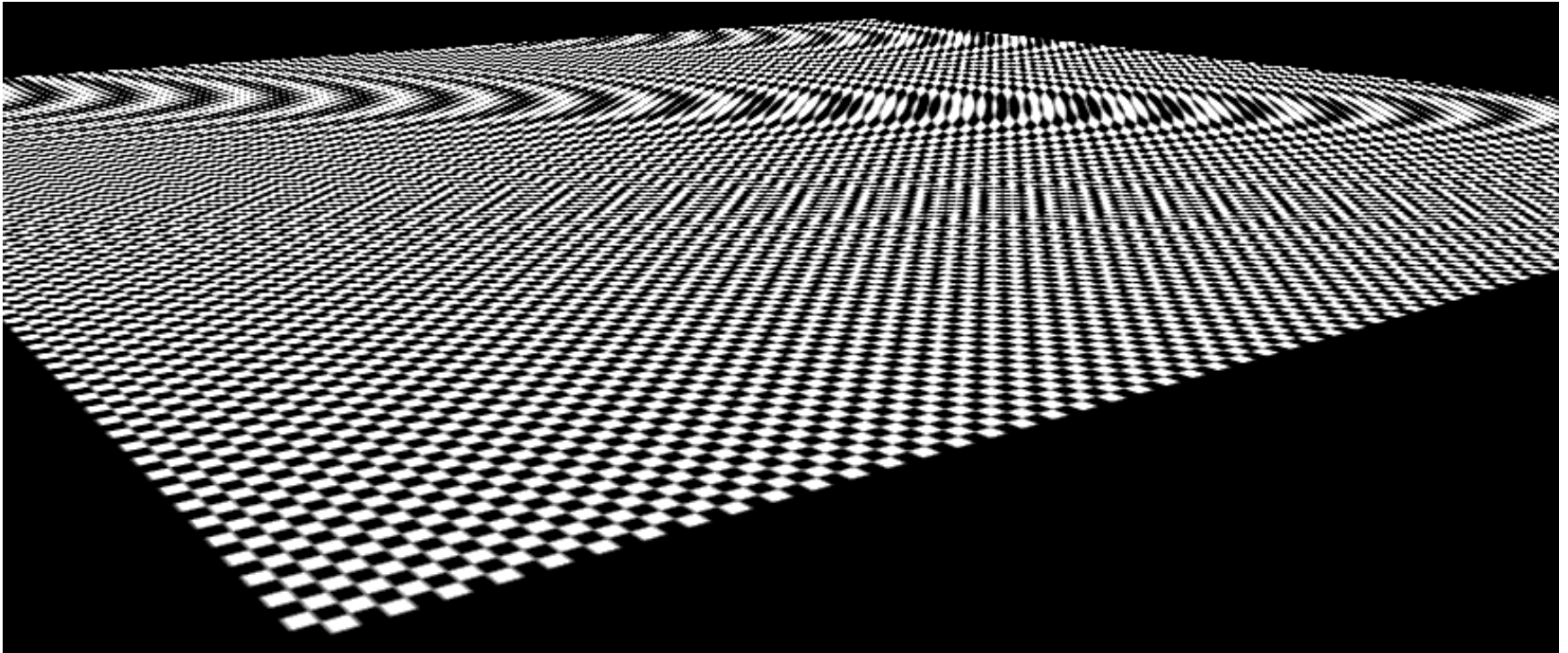▶ Clamp (u, v) parameters to [0..1] and use as texture coordinates (s, t)

UCSD

Lecture Overview

- Texture Mapping
  - Wrapping
  - Texture coordinates
  - Anti-aliasing

UCSD

Aliasing

▸ What could cause this aliasing effect?

UCSD

Aliasing

Sufficiently sampled, no aliasing

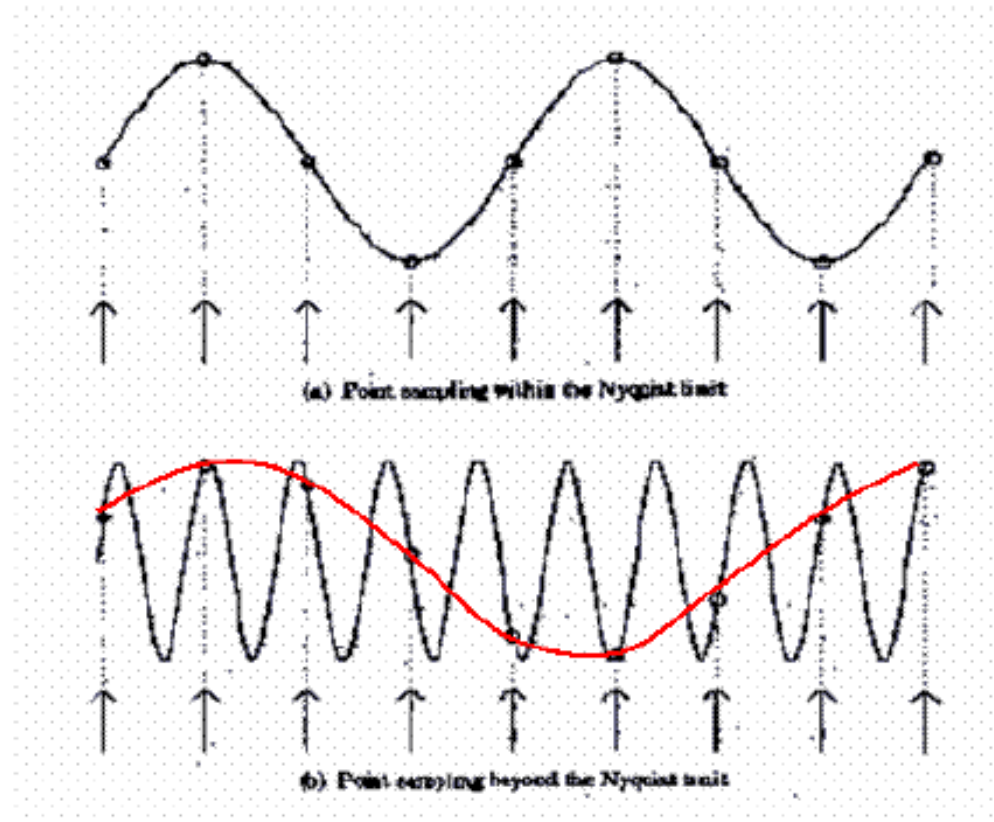**Insufficiently sampled, aliasing**

Image: Robert L. Cook

High frequencies in the input data can appear as lower frequencies in the sampled signal

UCSD

Antialiasing: Intuition

▸ Pixel may cover large area on triangle in camera space
▸ Corresponds to many texels in texture space
▸ Need to compute average

Image plane   Camera space   Texture space

Texels

Pixel area

y

x

v

u

UCSD
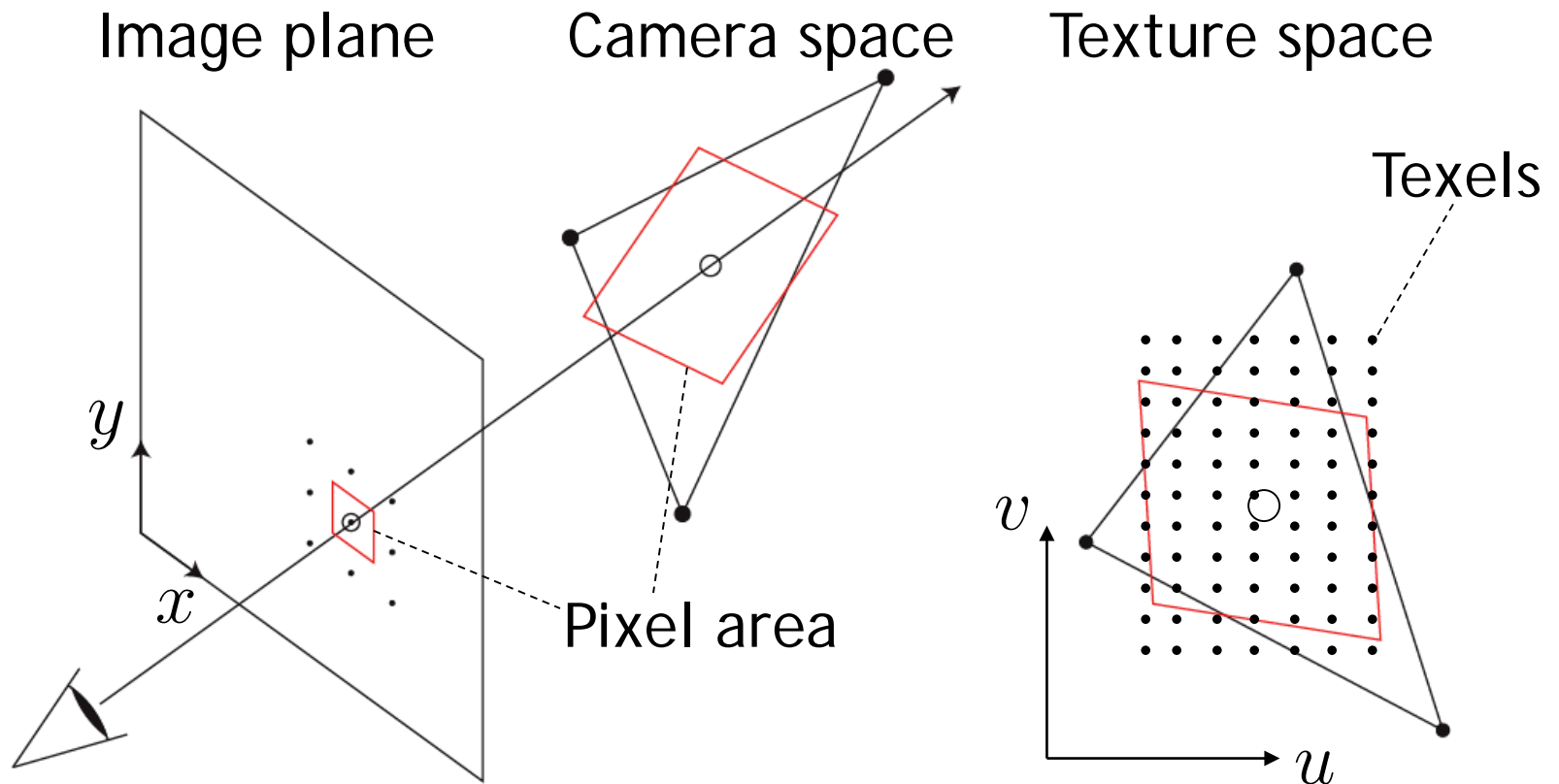
Lecture Overview

- Texture Mapping
  - Mip Mapping

UCSD

Antialiasing Using Mip-Maps

▸ **Averaging over texels is expensive**

  ▸ Many texels as objects get smaller

  ▸ Large memory access and compuation cost

▸ **Precompute filtered (averaged) textures**

  ▸ Mip-maps

▸ **Practical solution to aliasing problem**

  ▸ Fast and simple

  ▸ Available in OpenGL, implemented in GPUs

  ▸ Reasonable quality

UCSD

Mipmaps

- MIP stands for *multum in parvo = "much in little"* (Williams 1983)

Before rendering

- Pre-compute and store down scaled versions of textures
  - Reduce resolution by factors of two successively
  - Use high quality filtering (averaging) scheme
- Increases memory cost by 1/3
  - $1/3 = \frac{1}{4}+1/16+1/64+\ldots$
- Width and height of texture should be powers of two (non-power of two supported since OpenGL 2.0)

UCSD

Mipmaps

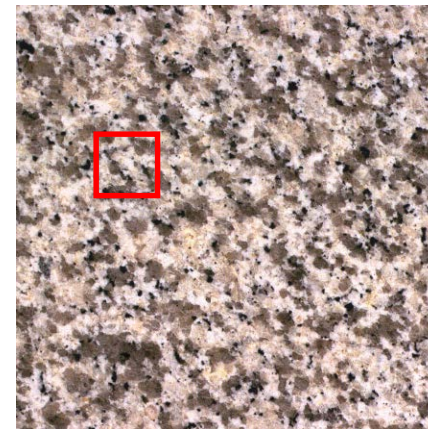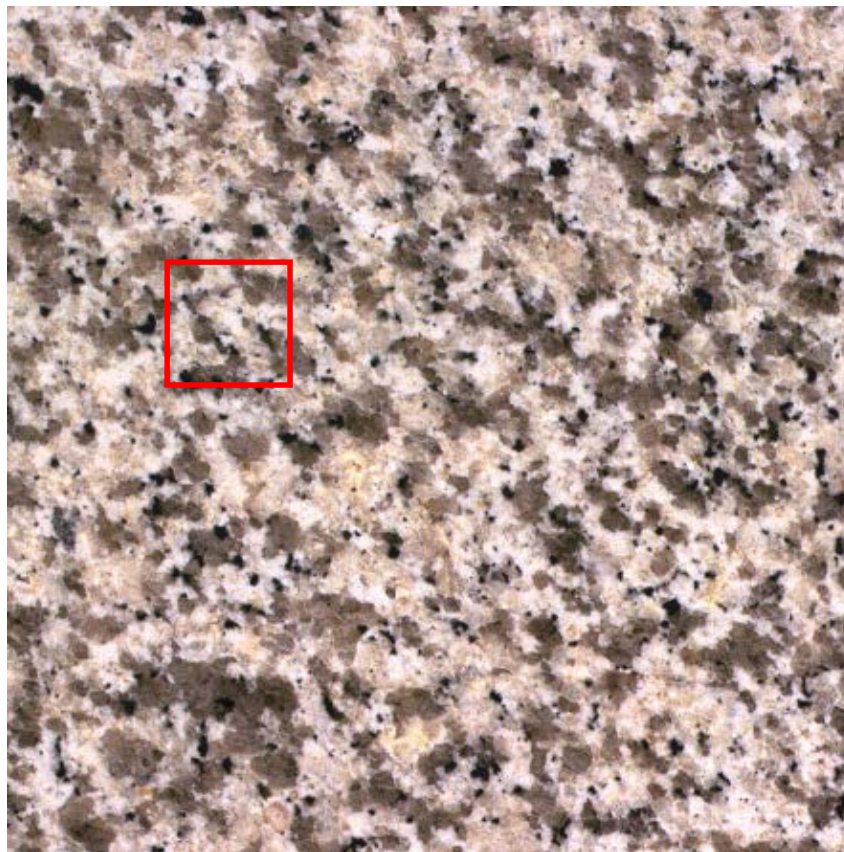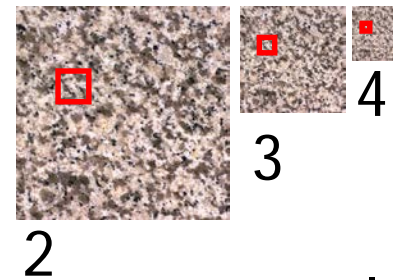- Example: resolutions 512x512, 256x256, 128x128, 64x64, 32x32 pixels

Level 1

4

3

2

"multum in parvo"

Level 0

UCSD
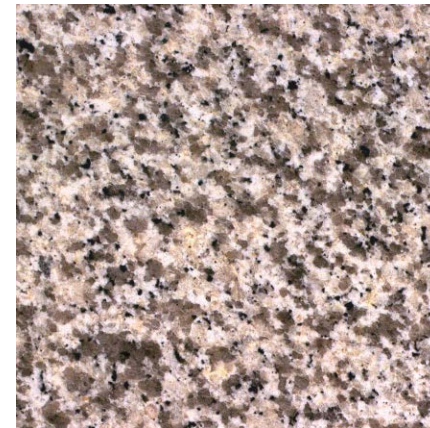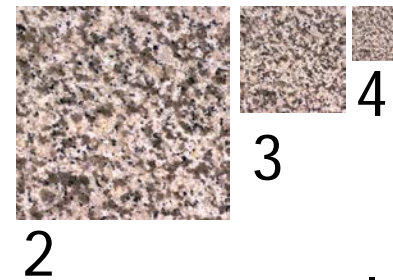
Mipmaps

▸ One texel in level 4 is the average of $4^4=256$ texels in level 0

Level 1

4

3

2

"multum in parvo"

Level 0

UCSD
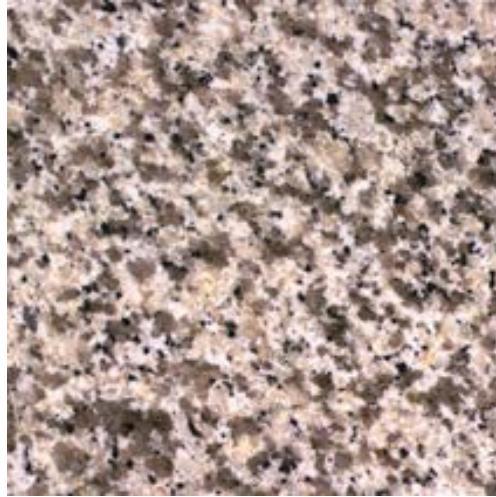
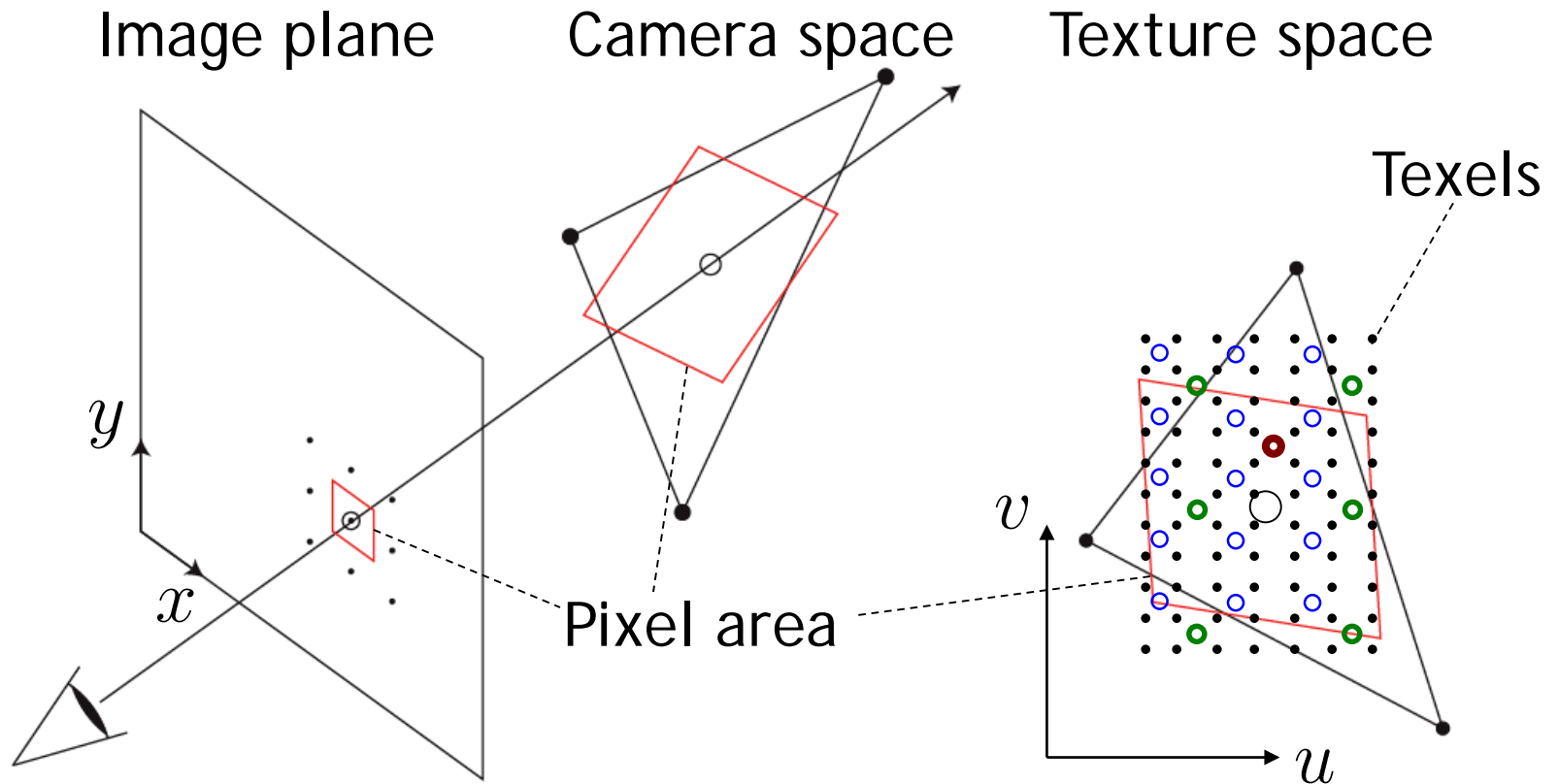# Mipmaps

Level 0

Level 1

Level 2

Level 3

Level 4

UCSD

Rendering With Mipmaps

- "Mipmapping"
- Interpolate texture coordinates of each pixel as without mipmapping
- Compute approximate size of pixel in texture space
- Look up color in nearest mipmap
  - E.g., if pixel corresponds to 10x10 texels use mipmap level 3
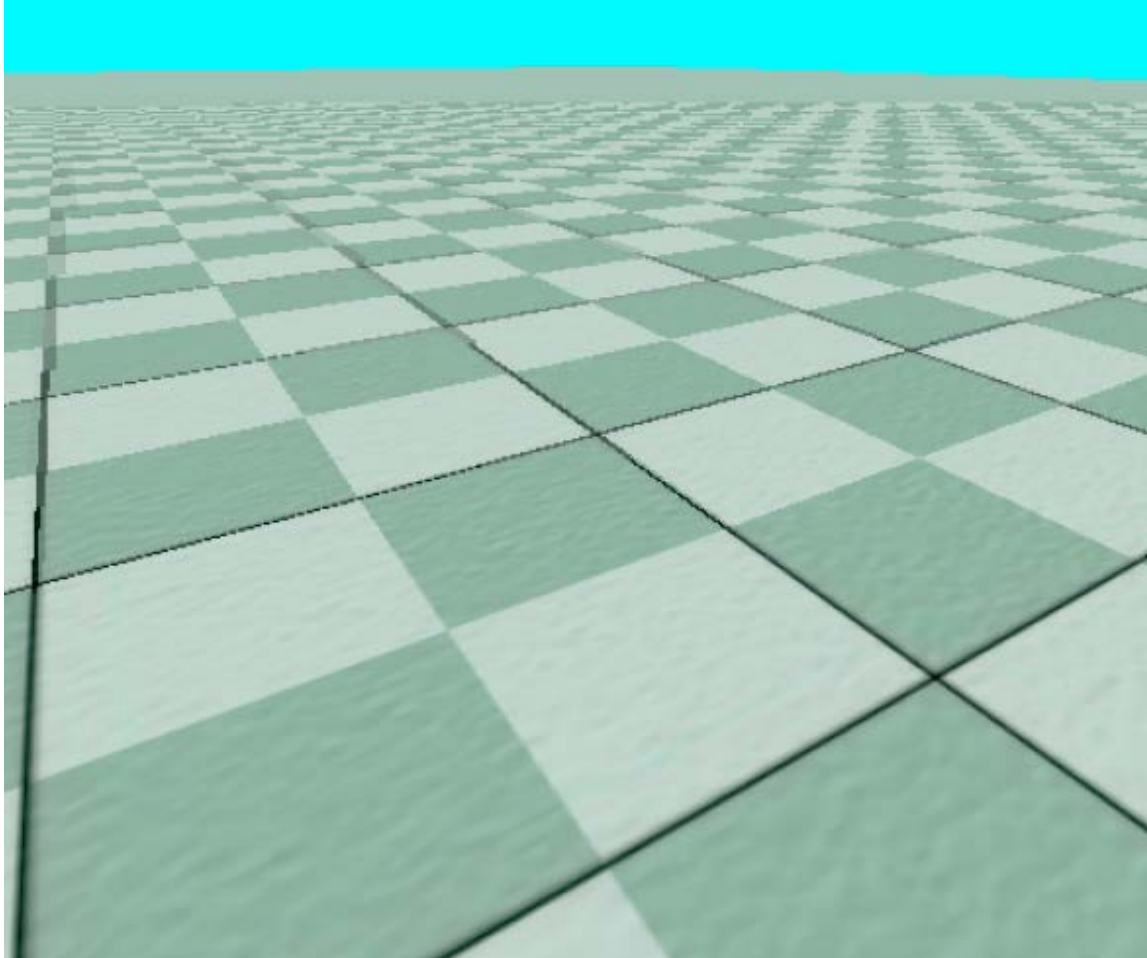 - Use nearest neighbor or bilinear interpolation as before

Mipmapping

Image plane   Camera space   Texture space

Texels

y

x

Pixel area

v

u

- Mip-map level 0
- Mip-map level 1
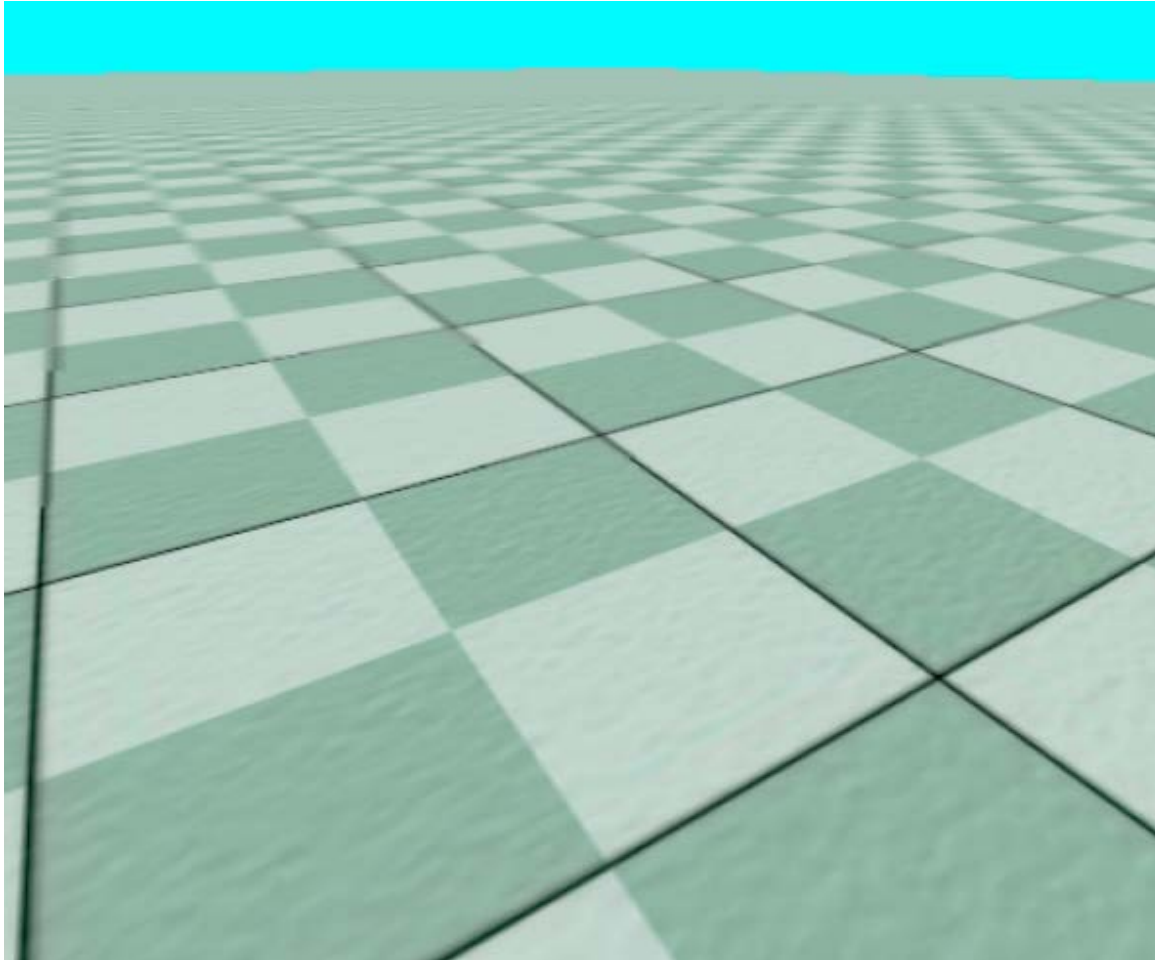- Mip-map level 2
- Mip-map level 3

UCSD

Nearest Mipmap, Nearest Neighbor

▸ Visible transition between mipmap levels

UCSD

Nearest Mipmap, Bilinear

▸ Visible transition between mipmap levels

UCSD

Trilinear Mipmapping

▸ **Use two nearest mipmap levels**

▸ E.g., if pixel corresponds to 10x10 texels, use mipmap levels 3 (8x8) and 4 (16x16)

▸ **2-Step approach:**

▸ Step 1: perform bilinear interpolation in both mip-maps

▸ Step 2: linearly interpolate between the results

▸ **Requires access to 8 texels for each pixel**

▸ **Supported by hardware without performance penalty**

UCSD

More Info

- Mipmapping tutorial w/source code:
  - http://www.videotutorialsrock.com/opengl_tutorial/mipmapping/text.php

UCSD