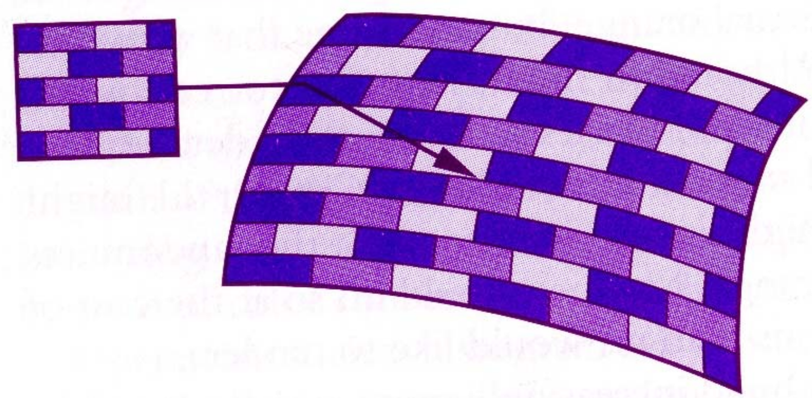
# Texture Mapping

## CS 211A

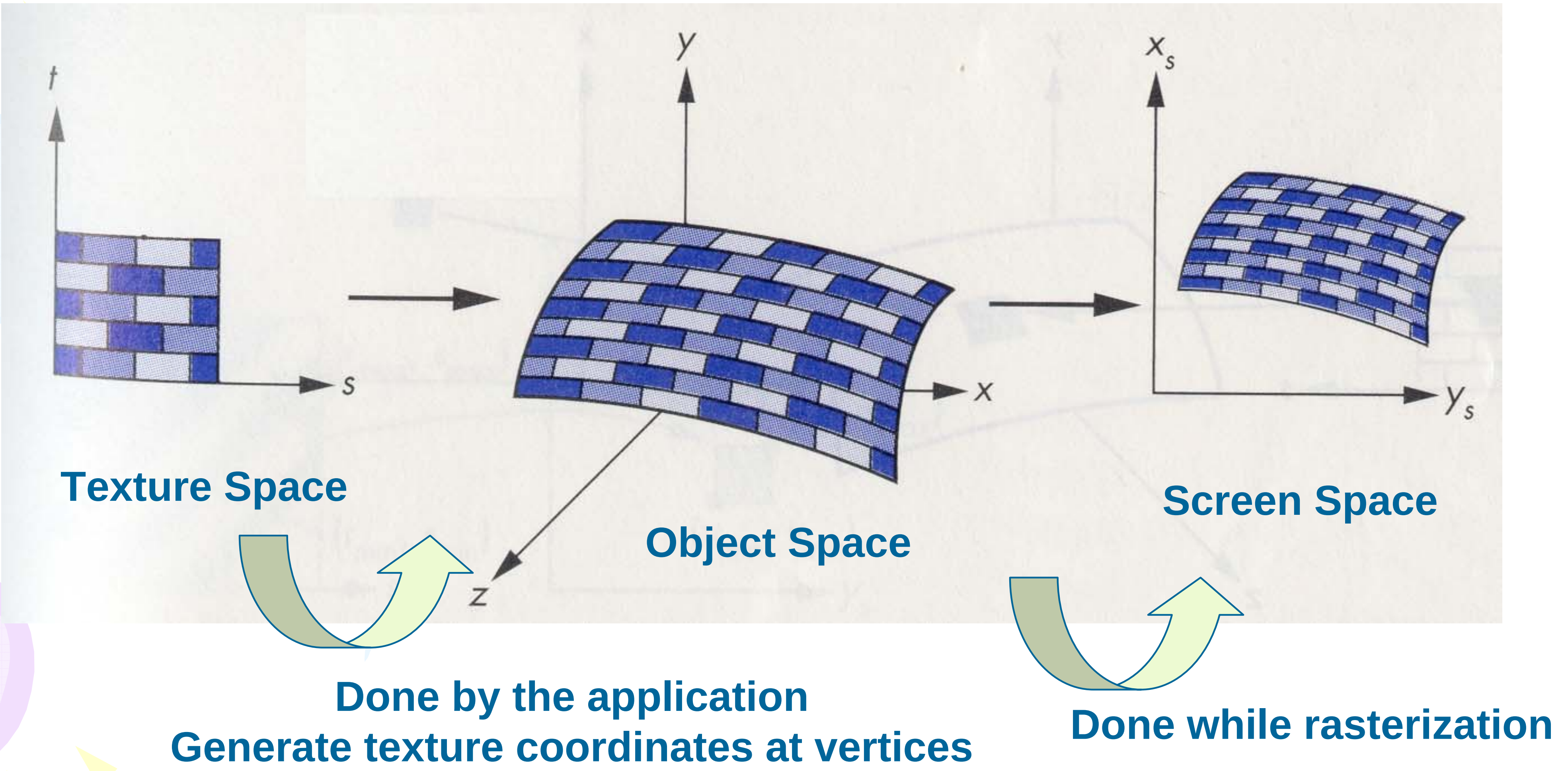
# What is Texture Mapping?

- Color is not sufficient for realistic appearances
- Wrap (Map) a image on a surface
  - Like a wall-paper
  - Like gift wrapping



# 2D Texture Mapping

- Three spaces



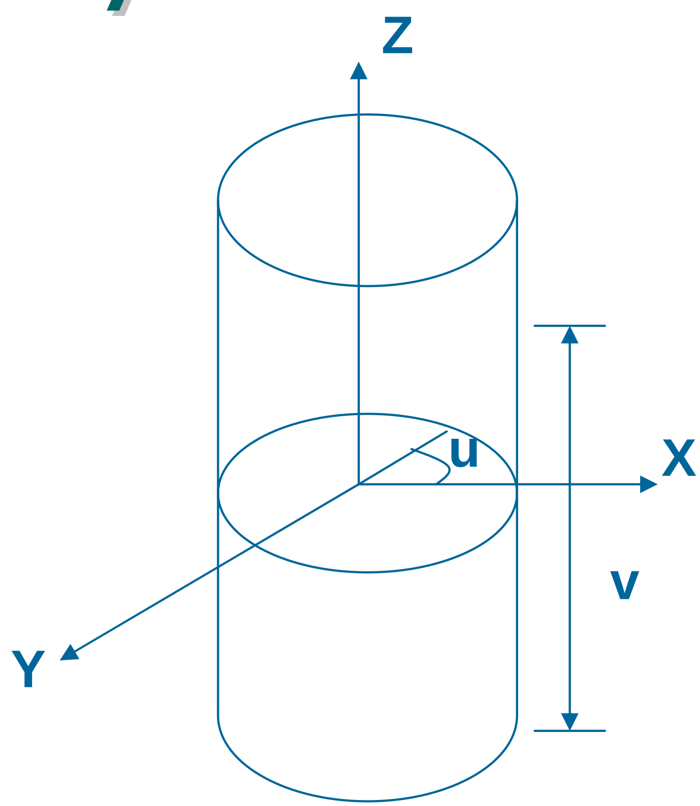
# Texture Space to Object Space

- Rectangular image mapped to arbitrary surfaces
  - The texture will get stretched differently at different places on the surface based on the curvature
  - Imagine wrapping a rectangular image on a sphere
  - Two Ways to do it

# Method 1

- Find the parametric representation of the surface defined by parameters (u,v)
  - Since 2D object embedded in real world
- Map (u,v) to (s,t) – (s,t) varies from 0 to 1
- Find the (u,v) for each vertex in the tessalated object and find the corresponding (s,t)

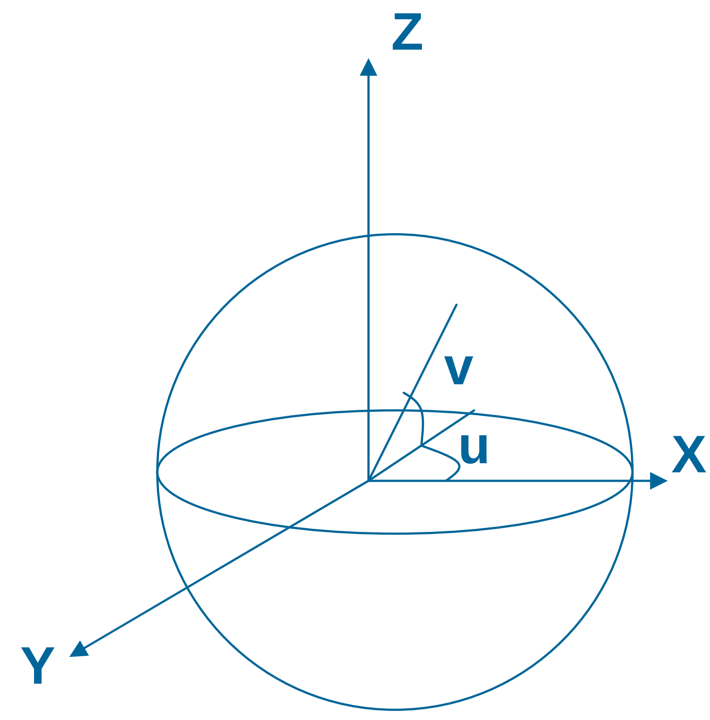
# Example: Open Cylinder



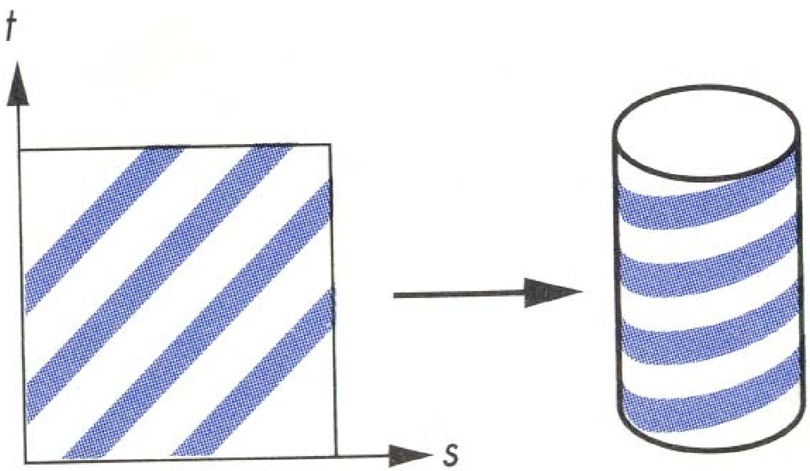

- u – angle, -180 ≤ u ≤ 180
- v – height, 0 ≤ v ≤ 1
- x = R cos(u)
- y = R sin(u)
- z = v
- Map (s,t) to (u,v)
  - s = ((u+180)/360)
  - t = v

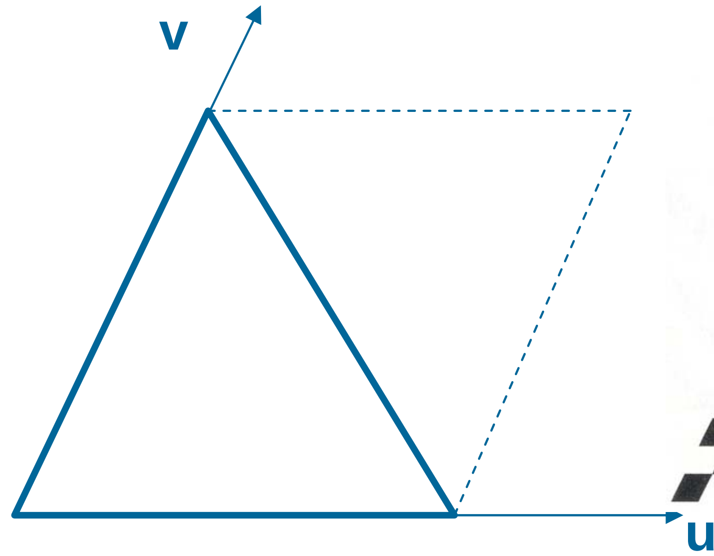
# Example: Sphere

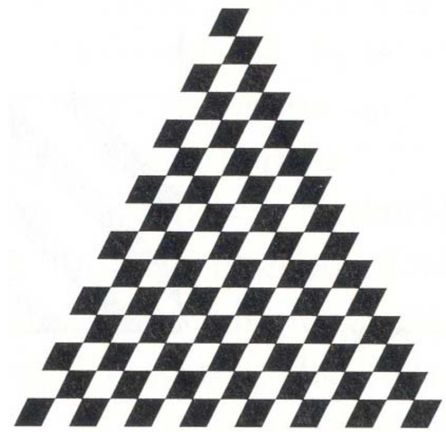
- u – horizontal angle
  - -180 ≤ u ≤ 180
- v – vertical angle
  - -90 ≤ v ≤ 90
- x = R cos (v) cos (u)
- y = R cos (v) sin (u)
- z = R sin (v)
- Map (s,t) to (u,v)
  - s = (u+180)/360
  - t = (v+90)/180

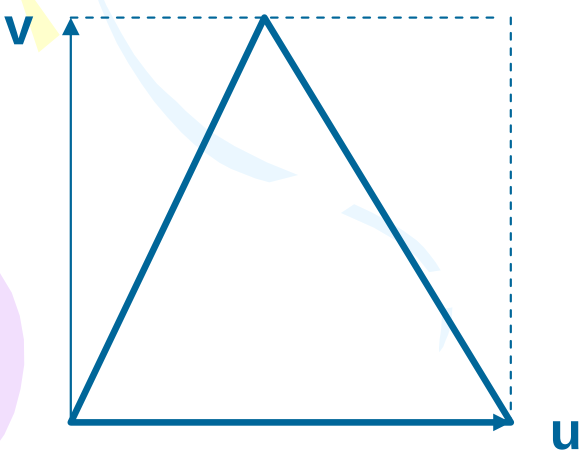


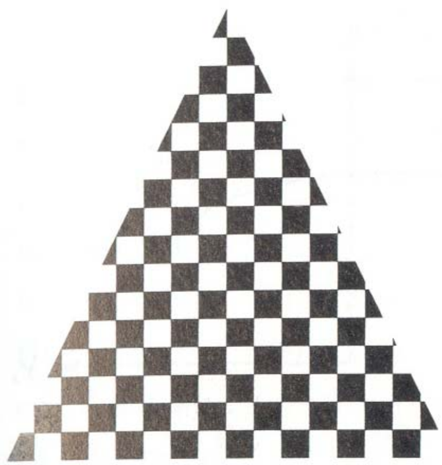
# Results



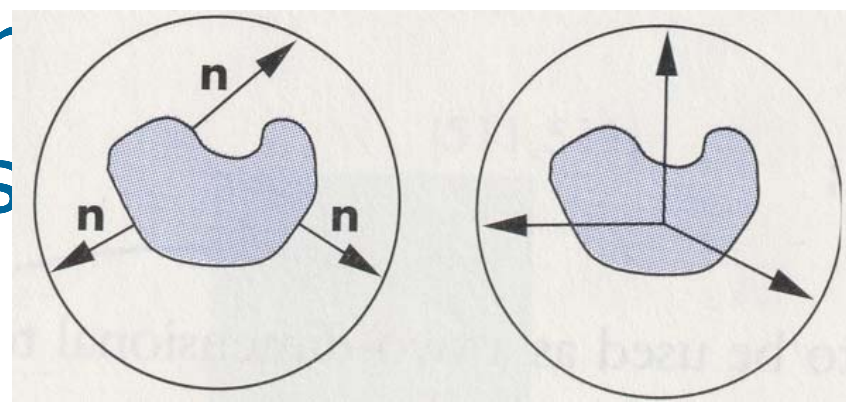


Depends on the parameterization

# Method 2: Intermediate Geometry

- Difficult to parameterize arbitrary geometry
- Define intermediate simple surface and parameterize it: a plane, sphere or cylinder
- Enclose arbitrary geometry within simple geometr
- More close thes ter the mapping

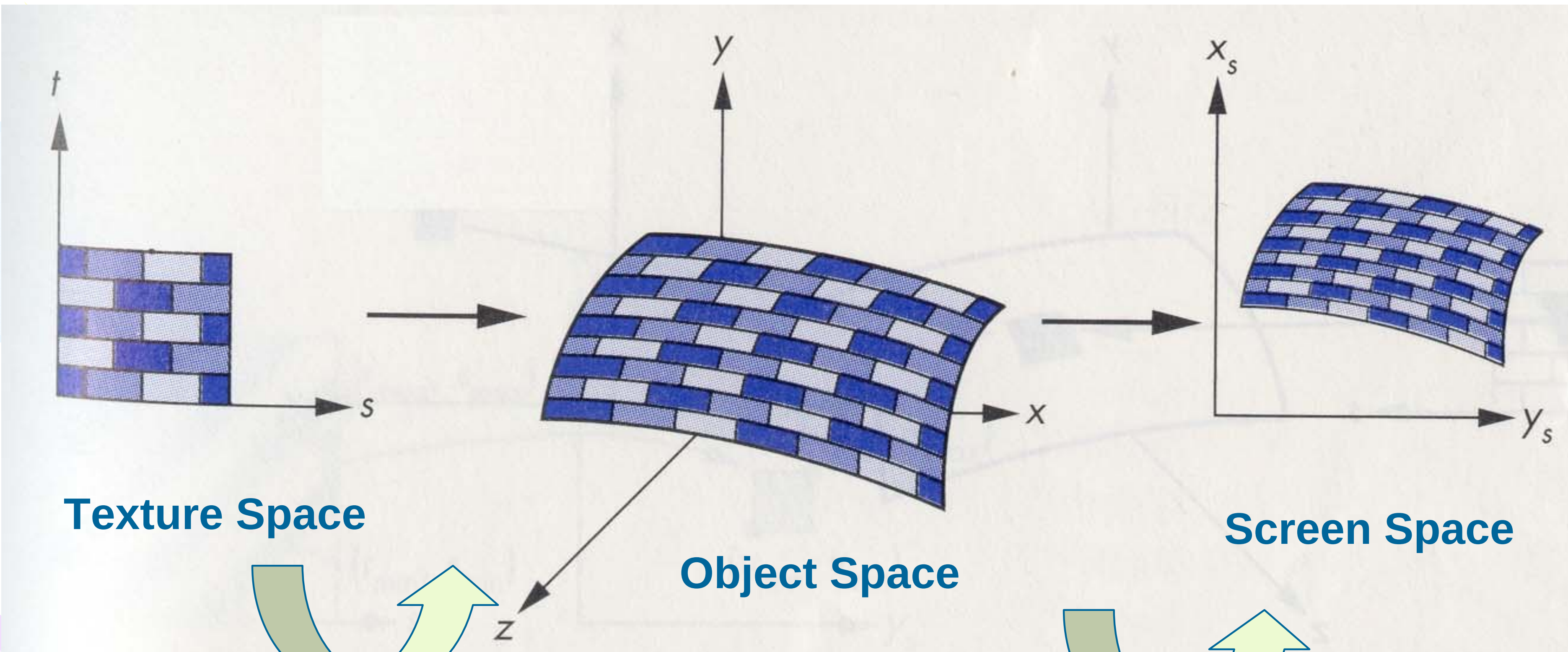


# Result (Planar Mapping)

# Result (Cylindrical Mapping)

# 2D Texture Mapping

- Three spaces

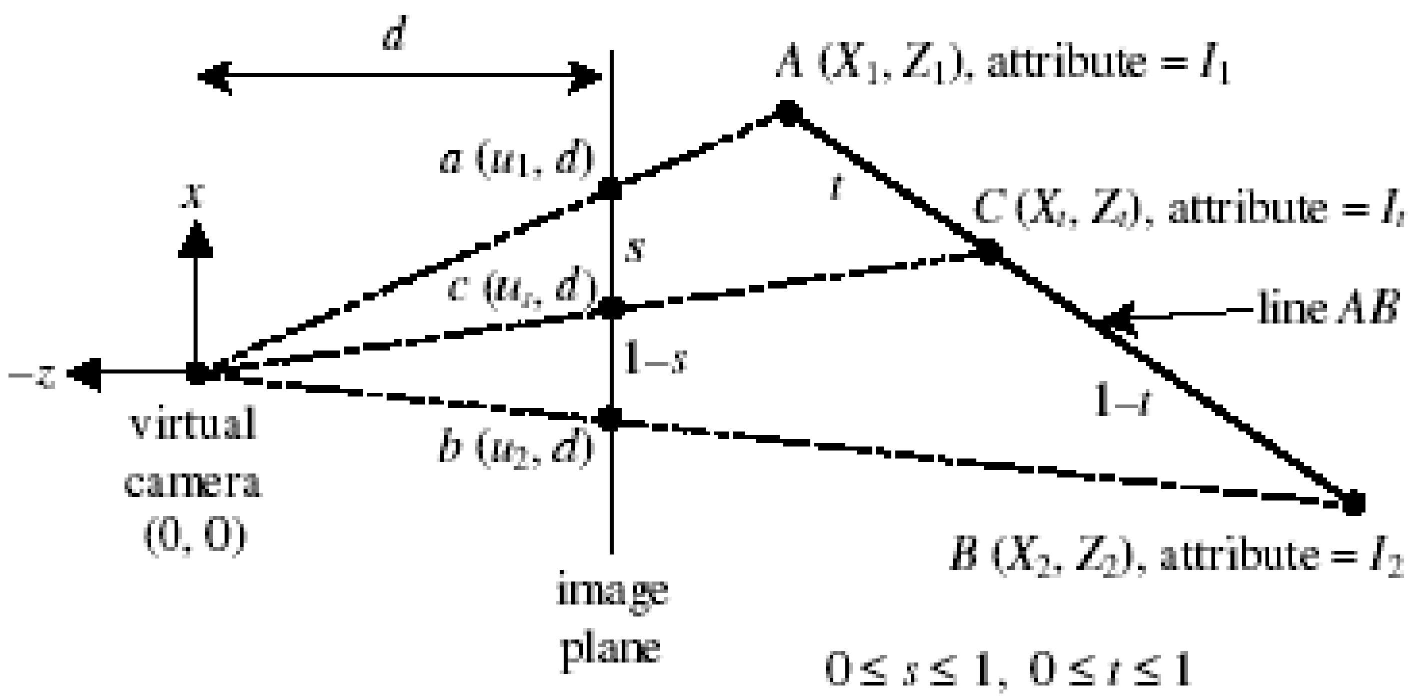



**Done by the application**
**Generate texture coordinates at vertices**

**Done while rasterization**

# Object Space to Screen Space

- The texture coordinates are known in the object space
- Needs to be interpolated in the screen space

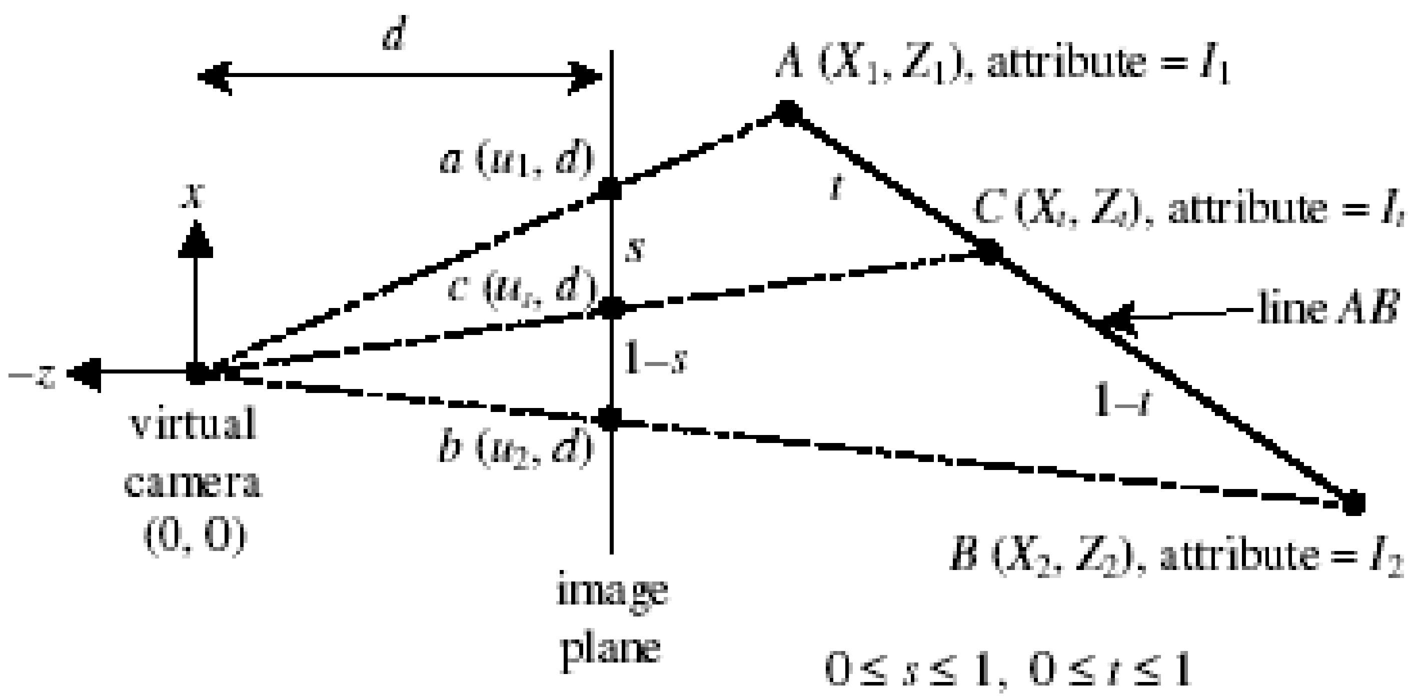


# Interpolation of Attributes

$$I_t = I_1 + t(I_2 - I_1)$$

$$t = \frac{sZ_1}{sZ_1 + (1-s)Z_2}$$

$$I_t = \left(\frac{I_1}{Z_1} + s\left(\frac{I_2}{Z_2} - \frac{I_1}{Z_1}\right)\right) \Big/ \frac{1}{Z_t}$$



# Sampling the Texture

- You have FP numbers between 0 and 1 for each pixel
- How do you get the colors from the texture image?

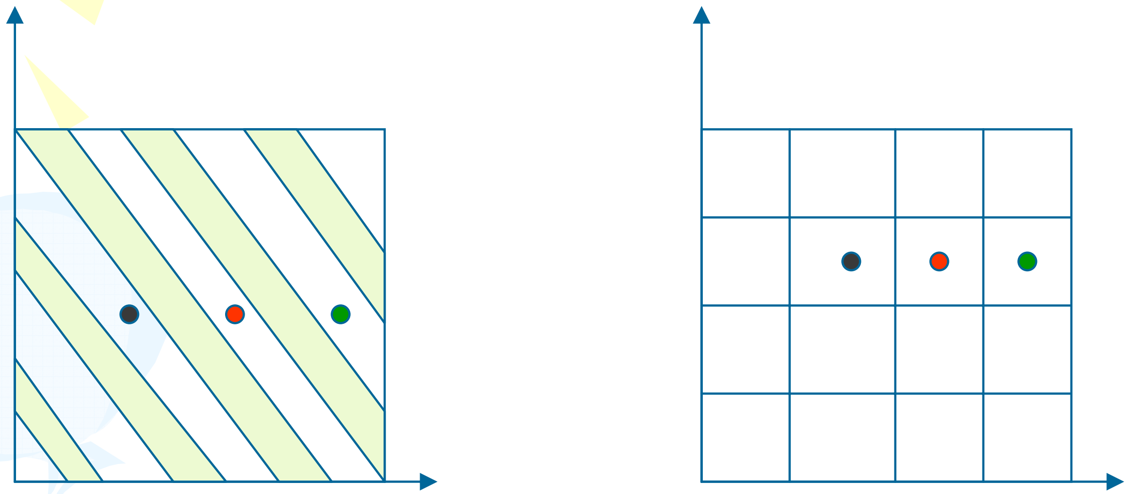
# Point Sampling

- Multiply by the texture size to generate another FP value
- Round off the FP values to integers (GL_NEAREST)
- Pick the color of the integer texel

# Linear Interpolation

- Multiply by the texture size to generate another FP value
- Interpolate the color from the four nearest texels using bilinear interpolation (GL_LINEAR)
- Does not remove aliasing completely since sampling is still inadequate

# Aliasing Problems

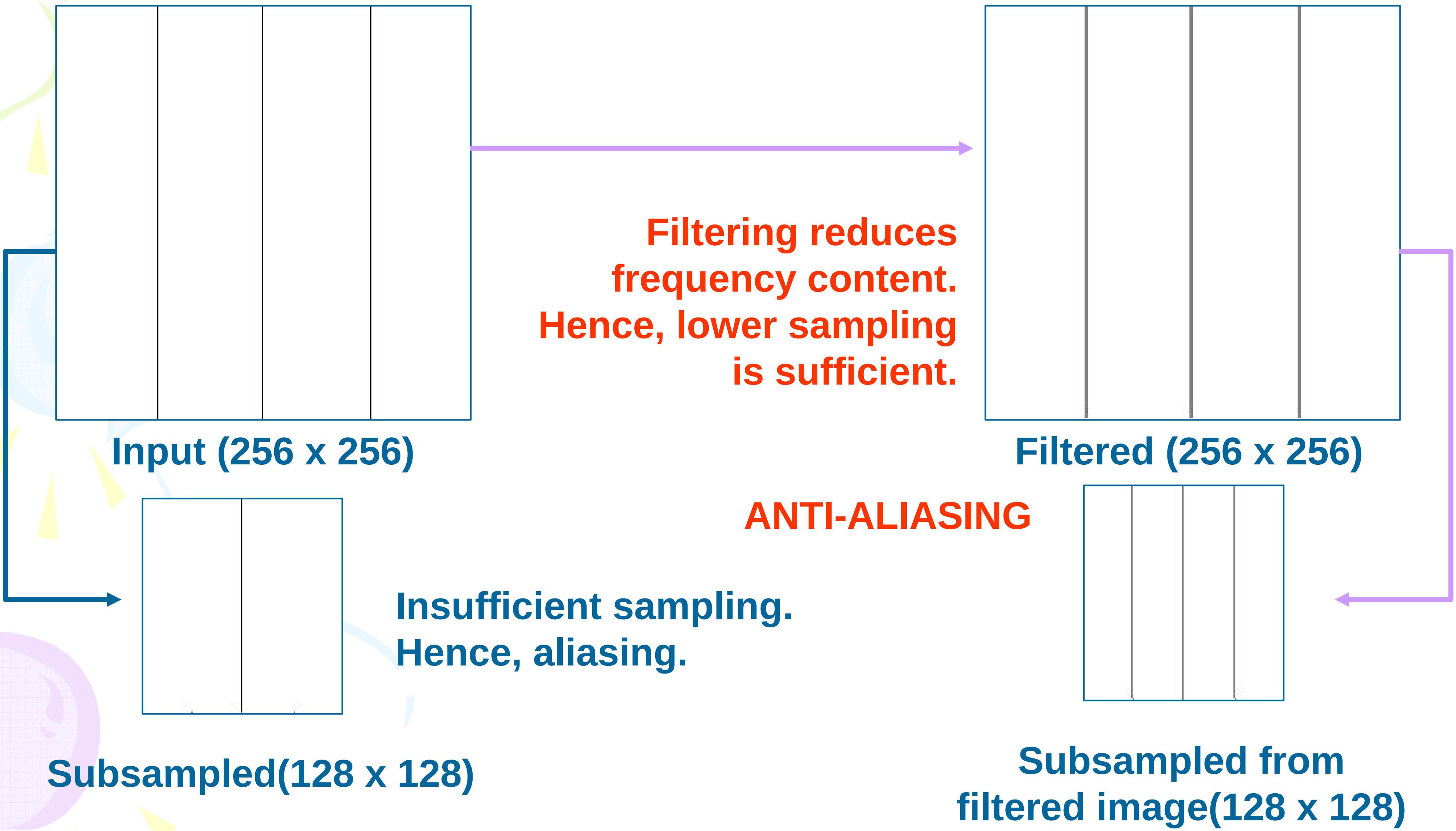


- Scan conversion samples the texture
- If # of pixels in triangle much smaller than the size of texture, it cannot sample all frequencies adequately
- Miss the stripes completely

# Reducing Frequency content

- Filter the image
  - Simplest: Averaging pixels (Box Filter)
- Reduces the frequency content
- Smaller image size
  - Matched is # of pixels triangle project to
  - Hence, sufficient samping

# How does it help?



# Level of Details (LODs)

- Keep many LODs of same image
- Filtered and subsampled
  - Reduced frequency content
- Pick the correct level based on the size of the projected triangle
- Anti-aliased image

# Mipmapping: Efficient storage and retrieval of LODS

- Special way of storing images of different resolutions
- $T_1$:128x128 (RGB)
- $T_2$:64x64 (RGB)
- $T_3$:32x32 (RGB)
- And so on…
- Choose appropriate resolution based on screen space projection

**Size: 4 x original texture**