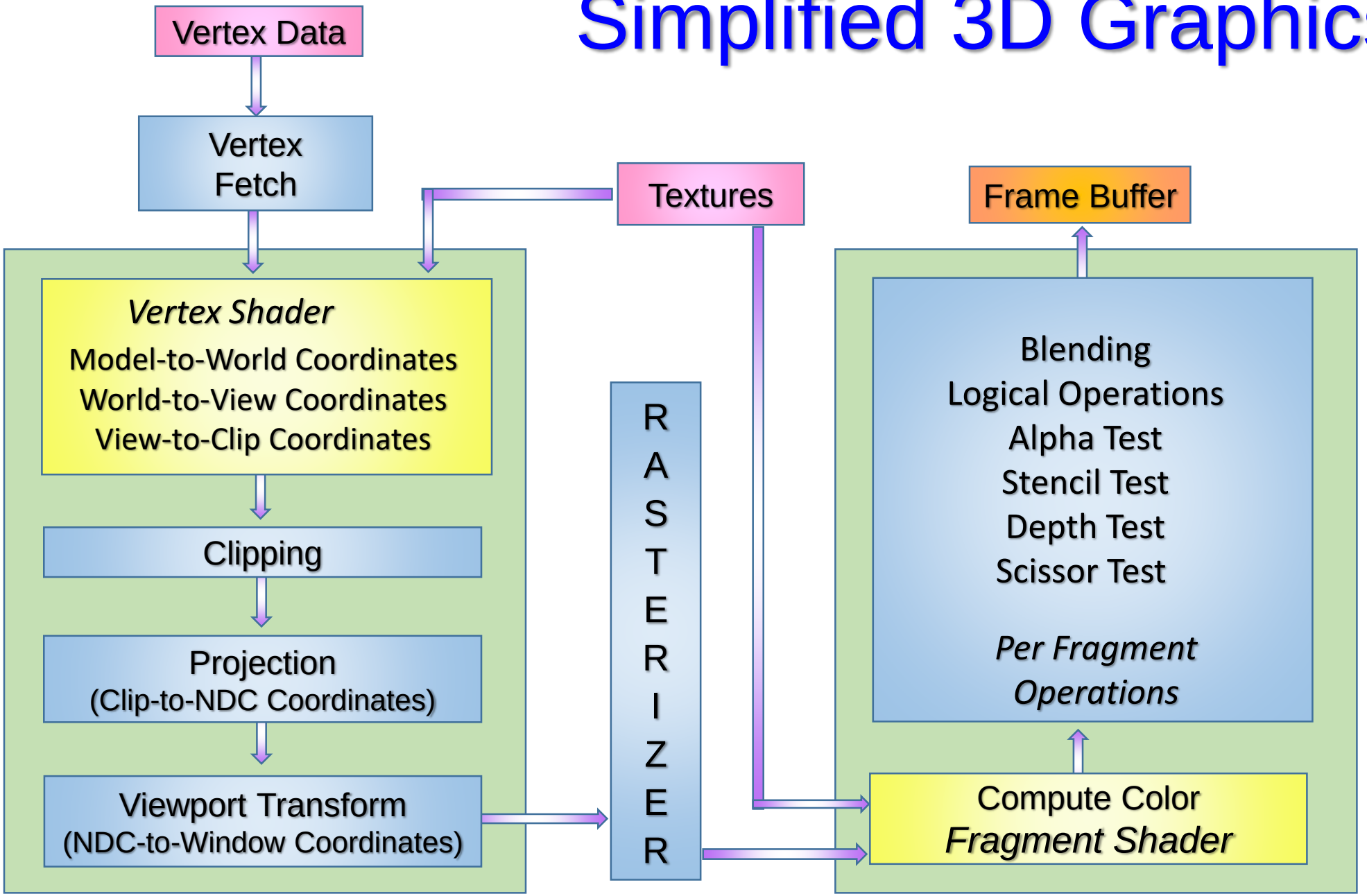
Introduction to Computer Graphics

Introduction to Texture Mapping

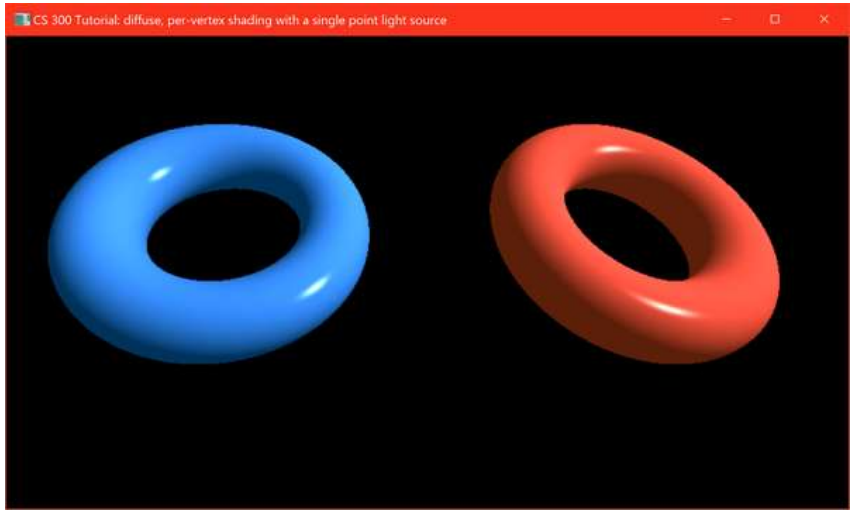
Prasanna Ghali

Simplified 3D Graphics Pipe



What is Texture Mapping? (1/3)

- Images synthesized from polygonal models lack details and richness of real surfaces
  - Imperfections caused by wear and tear
  - Irregular color variations
  - Properties of natural materials such as stone, wood, soil, …



What is Texture Mapping? (2/3)

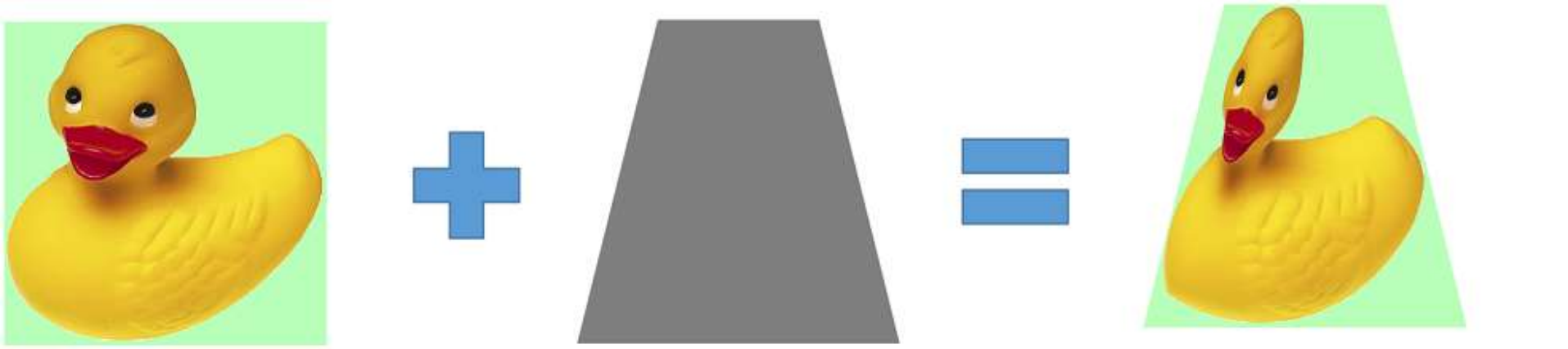
- Texture mapping is technique for taking simple polygon and giving it appearance of something more complex
- Adds detail to polygonal surfaces using images, functions, or other data sources to make surface color vary from point to point
- De rigueur in modern GPUs to add detail without increasing complexity of 3D models

What is Texture Mapping? (3/3)

- Technique for adding detail to surfaces using images, functions, or other data sources
- We're concerned with image mapping

What is Texture Mapping?

- Texture mapping involves mapping an image called *texture* to simpler surface

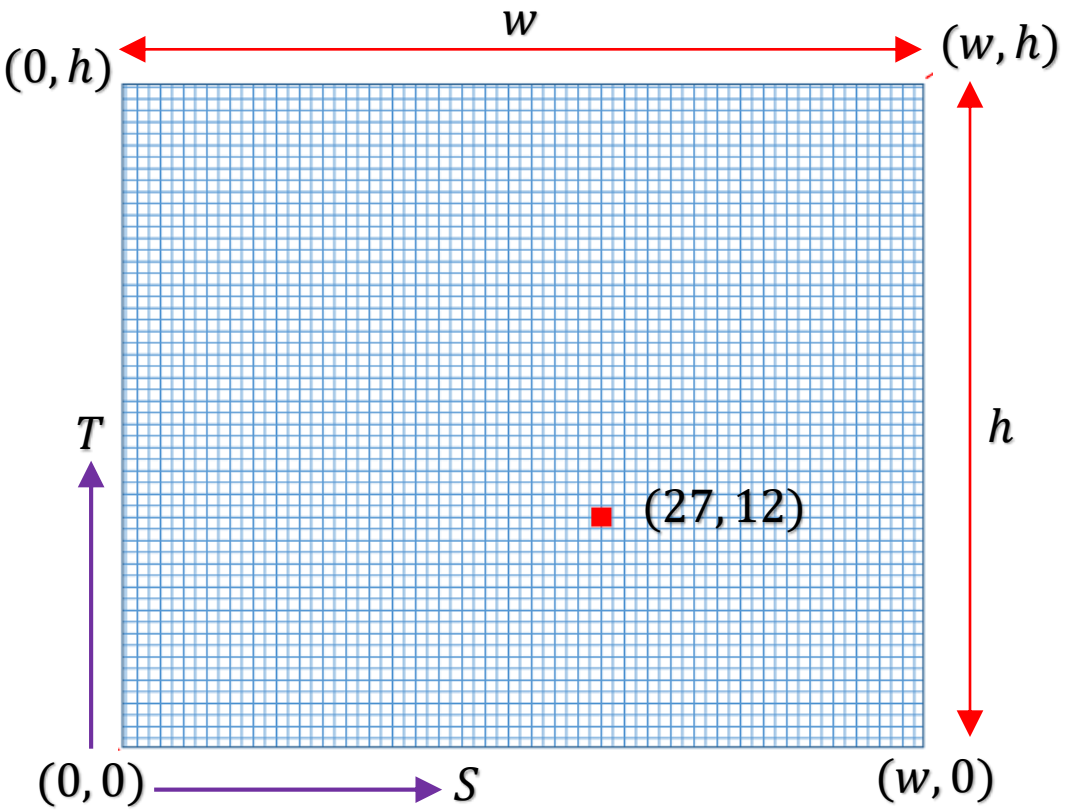


Texture Images and Texels

- Texture image is *discrete collection* of color samples
  - Each color sample called *texel* (*tex*ture *el*ement)
  - Better performance when images have powers-of-two dimensions
- One-dimensional textures suitable for linear primitives such as lines
- Two-dimensional textures suitable for planar primitives such as triangles – we're only concerned with rectangular textures
- Three-dimensional textures suitable for volumetric objects

Texture Images and Texels (1/2)

- To identify texels, texture images come with their own *discrete* coordinate system called *texel space*
- In OpenGL, texel referenced by integral 2-tuple (S, T) with origin at bottom-left corner
  - Reverse of notation for referencing texel's corresponding memory location in C/C++

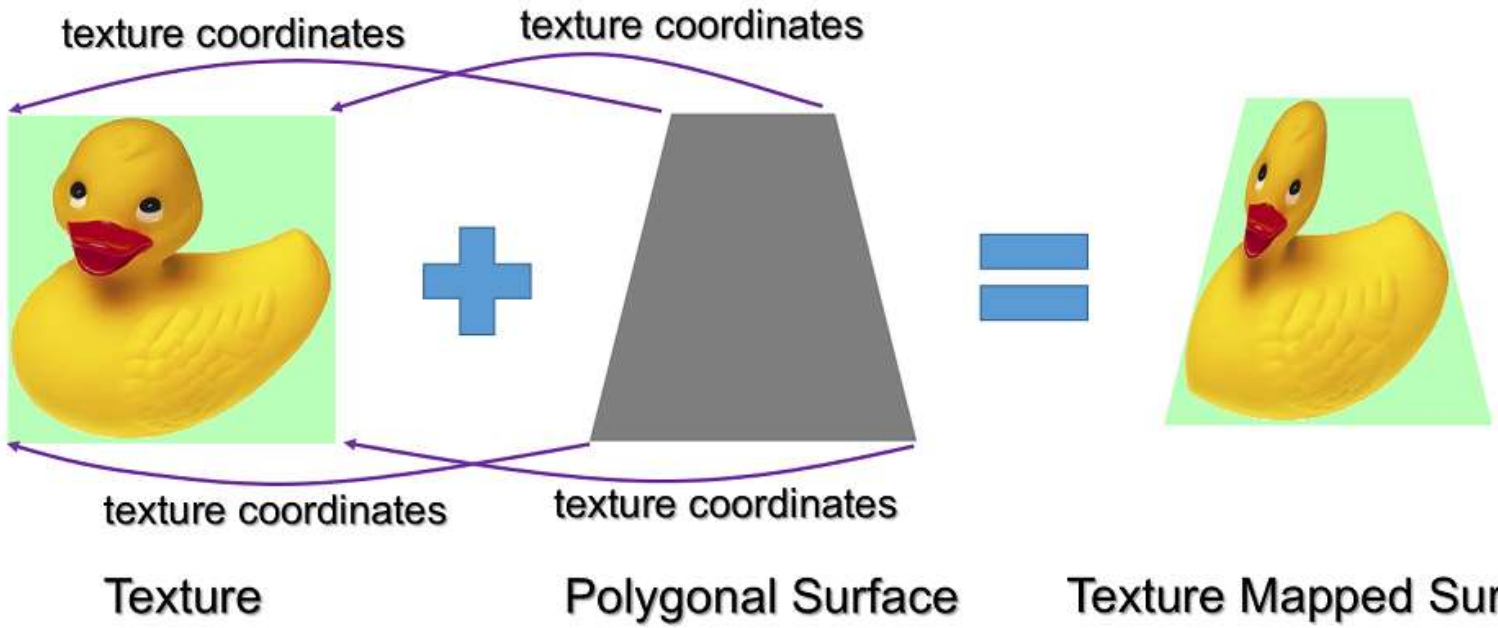


Texture Images and Texels (2/2)

- In addition to width and height, other consideration is format of data specified at each texel
- Image sample represented in variety of formats:
  - 16 or 24 bit RGB, 32 bit RGBA, various color index methods, …

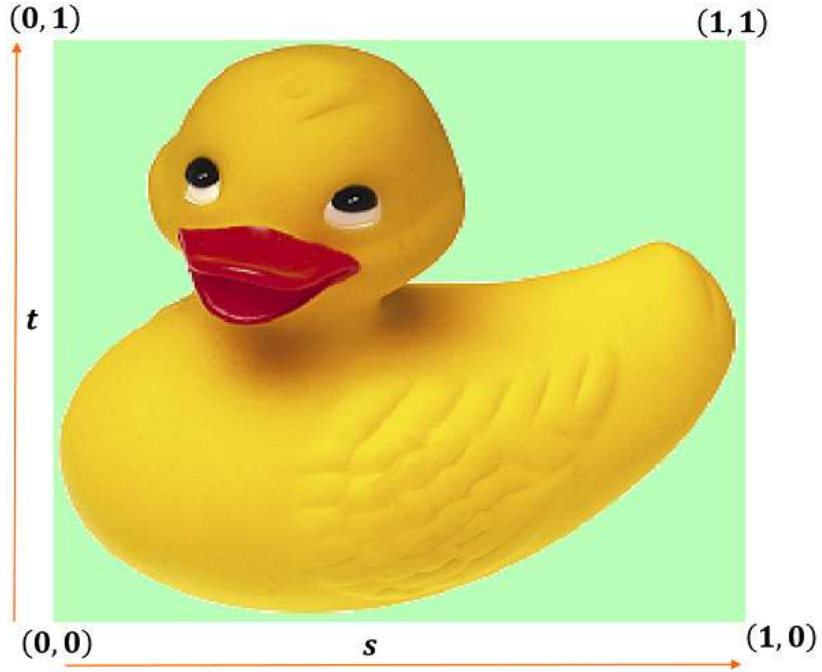
Texture Coordinates

- *Texture coordinates* – specified at each vertex – define where texels appear on polygonal (texture) surface
  - They control texture's placement on surface by determining which texels in texture image correspond to which vertices on surface



Texture Space (1/2)

- Texture coordinates are specified in normalized coordinate system called *texture space*
  - OpenGL uses s for horizontal axis ranging from left to right and t for vertical axis ranging from bottom to top with origin at bottom-left

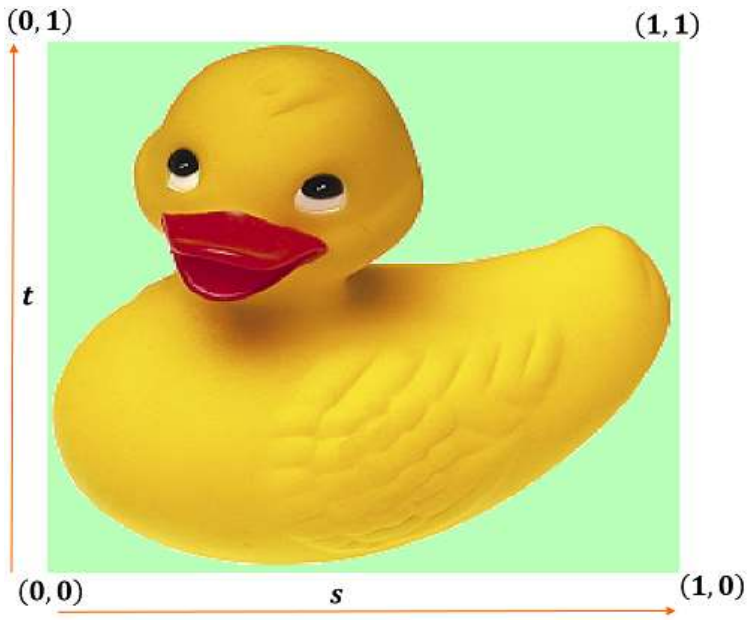


Texture Space (2/2)

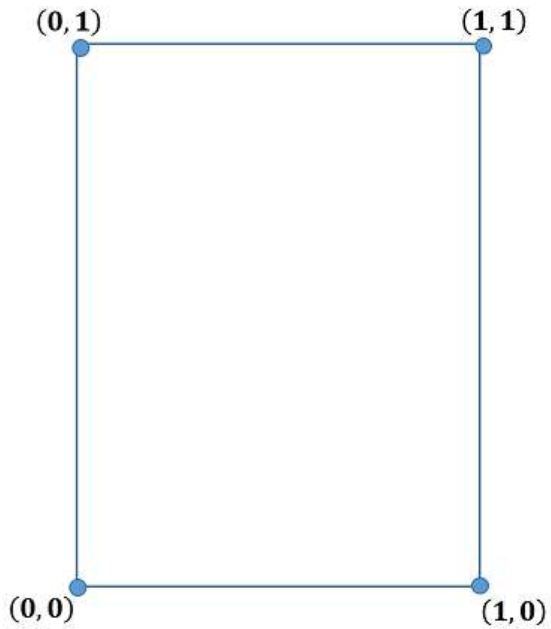
- Unit texture space $(s, t) \in [0, 1] \times [0, 1]$ is mapping of *discrete texel* space to *continuous texture* space
- More practical and efficient to use texture coordinates rather than texel coordinates:
  - Texture coordinates are independent of texture image's width and height and therefore don't need to be recomputed when texture image's dimensions change or when different texture image is used

Understanding Texture Coordinates (1/2)

- On left, we've texture image defined in texture space
- On right, we've polygon onto which entire texture image is to be pasted



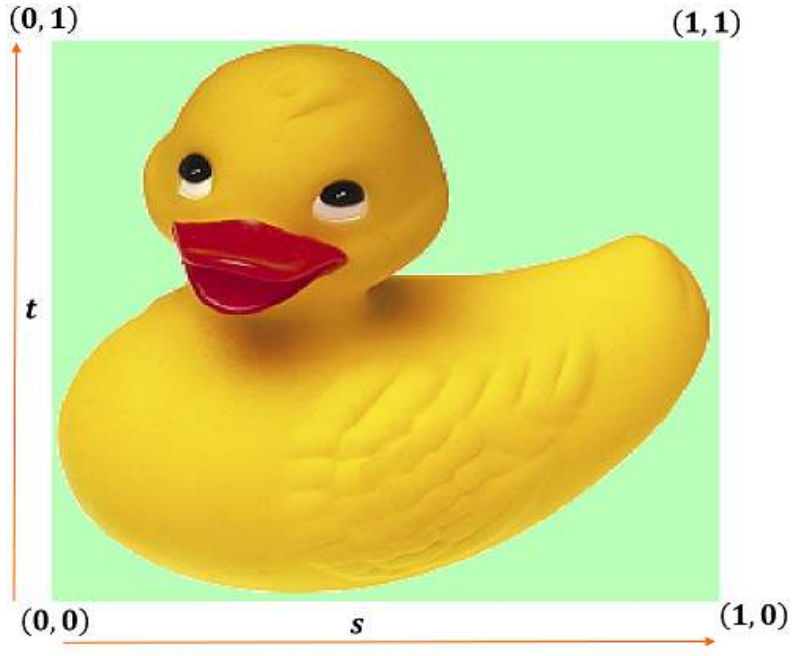
Texture Image



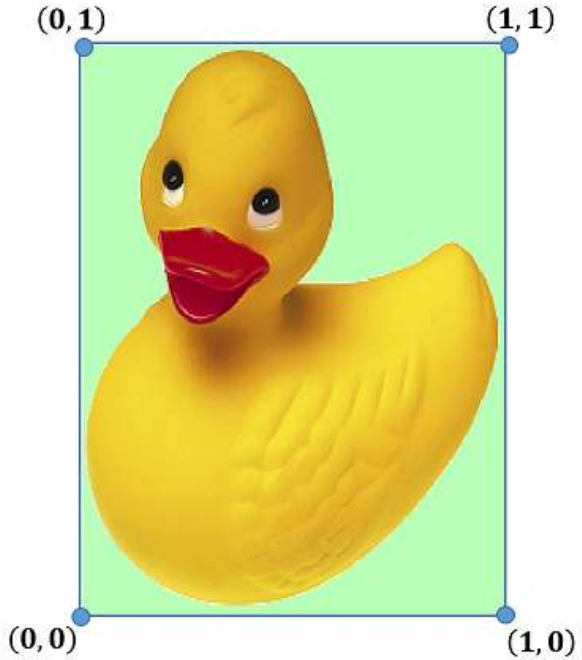
Polygon with texture coordinates

Understanding Texture Coordinates (2/2)

- Image will be “stretched” to fit polygon
  - Rasterizer will *interpolate* texture coordinates assigned to each vertex to compute texture coordinates for interior points



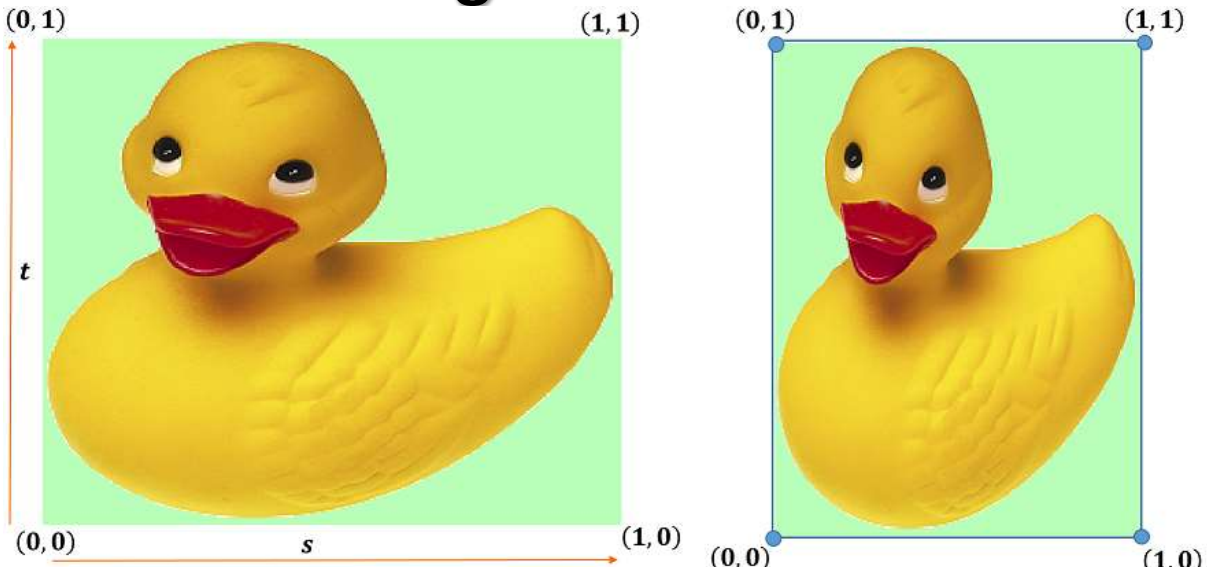
Texture Image



Texture Mapped Polygon

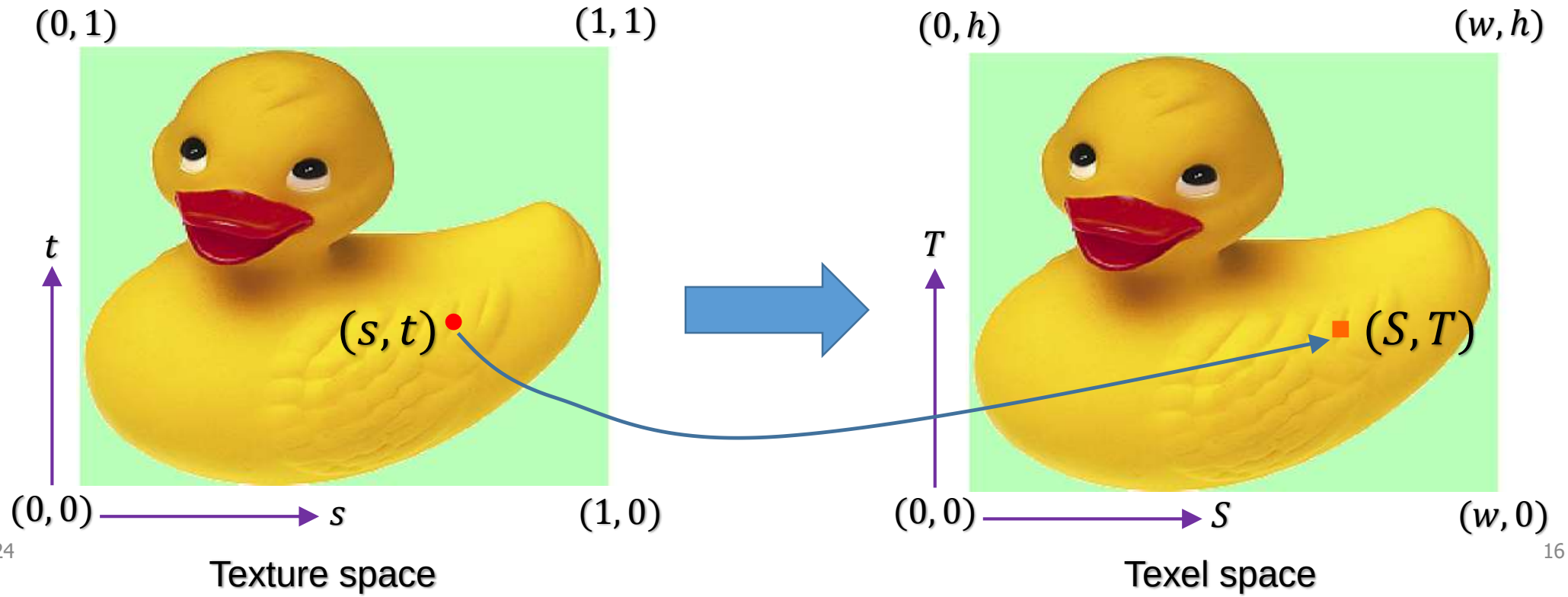
Understanding Texture Coordinates (2/2)

- Rasterizer will interpolate texture coordinates assigned to each vertex to compute texture coordinates for interior points
- Next, texture coordinates are mapped to texel coordinates to retrieve color from image



Mapping Texture Coordinates to Texel Coordinates (1/2)

$$(s * w, t * h) \rightarrow (S, T)$$



Mapping Texture Coordinates to Texel Coordinates (2/2)

- Fragment (x, y) on surface with interpolated texture coordinates (s, t) has floating-point texel coordinates $(S, T) = (w * s, h * t)$
- *Point sampling* selects single texel value containing texture coordinate's sample point (S, T)
  - Integer coordinates of selected texel are $(\lfloor S \rfloor, \lfloor T \rfloor)$

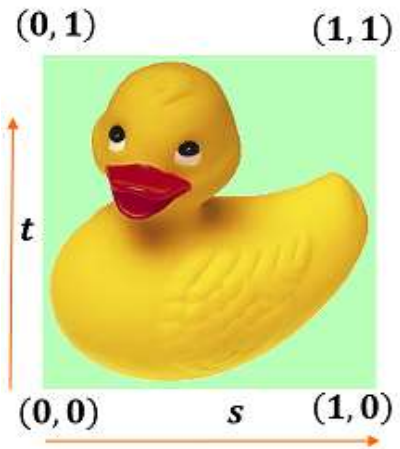
Wrapping Modes (1/2)

- Values of s and t outside $[0, 1]$ don't map to image, but such values are still valid as texture coordinates
- Why would texture coordinates be outside $[0, 1]$?
  - When we want to texture map an image across polygon surface with dimensions larger than texture dimensions

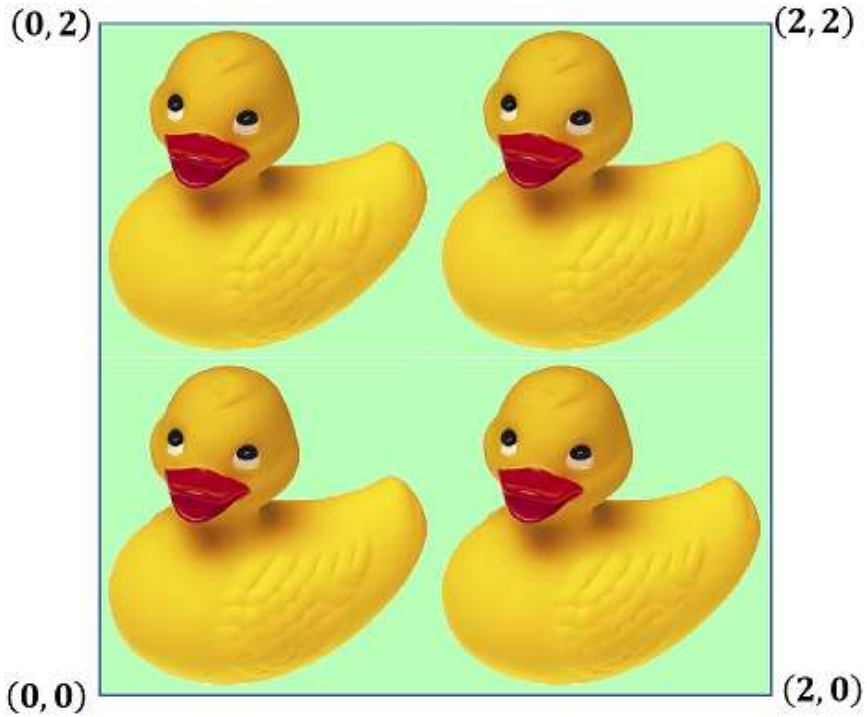
Wrapping Modes (2/2)

- To paste (small) texture image on (large) polygon, texture coordinates must be *wrapped* for corresponding texel coordinates to access valid texel data
- Two basic wrapping modes:
  - *Repeat* – repeats image by ignoring integer part of texture coordinate and retaining fractional part
  - *Clamp-to-edge* – clamps texture coordinate to closest texel on edge of texture map

Wrapping with Repeat (1/2)



Texture Image



Polygon with texture coordinates

Wrapping with Repeat (2/2)

- Image *repeated* across polygon independently in horizontal and vertical directions
  - Also known as *tiling*
- How?
  - Integer part of either s or t, or both s and t is dropped

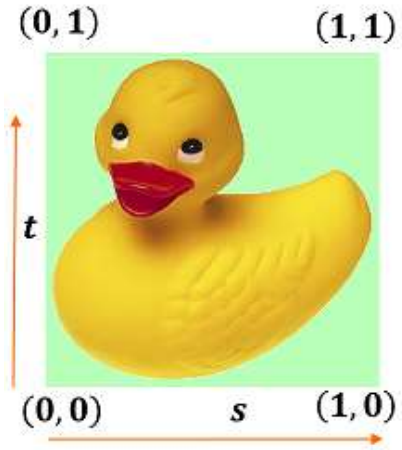
$$repeat(s) \leftarrow s - \lfloor s \rfloor$$

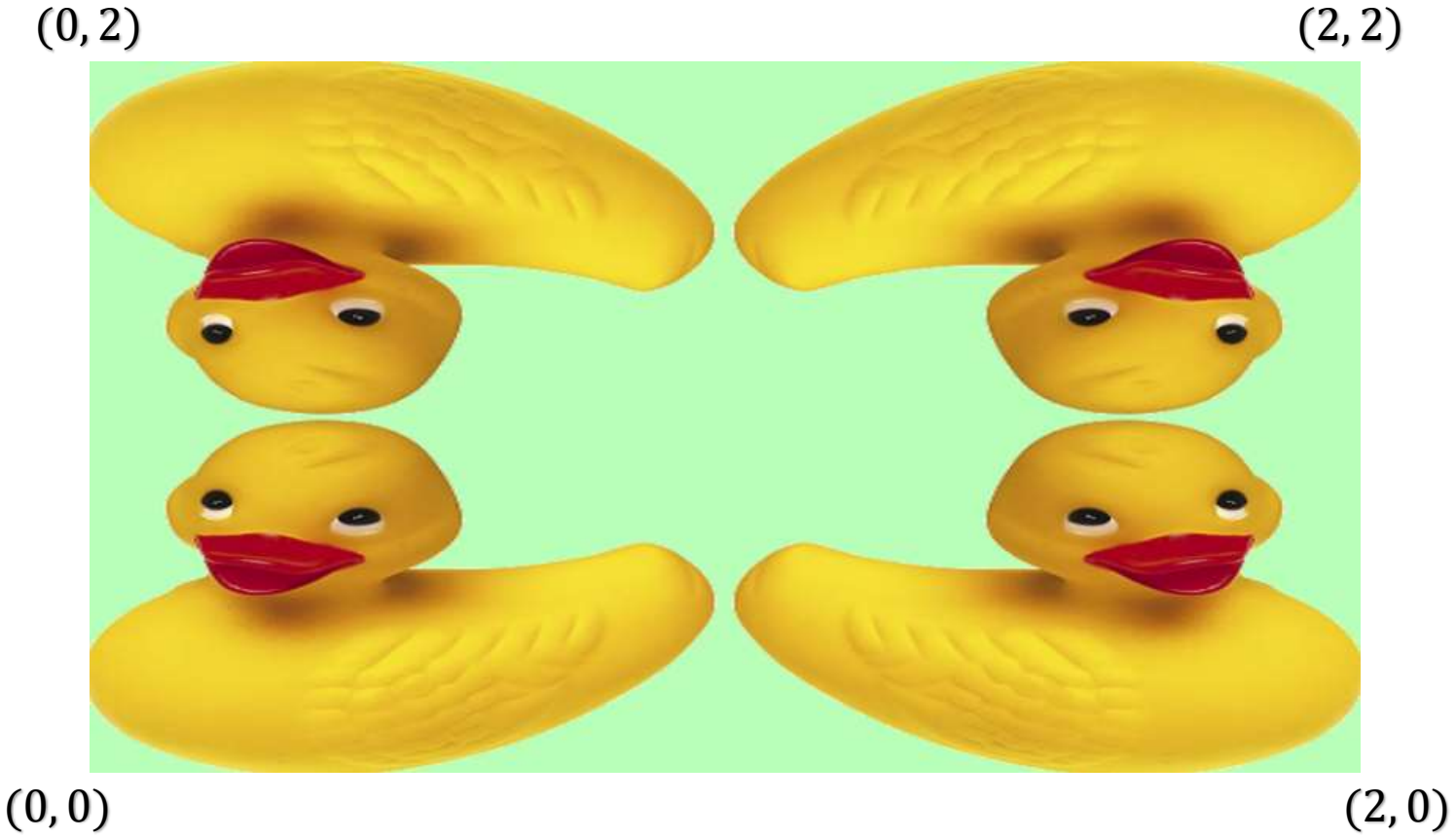
$\lfloor s \rfloor$ is largest integer smaller than real number s

$$repeat(t) \leftarrow t - \lfloor t \rfloor$$

$s - \lfloor s \rfloor$ is fractional part of real number s

Wrapping with Mirror (1/2)





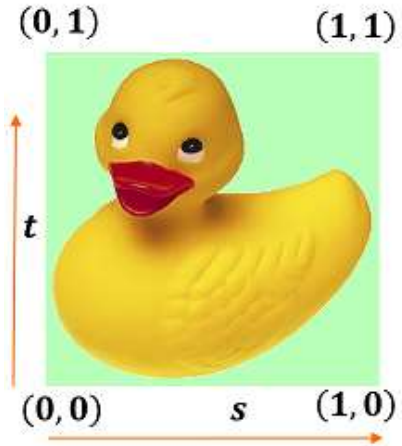
Wrapping with Mirror (2/2)

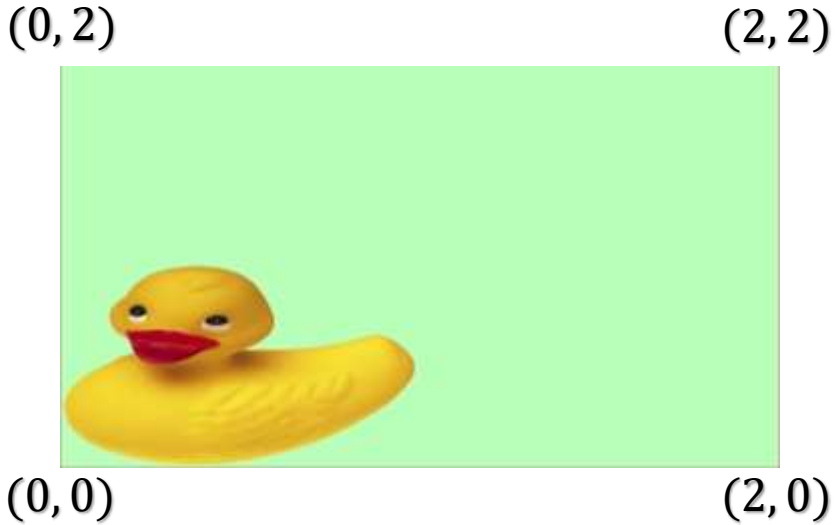
- Image *repeated* across polygon independently in horizontal and vertical directions but *flipped* on every other repetition

$$mirror(s) \leftarrow \begin{cases} s - \lfloor s \rfloor & \lfloor s \rfloor \; is \; even \\ 1.0 - (s - \lfloor s \rfloor) & \lfloor s \rfloor \; is \; odd \end{cases}$$

$$mirror(t) \leftarrow \begin{cases} t - \lfloor t \rfloor & \lfloor t \rfloor \; is \; even \\ 1.0 - (t - \lfloor t \rfloor) & \lfloor t \rfloor \; is \; odd \end{cases}$$

Wrapping with Clamp-to-Edge





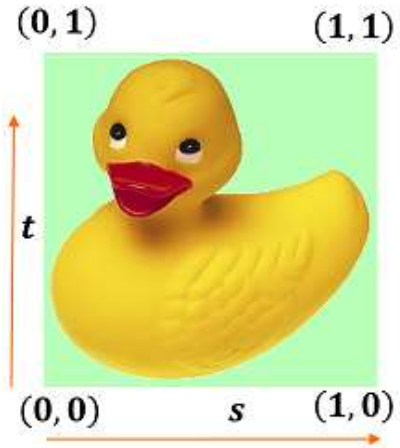
$$clamp(s) \leftarrow max(min(s, 1.0), 0.0)$$

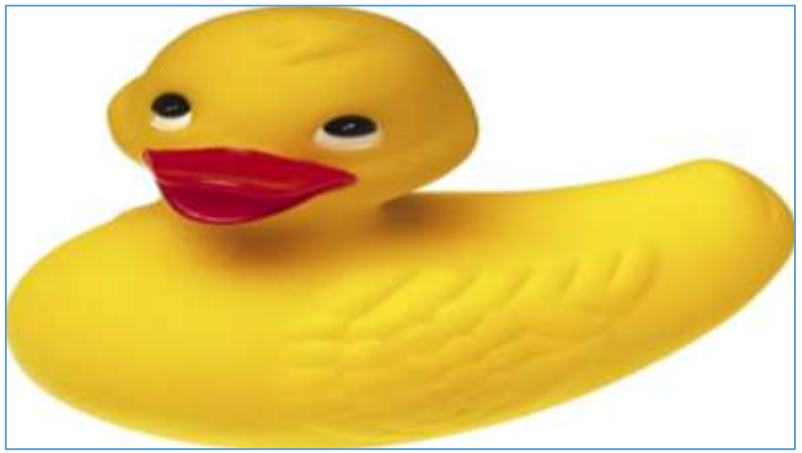
$$clamp(t) \leftarrow max(min(t, 1.0), 0.0)$$

Color Combiner or Texture Blending Modes

- At each fragment, we've color from texturing and say color from lighting (we'll call this color a surface color)
- Texture color can be further transformed with surface color using color combine or texture blending modes
  - Decal
  - Modulate
  - Replace

Decal Mode (1/2)



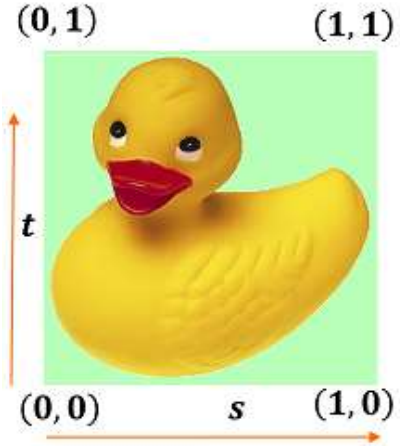


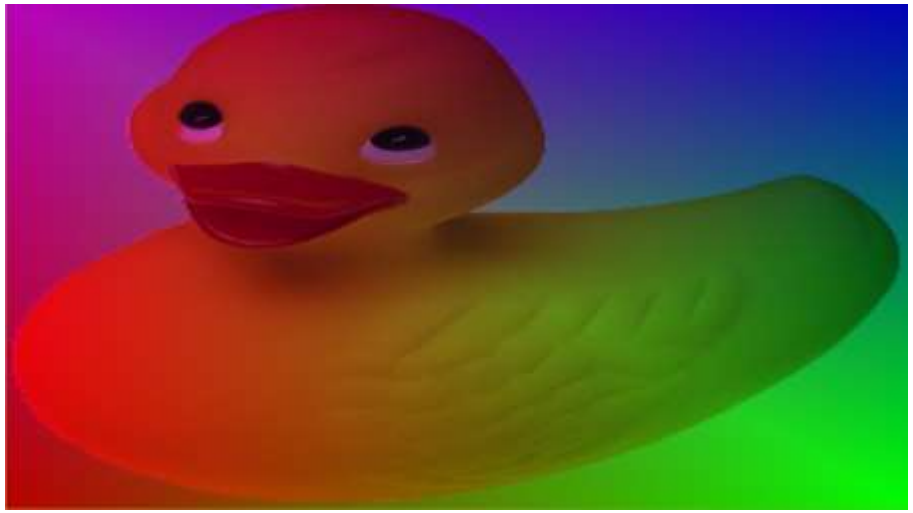
Decal Mode (2/2)

- Texture alpha value α blends between surface and texture colors
- If surface color is $C_s = (r_s, g_s, b_s)$ and texture color is $C_t = (r_t, g_t, b_t, \alpha_t)$, then final color is

$$C = C_s + \alpha_t(C_t - C_s)$$

Modulate Mode (1/2)



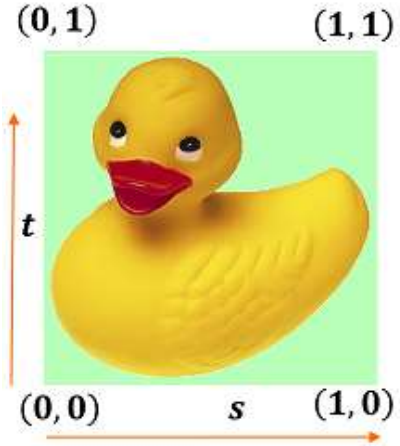


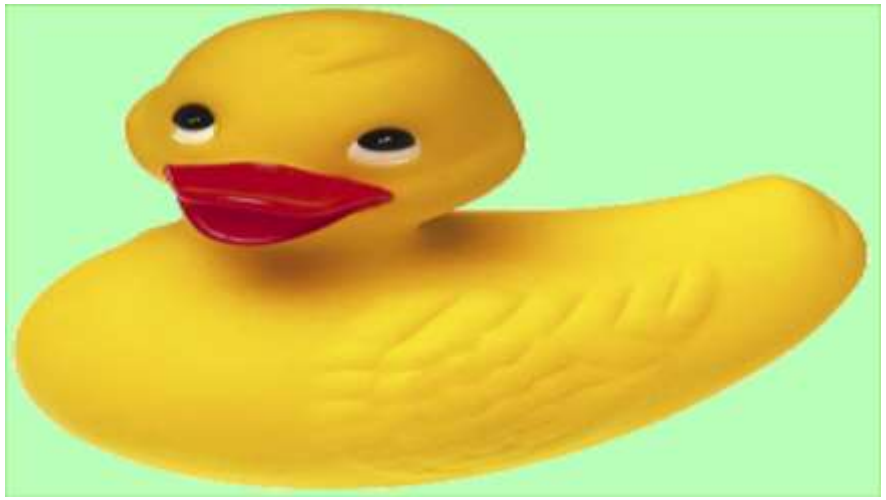
Modulate Mode (2/2)

- Texture color modulated by surface color to give surface shaded appearance
- If surface color is $C_s = (r_s, g_s, b_s)$ and texture color is $C_t = (r_t, g_t, b_t)$, then final color is

$$C = C_s \otimes C_t = (r_s r_t, g_s g_t, b_s b_t)$$

Replace Mode (1/2)





Replace Mode (2/2)

- Texture color replaces/overwrites surface color
- If surface color is $C_s = (r_s, g_s, b_s)$ and texture color is $C_t = (r_t, g_t, b_t)$, then final color is

$$C = C_t = (r_t, g_t, b_t)$$

Texture Mapping in OpenGL

- In order to use texture mapping in OpenGL applications, following steps must be implemented:
  - Create a texture object and load texel data into it
  - Specify sampling parameters such as wrapping, filtering
  - Include texture coordinates as an attribute to vertices
  - Associate a texture *sampler* with each texture map used in a fragment shader
  - For each fragment, retrieve texel values through texture sampler in the fragment shader
  - Use texel value to specify corresponding fragment's color …
- Tutorial will be assigned to go thro' above steps