CSE 167:
Introduction to Computer Graphics
Lecture #9: Texture Mapping

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2020

Announcements

- Sunday, November 1st at 11:59pm:
  - Homework Project 1 late deadline
  - 25% penalty on score
- Sunday, November 8th at 11:59pm:
  - Homework Project 2 due

Lecture Overview

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

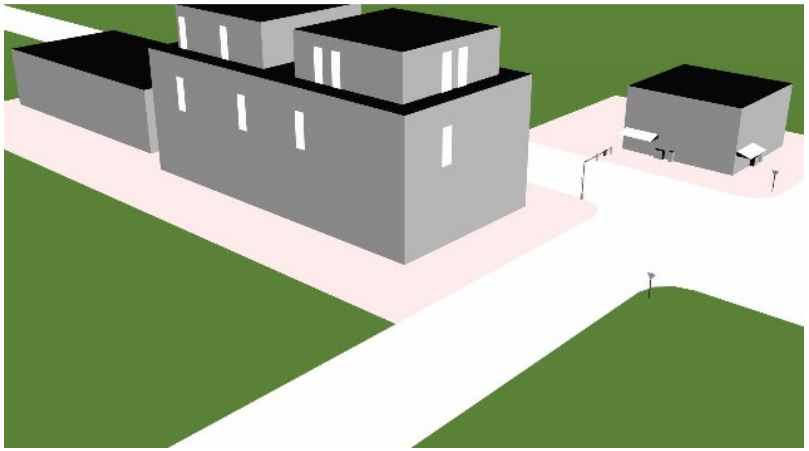
Large Triangles

Pros:

- Often sufficient for simple geometry
- Fast to render

Cons:

- Per vertex colors look boring and computer-generated

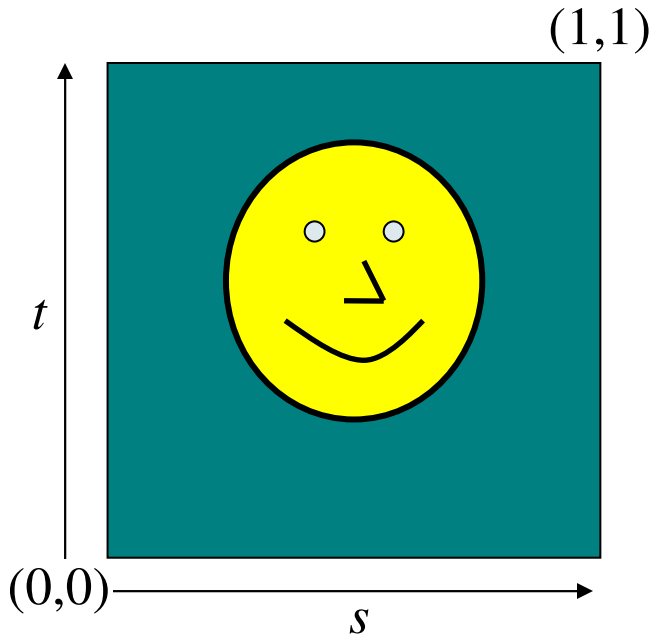


Texture Mapping

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

Texture Mapping

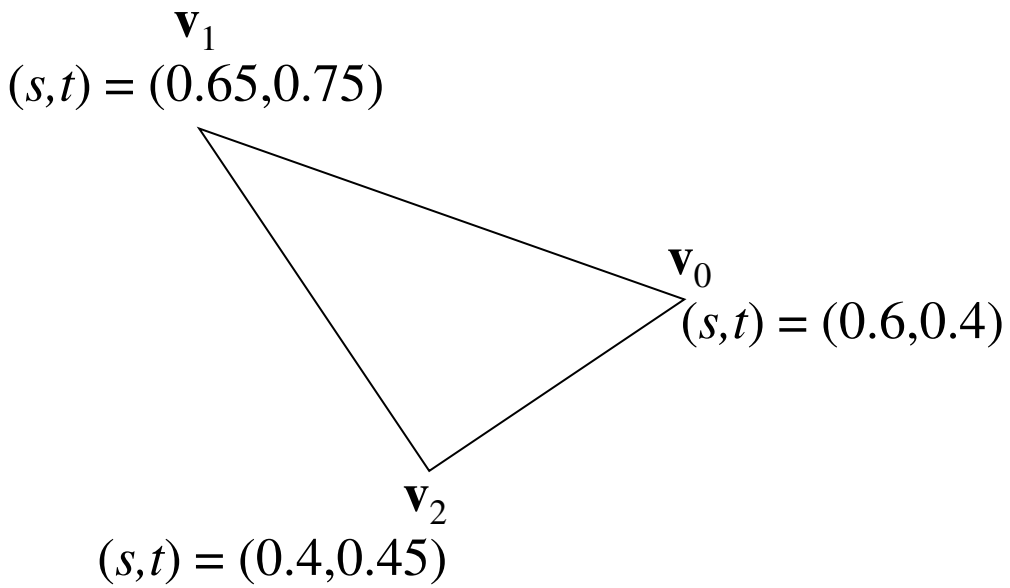
- Goal: map locations in texture to locations on 3D geometry
- Texture coordinate space
  - Texture pixels (texels) have texture coordinates (s,t)
- Convention
  - Bottom left corner of texture is at $(s,t) = (0,0)$
  - Top right corner is at $(s,t) = (1,1)$



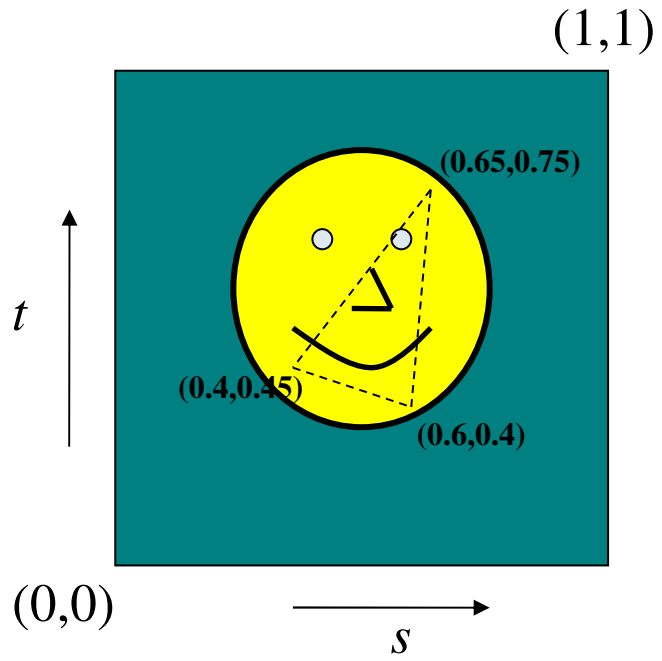
Texture coordinates

Texture Mapping

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



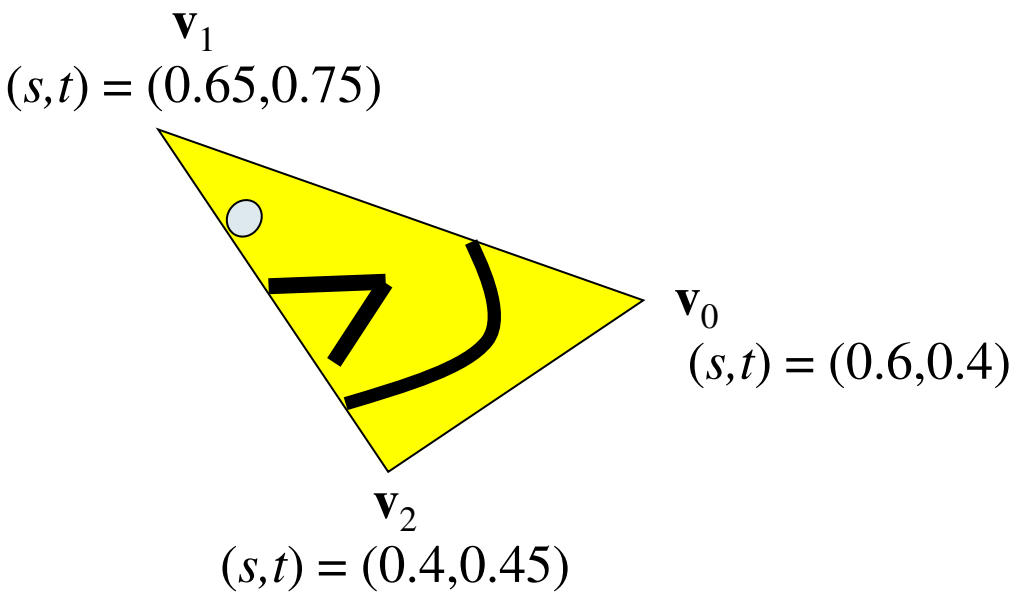
Triangle in any space before projection



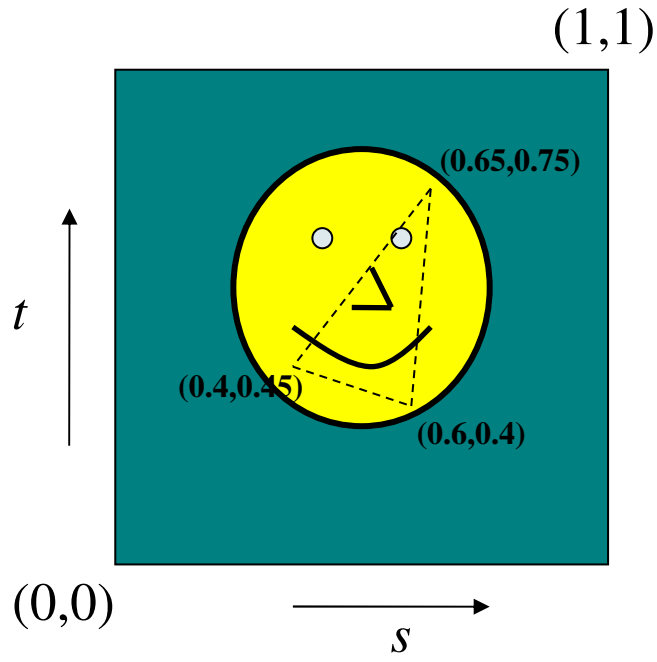
Texture coordinates

Texture Mapping

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

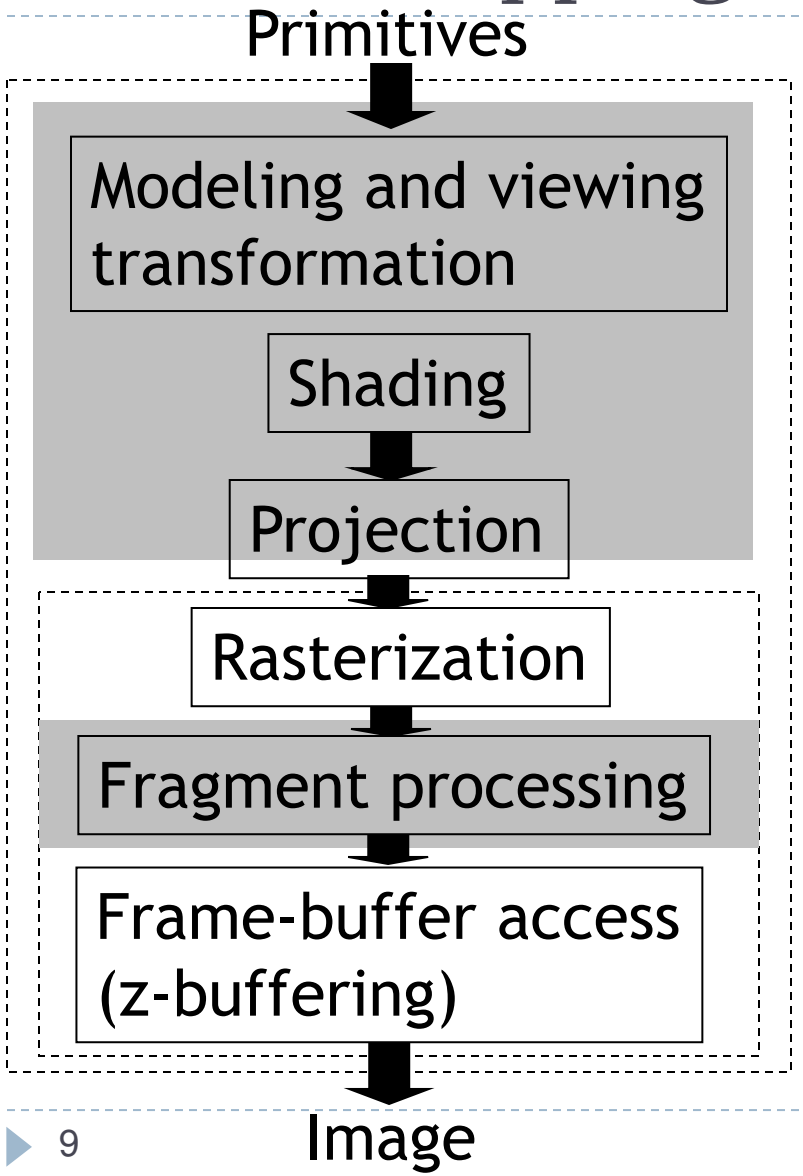


Triangle in any space before projection



Texture coordinates

Texture Mapping

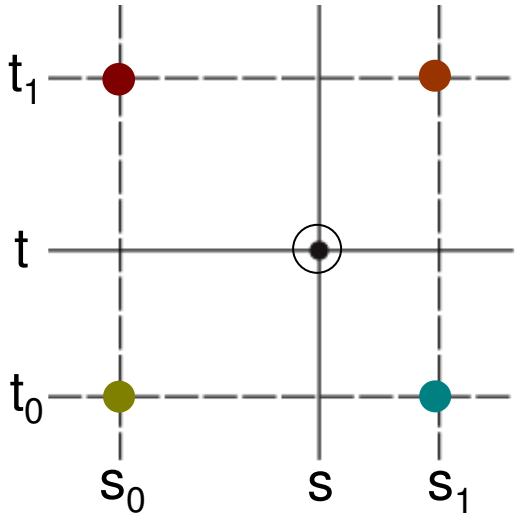


Includes texture mapping

Texture Look-Up

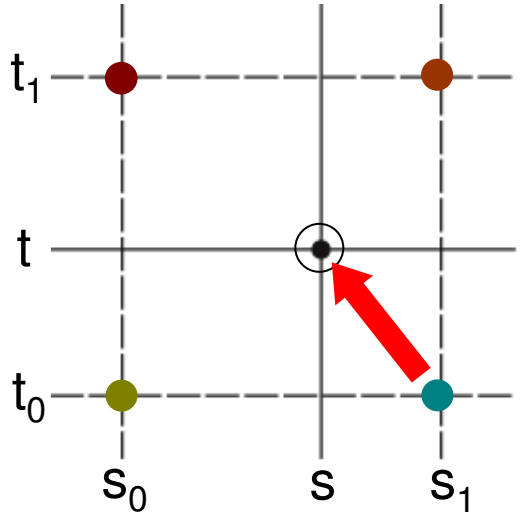
- Given interpolated texture coordinates (s, t) at current pixel
- Closest four texels in texture space are at

  (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

Bilinear Interpolation

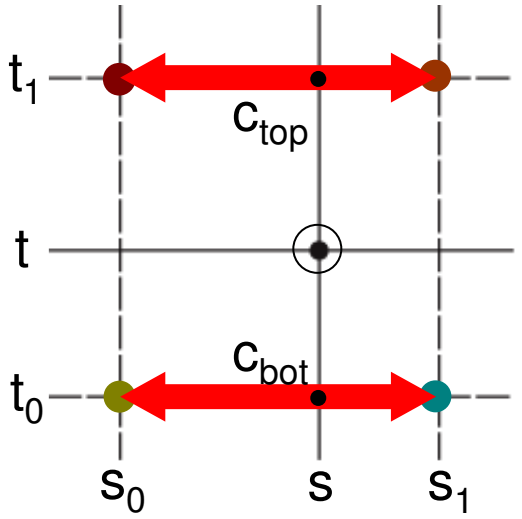
1. Linear interpolation horizontally:

   Ratio in s direction r_s:

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

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

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



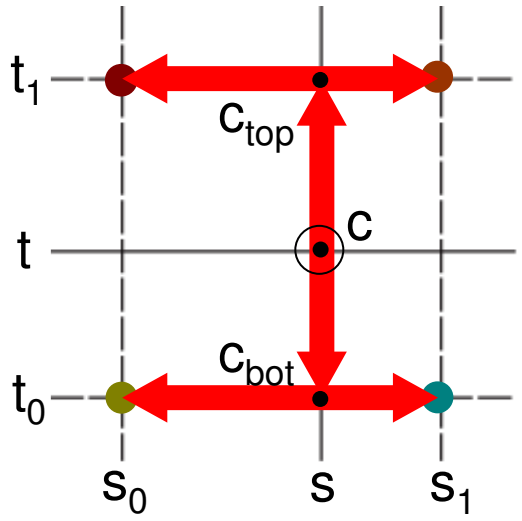
Bilinear Interpolation

2. Linear interpolation vertically

Ratio in t direction r_t:

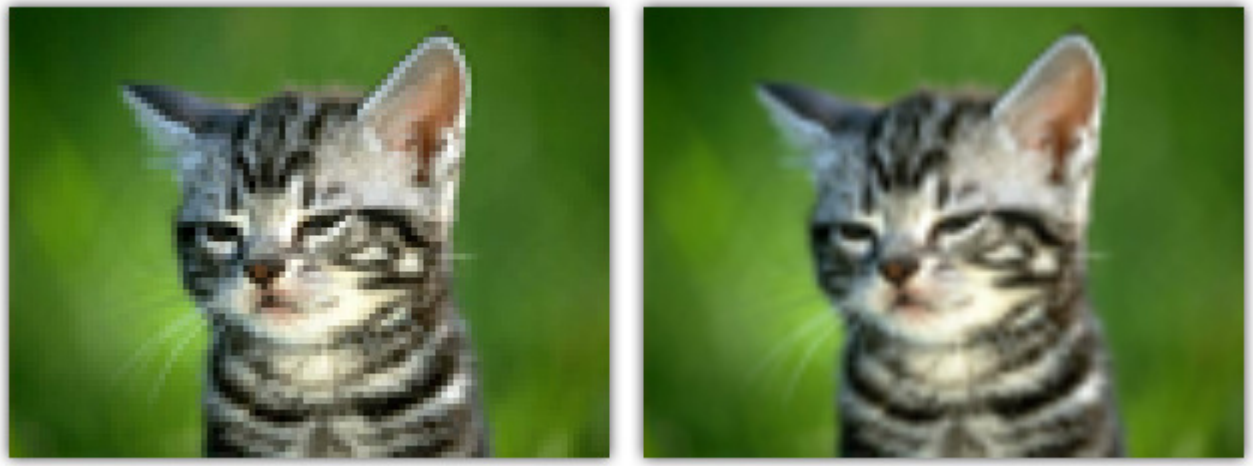
$$r_t = \frac{t - t_0}{t_1 - t_0}$$

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



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);



Source: https://open.gl/textures

OpenGL Example: Loading a Texture

```
// Loads image as texture, returns ID of texture
GLuint loadTexture(Image* image)
{
  GLuint textureId;

  glGenTextures(1, &textureId); // Get unique ID for texture
  glBindTexture(GL_TEXTURE_2D, textureId); // Tell OpenGL which texture to edit
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); // set bi-linear interpolation
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); // for both filtering modes
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE); // set texture edge mode
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

  Image* image = loadJPG("photo.jpg");  // load image from disk; uses third party Image library

  // Depending on the image library, the texture image may have to be flipped vertically

  // Load image into OpenGL texture in GPU memory:
  glTexImage2D(GL_TEXTURE_2D,      // Always GL_TEXTURE_2D for image textures
     0,                            // 0 for now
     GL_RGB,                       // Format OpenGL uses for image without alpha channel
     image->width, image->height,  // Width and height
     0,                            // The border of the image
     GL_RGB,           // GL_RGB, because pixels are stored in RGB format
     GL_UNSIGNED_BYTE, // GL_UNSIGNED_BYTE, because pixels are stored as unsigned numbers
     image->pixels);               // The actual RGB image data

  return textureId; // Return the ID of the texture
}
```

Vertex Shader

```
#version 150

in vec3 vert;
in vec2 vertTexCoord;
out vec2 fragTexCoord;

void main()
{
  // Pass the tex coord straight through to the fragment shader
  fragTexCoord = vertTexCoord;

  gl_Position = vec4(vert, 1);
}
```

Fragment Shader

```
#version 150

uniform sampler2D tex; // this is the texture
in vec2 fragTexCoord;  // these are the texture coordinates
out vec4 finalColor;   // this is the output color of the pixel

void main()
{
  finalColor = texture(tex, fragTexCoord);
}
```