CSE 167: Introduction to Computer Graphics Lecture #6: Lights

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2014

Announcements

- Project 2 due Friday, Oct. 24th
- Midterm Exam Thursday, Oct. 30th

Lecture Overview

- OpenGL Light Sources
- Types of Geometry Shading
- Shading in OpenGL
  - Fixed-Function Shading
  - Programmable Shaders
    - Vertex Programs
    - Fragment Programs
    - GLSL

Light Sources

- Real light sources can have complex properties
  - Geometric area over which light is produced
  - Anisotropy (directionally dependent)
  - Reflective surfaces act as light sources (indirect light)

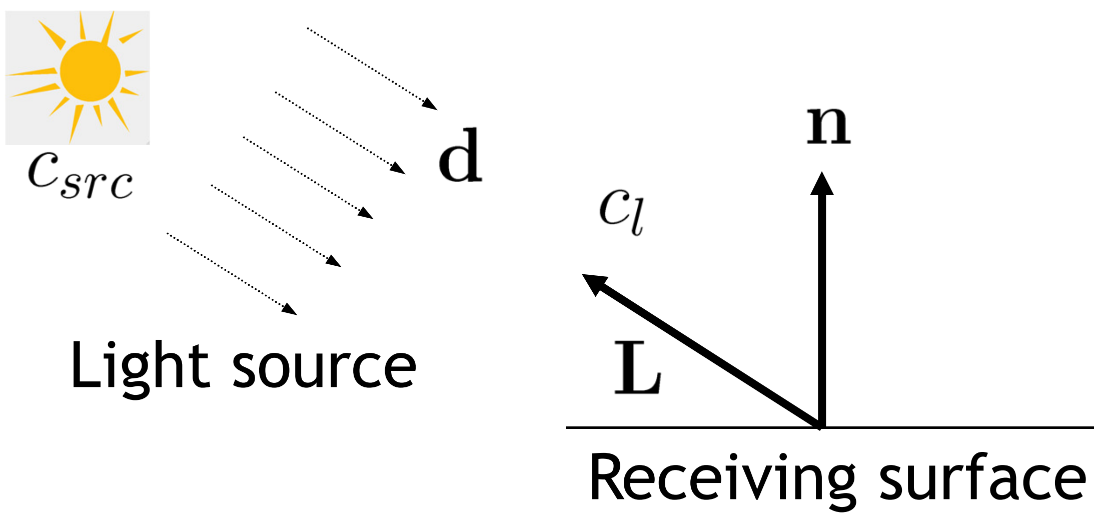
- OpenGL uses a drastically simplified model to allow real-time rendering

OpenGL Light Sources

- At each point on surfaces we need to know
  - Direction of incoming light (the $\mathbf{L}$ vector)
  - Intensity of incoming light (the c_l values)
- Standard light sources in OpenGL
  - Directional: from a specific direction
  - Point light source: from a specific point
  - Spotlight: from a specific point with intensity that depends on direction

Directional Light

- Light from a distant source
  - Light rays are parallel
  - Direction and intensity are the same everywhere
  - As if the source were infinitely far away
  - Good approximation of sunlight
- Specified by a unit length direction vector, and a color



$$\mathbf{L} = -\mathbf{d}$$

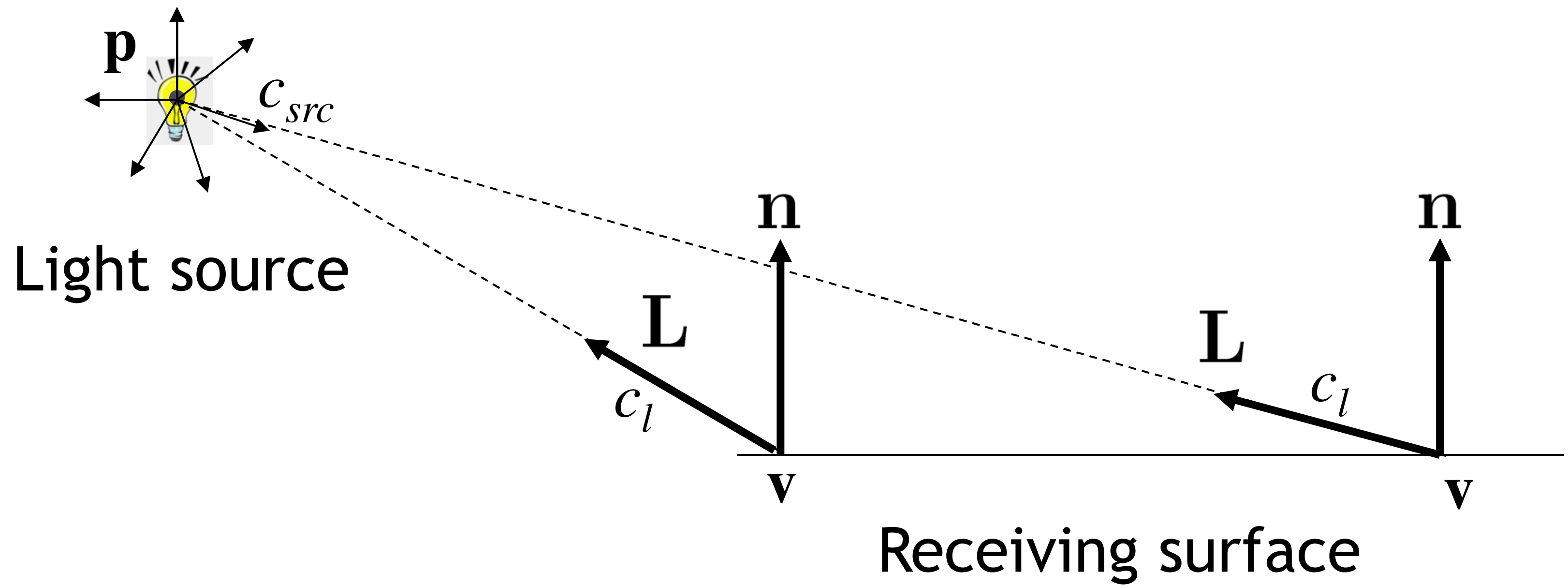
$$c_l = c_{src}$$

Point Lights

- Similar to light bulbs
- Infinitely small point radiates light equally in all directions
  - Light vector varies across receiving surface
  - What is light intensity over distance proportional to?
  - Intensity drops off proportionally to the inverse square of the distance from the light
    - Reason for inverse square falloff: Surface area A of sphere: $A = 4 \pi r^2$

UCSD

Point Lights in Theory



At any point v on the surface:

$$\mathbf{L} = \frac{\mathbf{p} - \mathbf{v}}{\|\mathbf{p} - \mathbf{v}\|}$$

$$c_l = \frac{c_{src}}{\|\mathbf{p} - \mathbf{v}\|^2}$$

UCSD

Point Lights in OpenGL

- OpenGL model for distance attenuation:

$$c_l = \frac{c_{src}}{k_c + k_l\left|\mathbf{p}-\mathbf{v}\right| + k_q\left|\mathbf{p}-\mathbf{v}\right|^2}$$

- Attenuation parameters:
  - k_c = constant attenuation, default: 1
  - k_l = linear attenuation, default: 0
  - k_q = quadratic attenuation, default: 0
- Default: no attenuation: $c_l = c_{src}$
- Change attenuation parameters with:
  - GL_CONSTANT_ATTENUATION
  - GL_LINEAR_ATTENUATION
  - GL_QUADRATIC_ATTENUATION

Lecture Overview

- OpenGL Light Sources
  - Spotlights
- Types of Geometry Shading
- Shading in OpenGL
  - Fixed-Function Shading
  - Programmable Shaders
    - Vertex Programs
    - Fragment Programs
    - GLSL

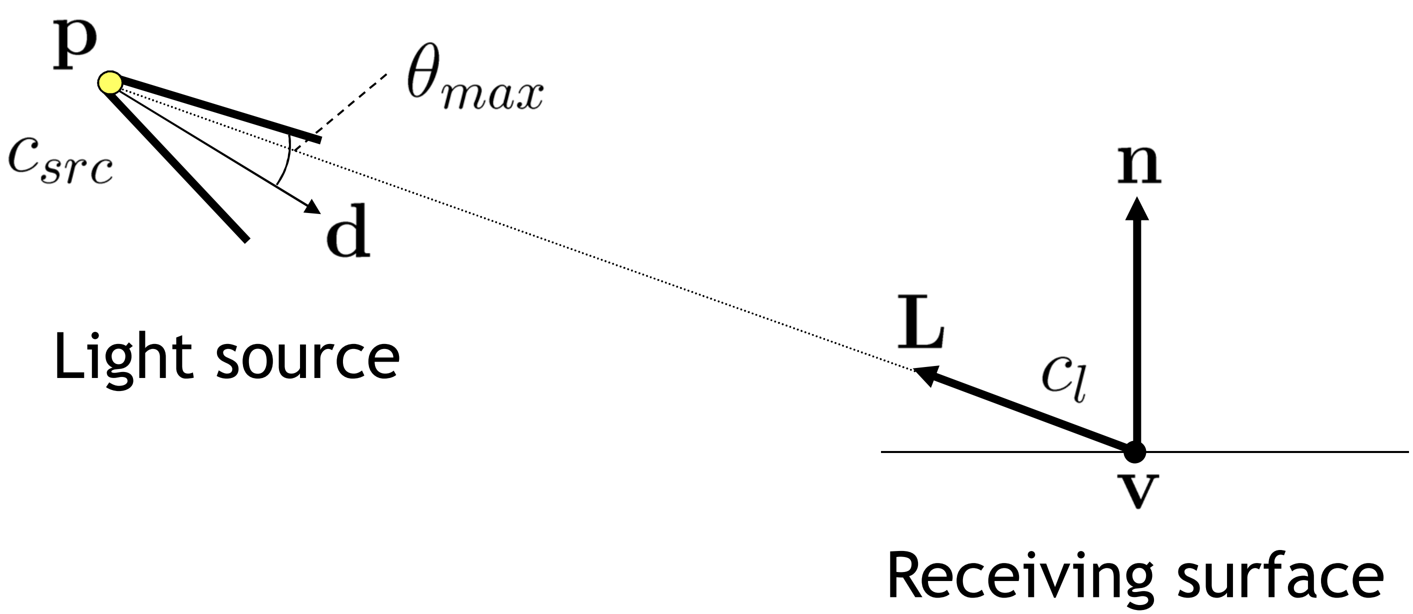
Spotlights

- Like point source, but intensity depends on direction

Parameters

- Position: location of light source
- Spot direction: center axis of light source
- Falloff parameters:
  - Beam width (cone angle)
  - The way the light tapers off at the edges of the beam (cosine exponent)

Spotlights

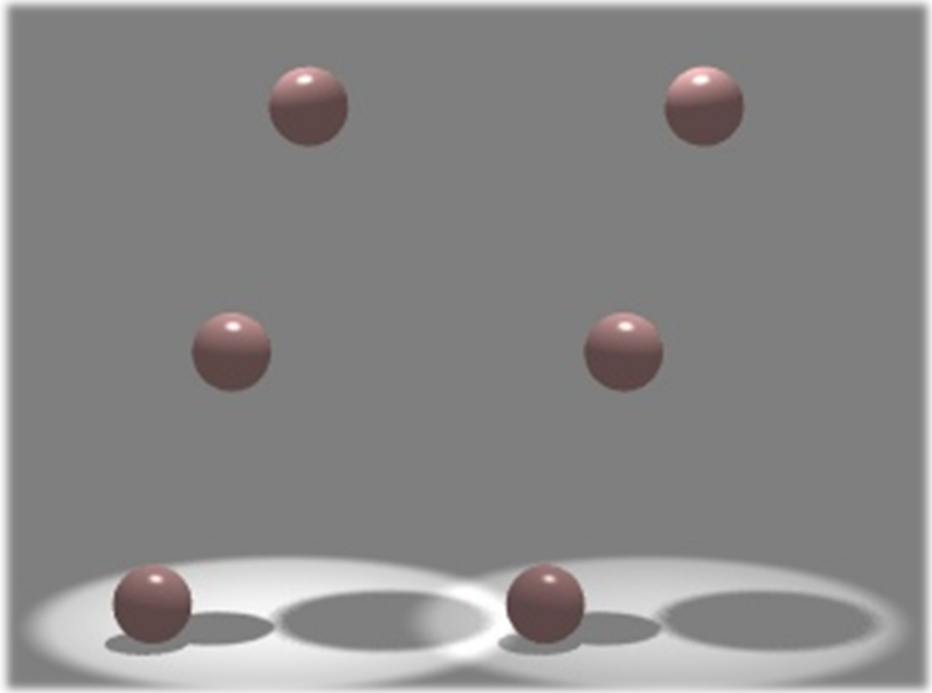


$$\mathbf{L} = \frac{\mathbf{p} - \mathbf{v}}{\|\mathbf{p} - \mathbf{v}\|}$$

$$c_l = \begin{cases} 0 & \text{if} \quad -\mathbf{L} \cdot \mathbf{d} \leq \cos(\theta_{max}) \\ c_{src}\,(-\mathbf{L} \cdot \mathbf{d})^f & \text{otherwise} \end{cases}$$

Spotlights

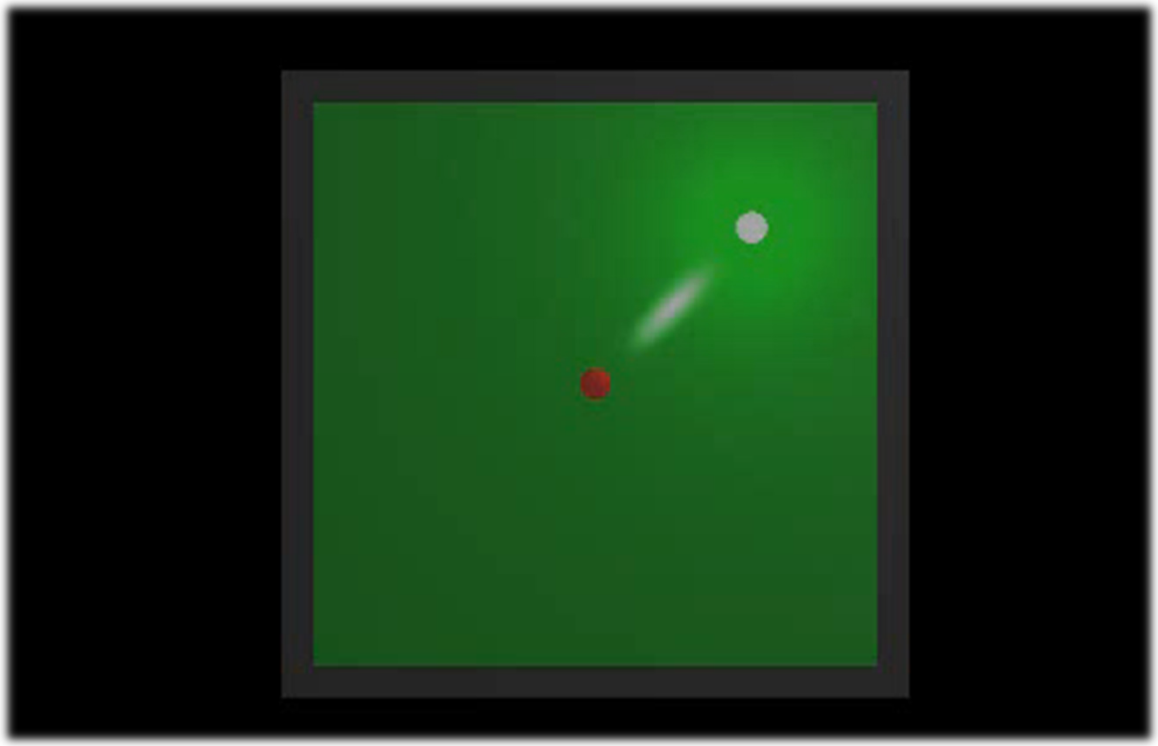
Photograph of real spotlight



Spotlights in OpenGL

Video

- C++ OpenGL Lesson on Basic Lighting
  - http://www.youtube.com/watch?v=g_0yV7jZvGg



UCSD

Lecture Overview

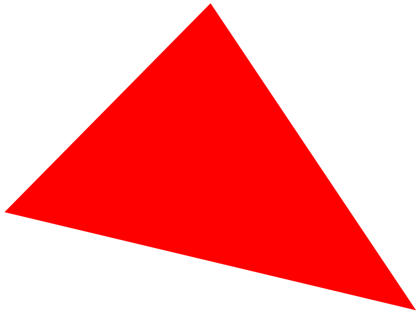
- OpenGL Light Sources
- Types of Geometry Shading
- Shading in OpenGL
  - Fixed-Function Shading
  - Programmable Shaders
    - Vertex Programs
    - Fragment Programs
    - GLSL

Types of Shading

- Per-triangle
- Per-vertex
- Per-pixel

Per-Triangle Shading

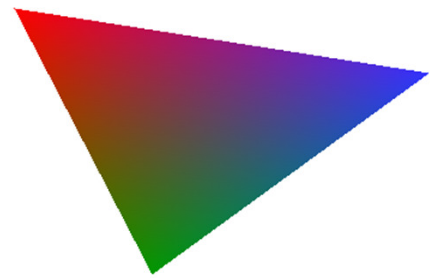
- A.k.a. *flat shading*
- Evaluate shading once per triangle
- Advantage
  - Fast
- Disadvantage
  - Faceted appearance

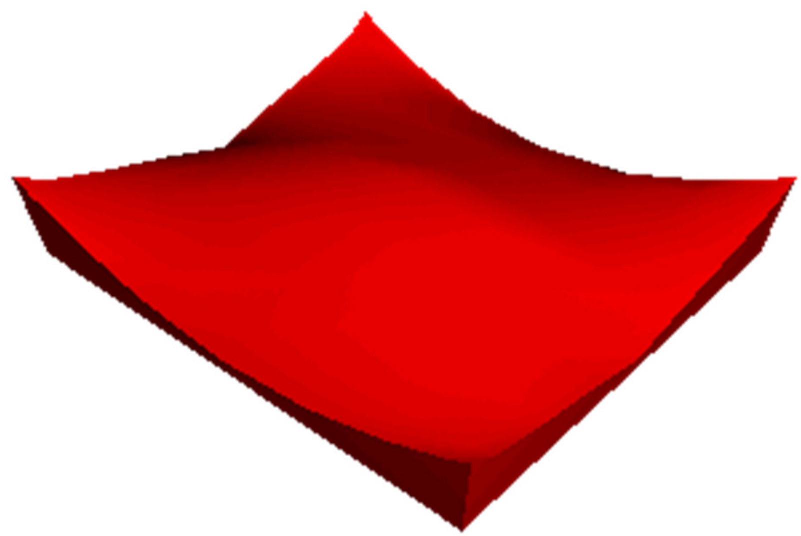


UCSD

Per-Vertex Shading

- Known as *Gouraud shading* (Henri Gouraud, 1971)
- Interpolates vertex colors across triangles
- Advantages
  - Fast
  - Smoother surface appearance than with flat shading
- Disadvantage
  - Problems with small highlights

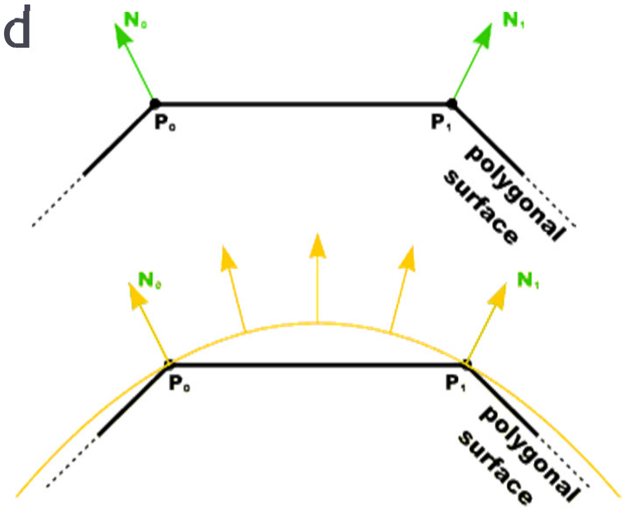




UCSD

Per-Pixel Shading

- A.k.a. *Phong Interpolation* (not to be confused with *Phong Illumination Model*)
  - Rasterizer interpolates <u>normals</u> (instead of colors) across triangles
  - Illumination model is evaluated at each pixel
  - Simulates shading with normals of a curved surface
- Advantage
  - Higher quality than Gouraud shading
- Disadvantage
  - Slow

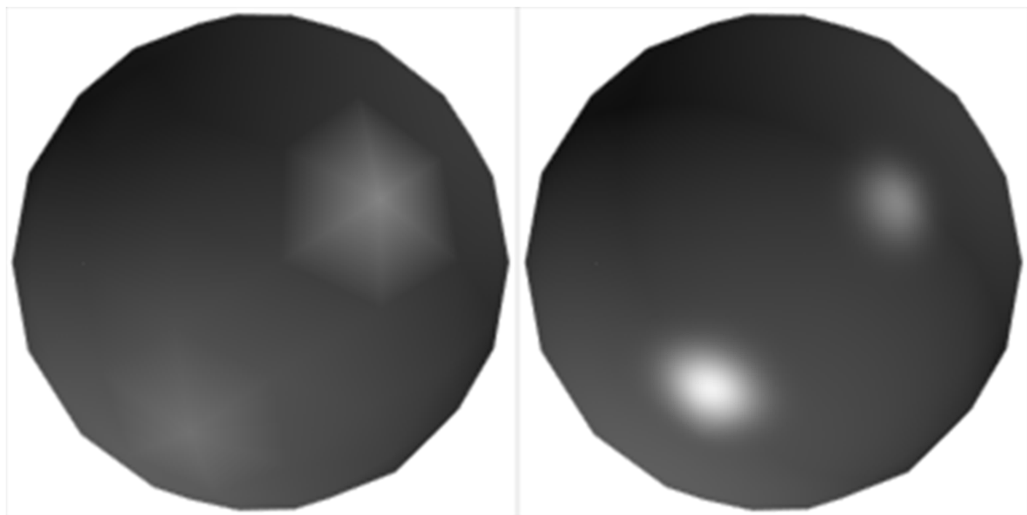


Source: Penny Rheingans, UMBC

UCSD

Gouraud vs. Per-Pixel Shading

- Gouraud shading has problems with highlights when polygons are large
- More triangles improve the result, but reduce frame rate



Gouraud Per-Pixel

Lecture Overview

- OpenGL Light Sources
- Types of Geometry Shading
- Shading in OpenGL
  - Fixed-Function Shading
  - Programmable Shaders
    - Vertex Programs
    - Fragment Programs
    - GLSL

Shading with Fixed-Function Pipeline

- Fixed-function pipeline only allows Gouraud (per-vertex) shading
- We need to provide a normal vector for each vertex
- Shading is performed in camera space
  - Position and direction of light sources are transformed by GL_MODELVIEW matrix
- If light sources should be in object space:
  - Set GL_MODELVIEW to desired object-to-camera transformation
  - Use object space coordinates for light positions
- More information:
  - http://glprogramming.com/red/chapter05.html
  - http://www.falloutsoftware.com/tutorials/gl/gl8.htm

Tips for Transforming Normals

- If you need to (manually) transform geometry by a transformation matrix **M,** which includes shearing or scaling:
  - Transforming the normals with **M** will not work: transformed normals are no longer perpendicular to surfaces
- Solution: transform the normals differently:
  - Either transform the end points of the normal vectors separately
  - Or transform normals with ${\mathbf{M}^{-1}}^T$
- OpenGL does this automatically if the following command is used:
  - glEnable(GL_NORMALIZE)
- More details on-line at:
  - http://www.oocities.com/vmelkon/transformingnormals.html

Lecture Overview

- OpenGL Light Sources
- Types of Geometry Shading
- Shading in OpenGL
  - Fixed-Function Shading
  - Programmable Shaders
    - Vertex Programs
    - Fragment Programs
    - GLSL

Programmable Shaders in OpenGL

- Initially, OpenGL only had a fixed-function pipeline for shading
- Programmers wanted more flexibility, similar to programmable shaders in raytracing software (term "shader" first introduced by Pixar in 1988)
- First shading languages came out in 2002:
  - **Cg** (C for Graphics, created by Nvidia)
  - **HLSL** (High Level Shader Language, created by Microsoft)
- They supported:
  - **Fragment shaders**: allowed per-pixel shading
  - **Vertex shaders**: allowed modification of geometry

Programmable Shaders in OpenGL

- OpenGL 2.0 supported the OpenGL Shading Language (GLSL) in 2003
- **Geometry shaders** were added in OpenGL 3.2
- **Tessellation shaders** were added in OpenGL 4.0
- Programmable shaders allow real-time:
  Shadows, environment mapping, per-pixel lighting, bump mapping, parallax bump mapping, HDR, etc.

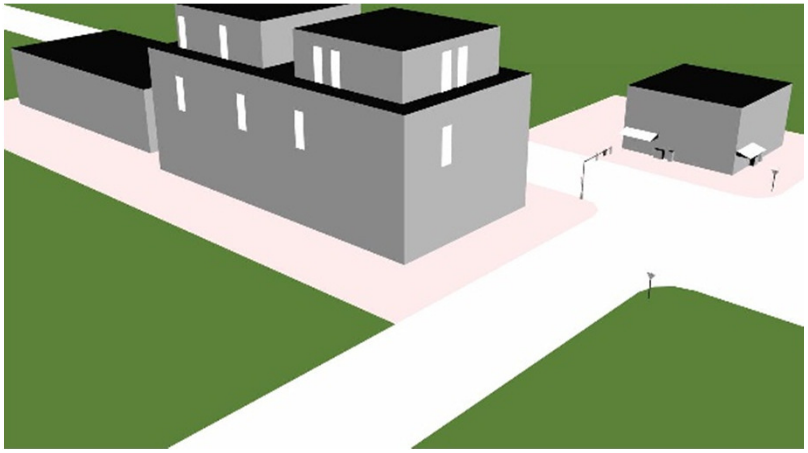
Demo

- NVIDIA Froggy
  - http://www.nvidia.com/coolstuff/demos#!/froggy
  - Bump mapping shader for Froggy's skin
  - Physically-based lighting model simulating sub-surface scattering
  - Supersampling for scene anti-aliasing
  - Raytracing shader for irises to simulate refraction for wet and shiny eyes
  - Dynamically-generated lights and shadows

Lecture Overview

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

Large Triangles



Pros:

- Often sufficient for simple geometry
- Fast to render

Cons:

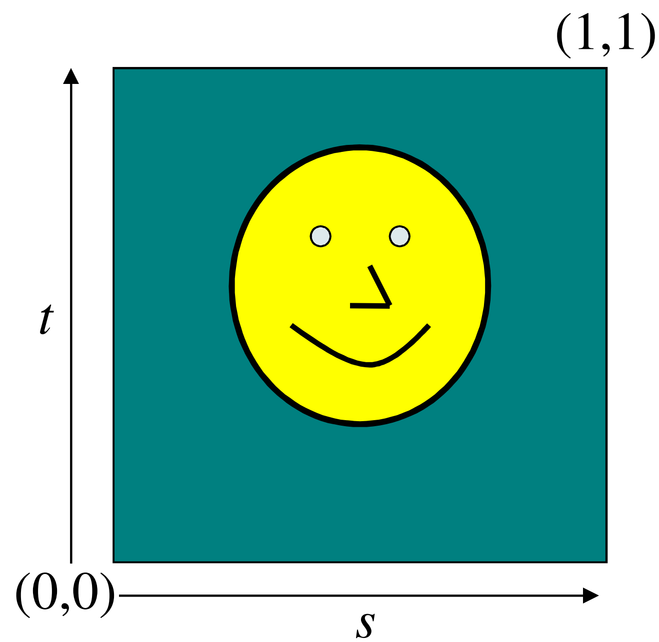
- Per vertex colors look boring and computer-generated

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 $(s,t) = (1,1)$

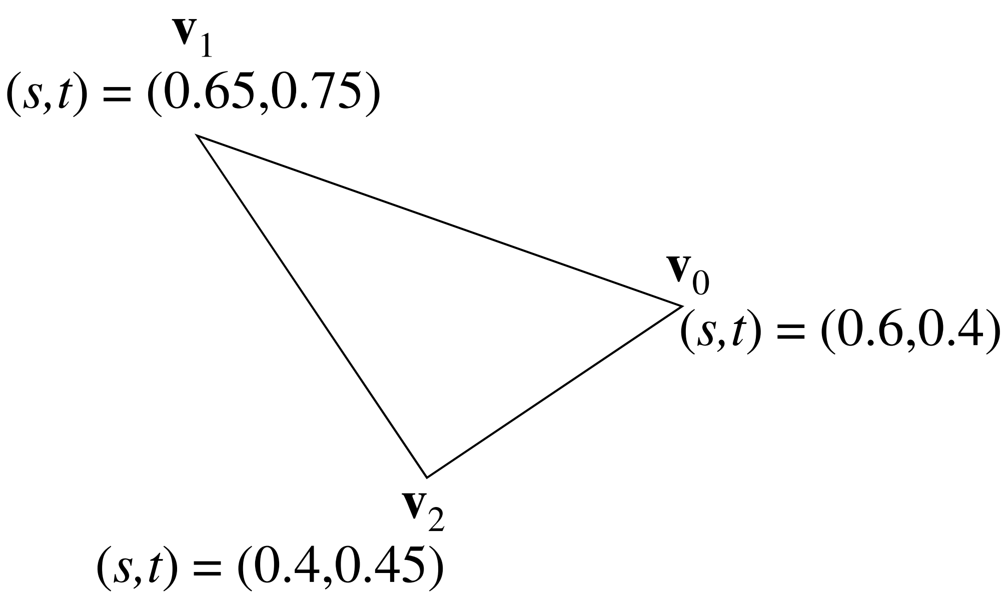


Texture coordinates

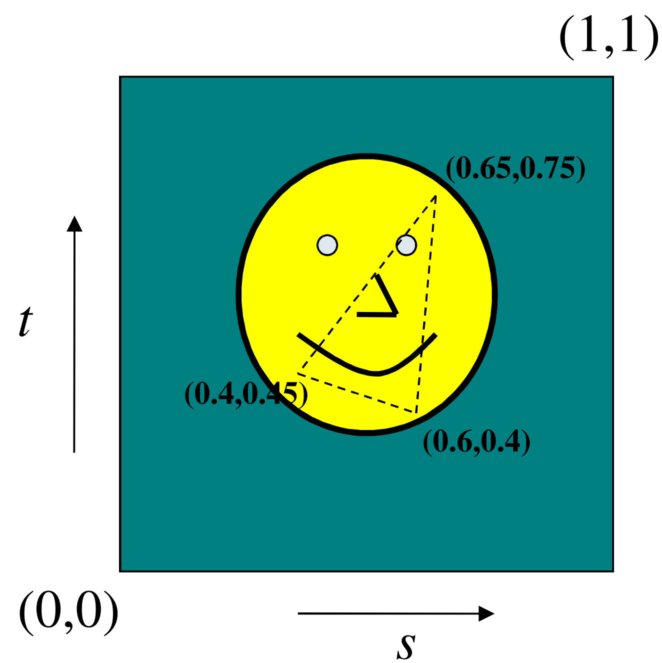
UCSD

Texture Mapping

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



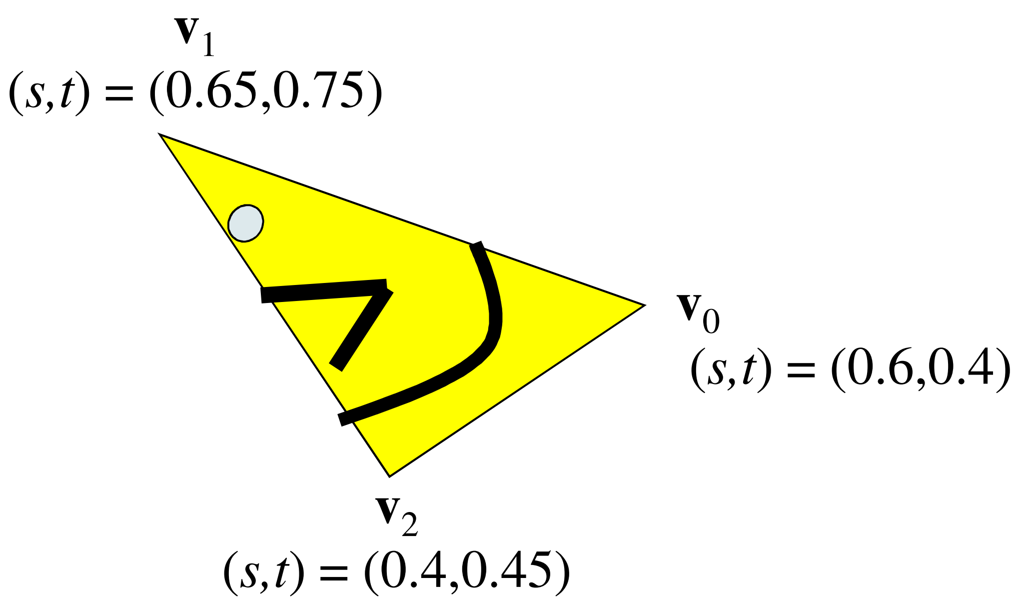
Triangle in any space before projection



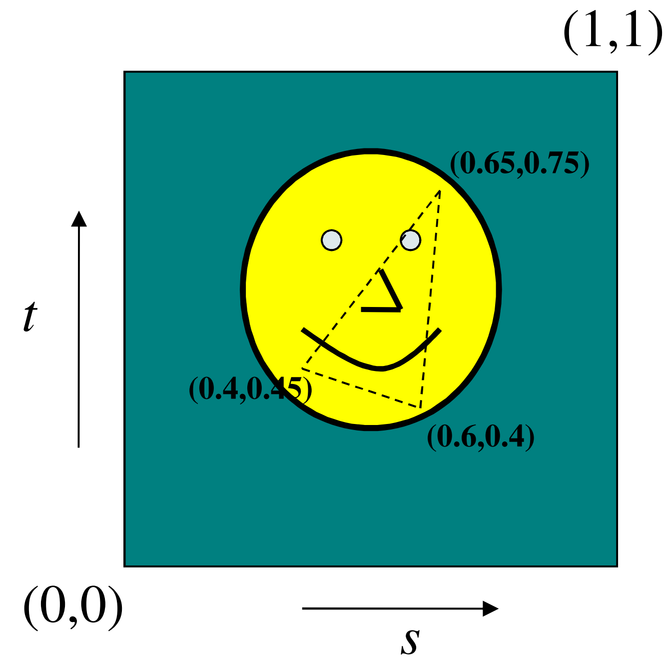
Texture coordinates

Texture Mapping

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



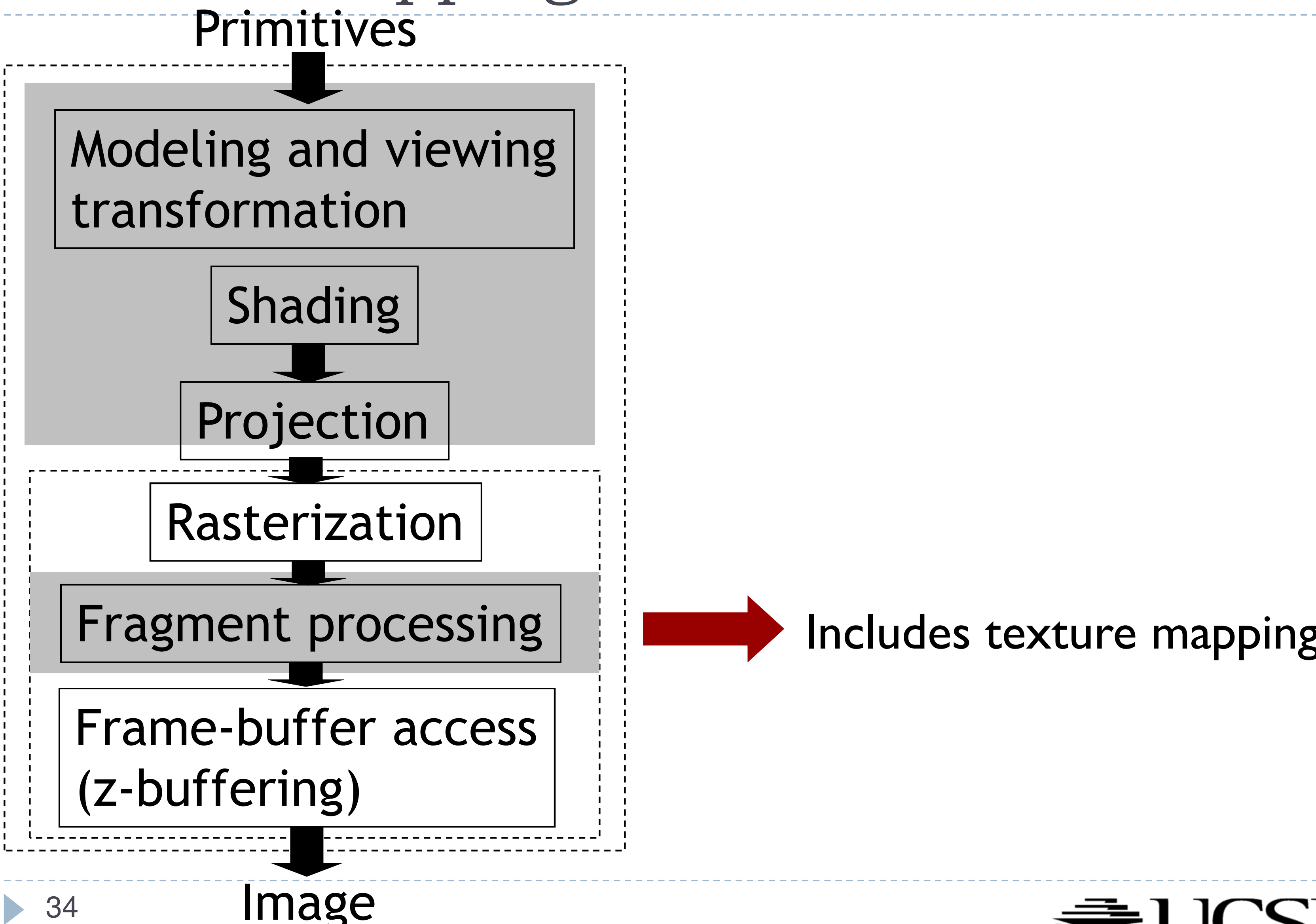
Triangle in any space before projection



Texture coordinates

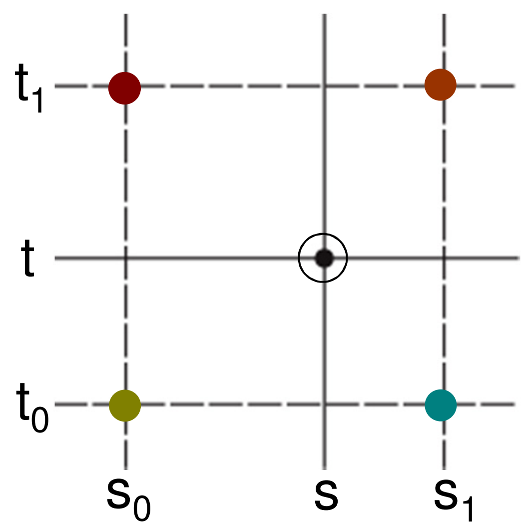
UCSD

Texture Mapping



Texture Look-Up

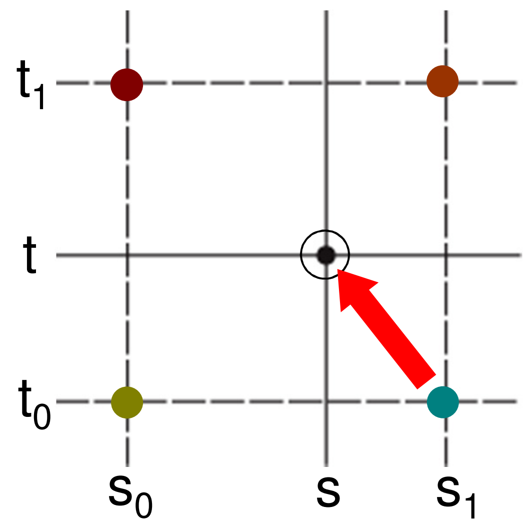
- Given interpolated texture coordinates (s, t) at current pixel
- Closest four texels in texture space are at (s_0,t_0), (s_1,t_0), (s_0,t_1), (s_1,t_1)
- How to compute pixel color?



UCSD

Nearest-Neighbor Interpolation

- Use color of closest texel



- Simple, but low quality and aliasing

UCSD

Bilinear Interpolation

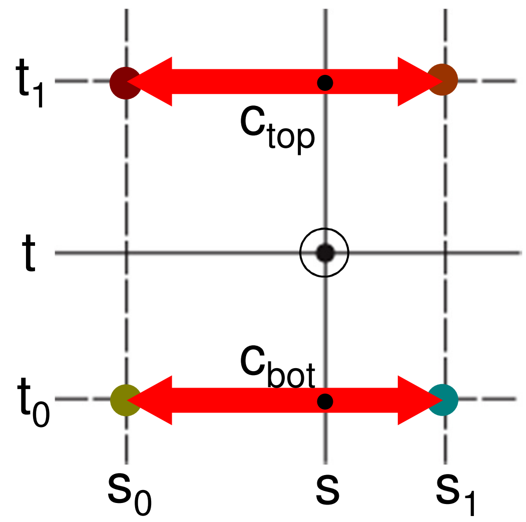
1. Linear interpolation horizontally:

   Ratio in s direction r_s:

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

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

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



UCSD

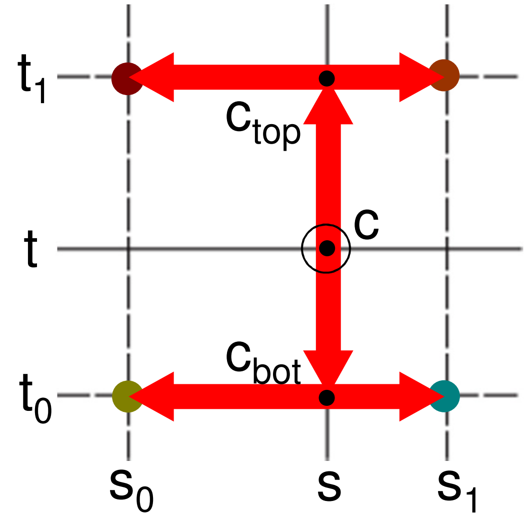
Bilinear Interpolation

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

Lecture Overview

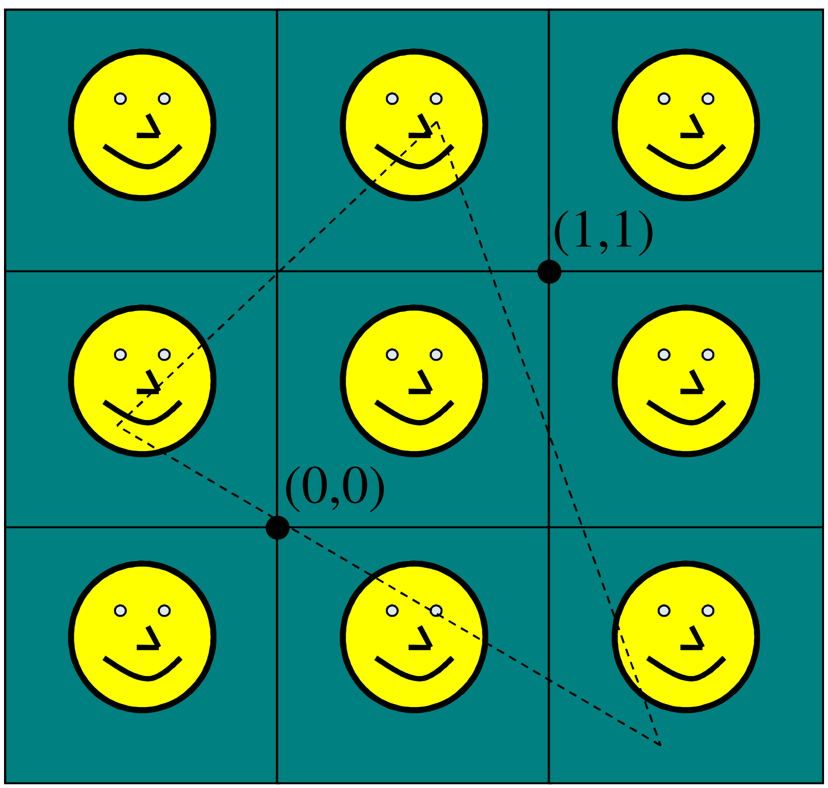
- Texture Mapping
  - Wrapping
  - Texture coordinates
  - Anti-aliasing

Wrap Modes

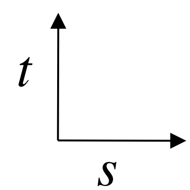
- Texture image extends from $[0,0]$ to $[1,1]$ in texture space
  - What if (s,t) texture coordinates are beyond that range?
- → Texture wrap modes

Repeat

- Repeat the texture
  - Creates discontinuities at edges
    - unless texture designed to line up



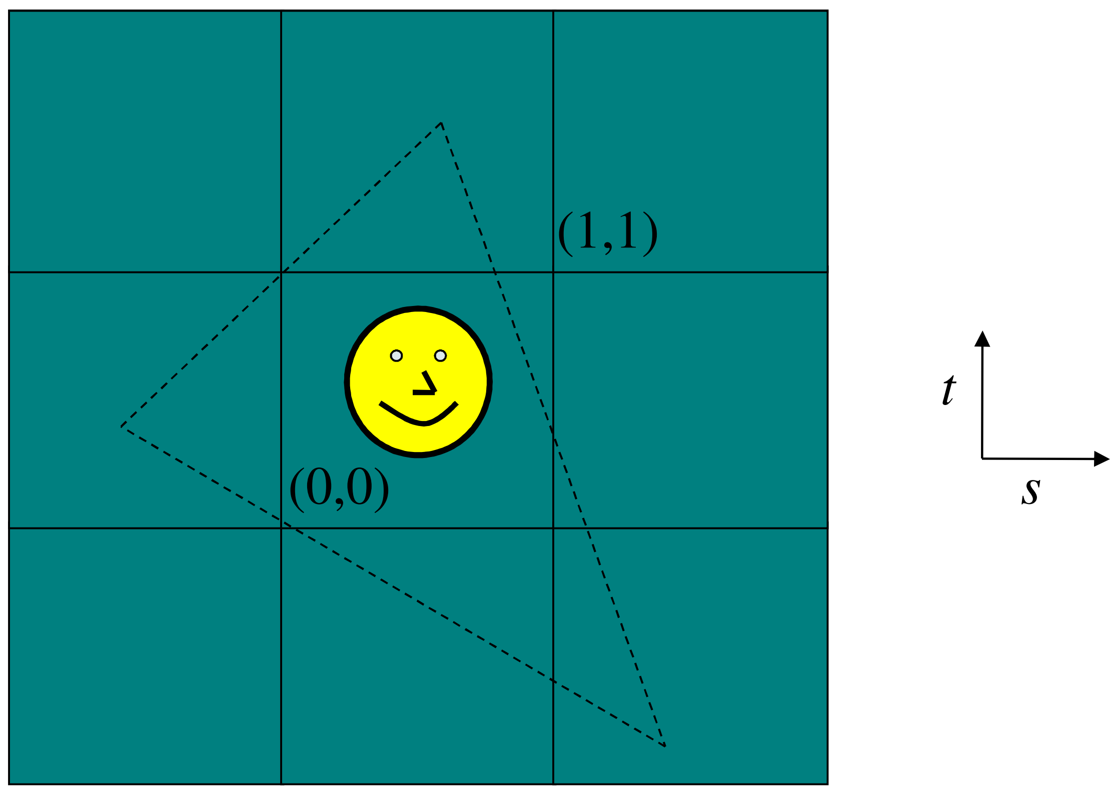
Texture Space



Seamless brick wall texture (by Christopher Revoir)

Clamp

- Use edge value everywhere outside data range [0..1]
- Or, ignore the texture outside [0..1]



Texture Space

UCSD

Wrap Mode Specification 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_CLAMP (requires border to be set)
    GL_CLAMP_TO_EDGE (repeats last pixel in texture),
    GL_REPEAT