CSE 167:
Introduction to Computer Graphics
Lecture #7: Lights

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

Announcements

- Project 2 due next Friday at 2pm
- This Friday: late grading 2-3:15pm
- Midterm exam #1: Tuesday Oct 31, during class

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)

- Need to use simplified model for real-time rendering

Types of Light Sources

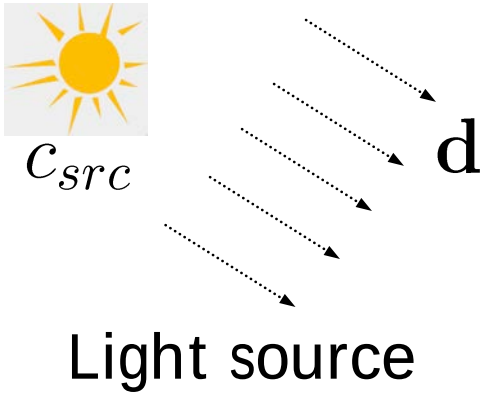
- At each point on surfaces we need to know
  - Direction of incoming light (the $\mathbf{L}$ vector)
  - Intensity of incoming light (the c_l values)
- Three light types:
  - Directional: from a specific direction
  - Point light: from a specific point
  - Spotlight: from a specific point with intensity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

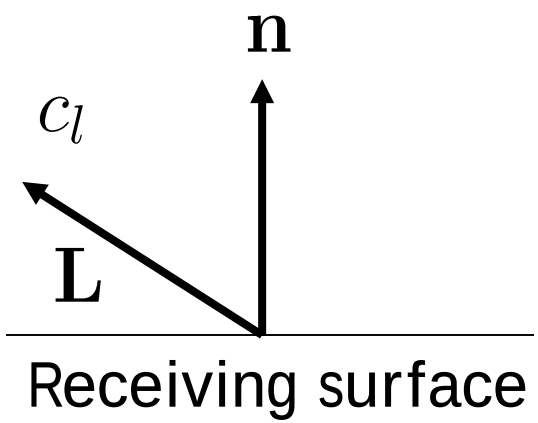
Lecture Overview

- Light Sources
  - Directional Lights
  - Point Lights
  - Spot Lights

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





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

$$c_l = c_{src}$$

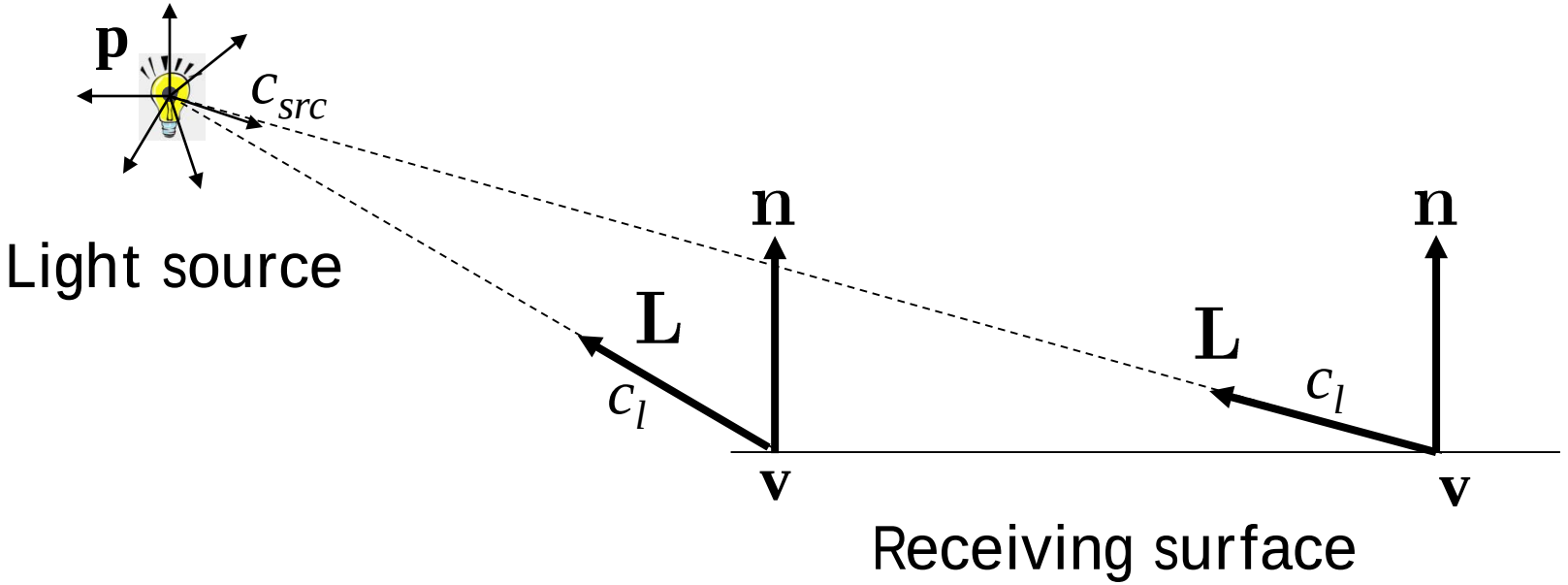
Lecture Overview

- Light Sources
  - Directional Lights
  - Point Lights
  - Spot Lights

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:
      $A = 4 \pi r^2$

Point Light Math



At any point v on the surface:

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

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

Light Attenuation

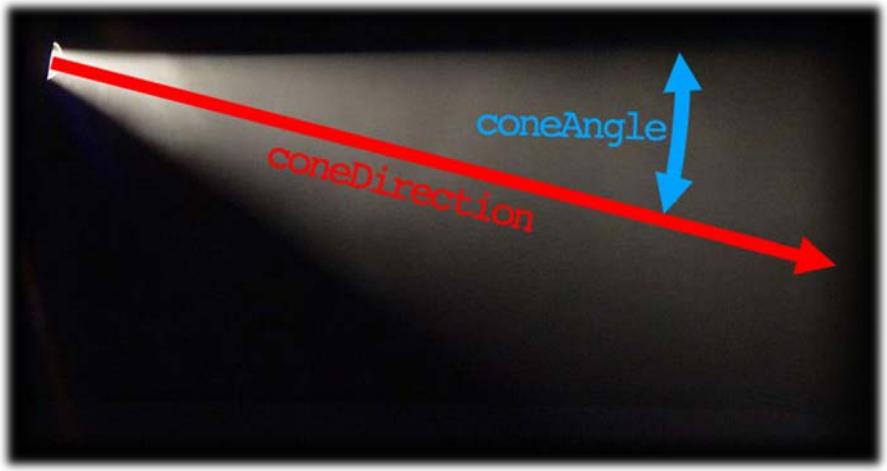
- Adding constant factor k to denominator for better control
- Quadratic attenuation: $k*(p-v)^2$
  - Most computationally expensive, most physically correct
- Linear attenuation: $k*(p-v)$
  - Less expensive, less accurate
- Constant attenuation: k
  - Fastest computation, least accurate

Lecture Overview

- Light Sources
  - Directional Lights
  - Point Lights
  - Spot Lights

Spotlights

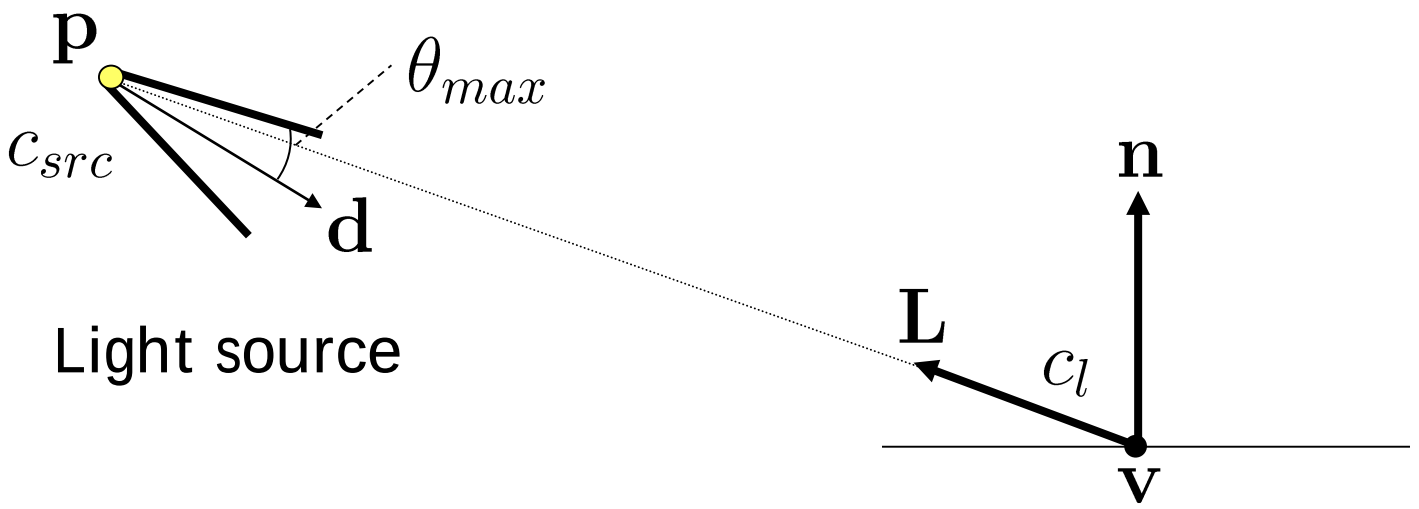
- Like point light, but intensity depends on direction



Parameters

- Position: location of light source
- Cone direction *d*: center axis of light source
- Intensity falloff:
  - Beam width (cone angle θ_{max})
  - The way the light tapers off at the edges of the beam (cosine exponent *f*)

Spotlights



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

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

Vertex Shader

```
#version 150

uniform mat4 camera;
uniform mat4 model;

in vec3 vert;
in vec2 vertTexCoord;
in vec3 vertNormal;

out vec3 fragVert;
out vec2 fragTexCoord;
out vec3 fragNormal;

void main()
{
    // Pass some variables to the fragment shader
    fragTexCoord = vertTexCoord;
    fragNormal = vertNormal;
    fragVert = vert;

    // Apply all matrix transformations to vert
    gl_Position = camera * model * vec4(vert, 1);
}
```


Source: Tom Dallling's OpenGL Tutorial

Fragment Shader for Diffuse Reflection

```
#version 150

uniform mat4 model;
uniform sampler2D tex;

uniform struct Light
{
   vec4 position; // if w component=0 it's directional
   vec3 intensities; // a.k.a the color of the light
   float attenuation; // only needed for point and spotlights
   float ambientCoefficient;
   float coneAngle;  // only needed for spotlights
   vec3 coneDirection; // only needed for spotlights
   float exponent;  // cosine exponent for how light tapers off
} light;

in vec2 fragTexCoord;
in vec3 fragNormal;
in vec3 fragVert;

out vec4 finalColor;
```

Source: Tom Dallling's OpenGL Tutorial

Fragment Shader Part 2

```
void main()
{
    // calculate normal in world coordinates
    mat3 normalMatrix = transpose(inverse(mat3(model)));
    vec3 normal = normalize(normalMatrix * fragNormal);

    // calculate the location of this fragment (pixel) in world coordinates
    vec3 fragPosition = vec3(model * vec4(fragVert, 1));

    // calculate the vector from this pixels surface to the light source
    vec3 surfaceToLight = light.position - fragPosition;

    // calculate the cosine of the angle of incidence
    float brightness = dot(normal, surfaceToLight) / (length(surfaceToLight) * length(normal));
    brightness = clamp(brightness, 0, 1);

    // calculate final color of the pixel, based on:
    // 1. The angle of incidence: brightness
    // 2. The color/intensities of the light: light.intensities
    // 3. The texture and texture coord: texture(tex, fragTexCoord)
    vec4 surfaceColor = texture(tex, fragTexCoord);
    finalColor = vec4(brightness * light.intensities * surfaceColor.rgb, surfaceColor.a);
}
```


Source: Tom Dallling's OpenGL Tutorial