CSE 167:
Introduction to Computer Graphics
Lecture #8: Lighting

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2011

Announcements

- Homework project #4 due Friday, October 28
  - Introduction: Oct 24 at 4pm in lab 250
  - Grading: Oct 28 starting at 1:30pm in lab 260
- Late submissions for project #3 accepted until this Friday
- This Friday, Oct 21: late+early grading starts at 2pm
- Midterm exam: Thursday, Oct 27, 2-3:20pm in class
- Midterm tutorial: Tuesday, Oct 25, 3:45pm-5pm, Atkinson Hall, room 4004
  - We will provide blank index cards

Lecture Overview

- Light Sources
- Shader programming:
  - Vertex shader

Light Sources

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

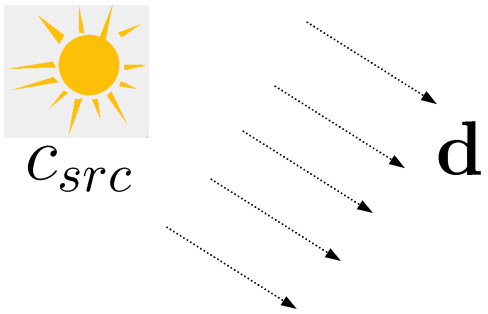
- Interactive rendering is based on simple, standard light sources

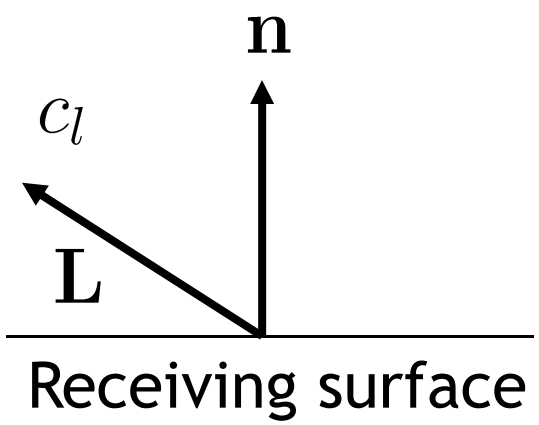
Light Sources

- 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 the 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





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

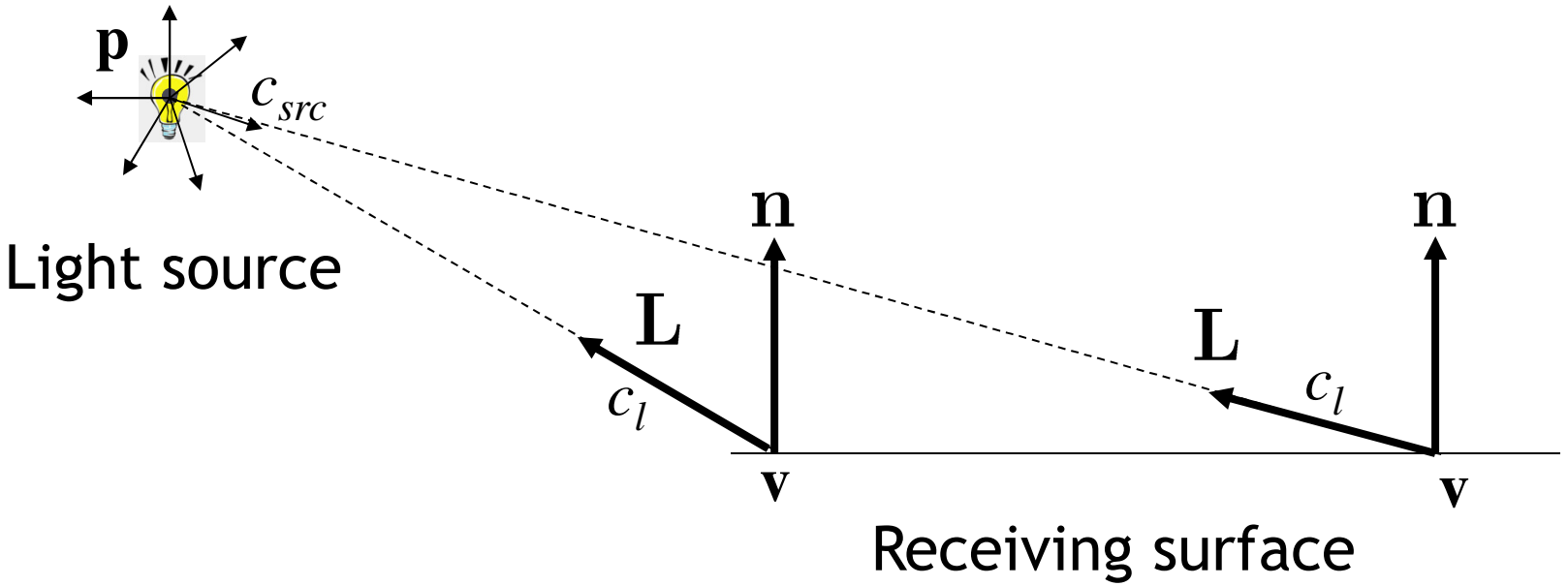
$$c_l = c_{src}$$

Point Lights

- Simple model for light bulbs
- Point that radiates light in all directions equally
  - Light vector varies across the surface
  - 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 Lights



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

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

Attenuation

- Sometimes, it is desirable to modify the inverse square falloff behavior of point lights
  - Common (OpenGL) model for distance attenuation

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

  - Not physically accurate

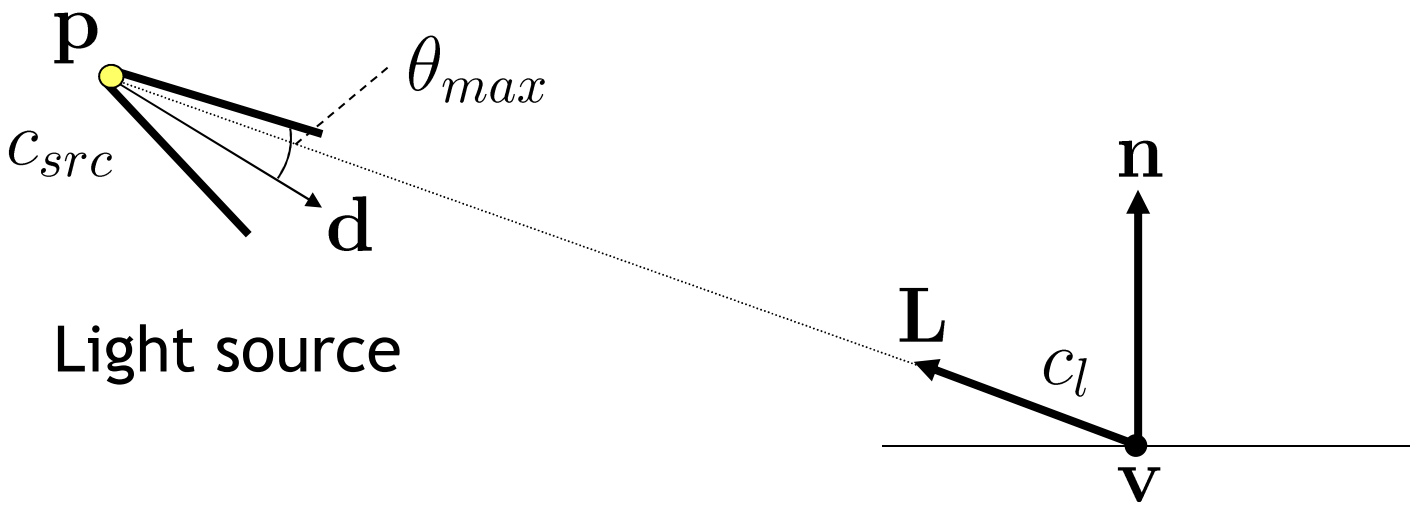
Spotlights

- Like point source, but intensity depends on direction

Parameters

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

Spotlights

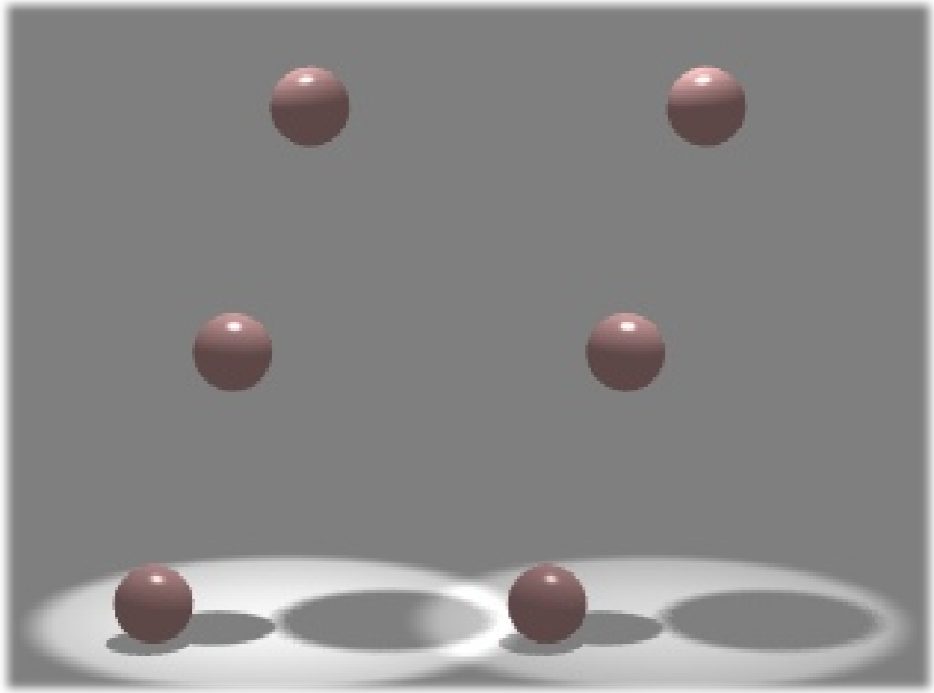


$$\mathbf{L} = \frac{\mathbf{p} - \mathbf{v}}{\|\mathbf{p} - \mathbf{v}\|}$$

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

Spotlights

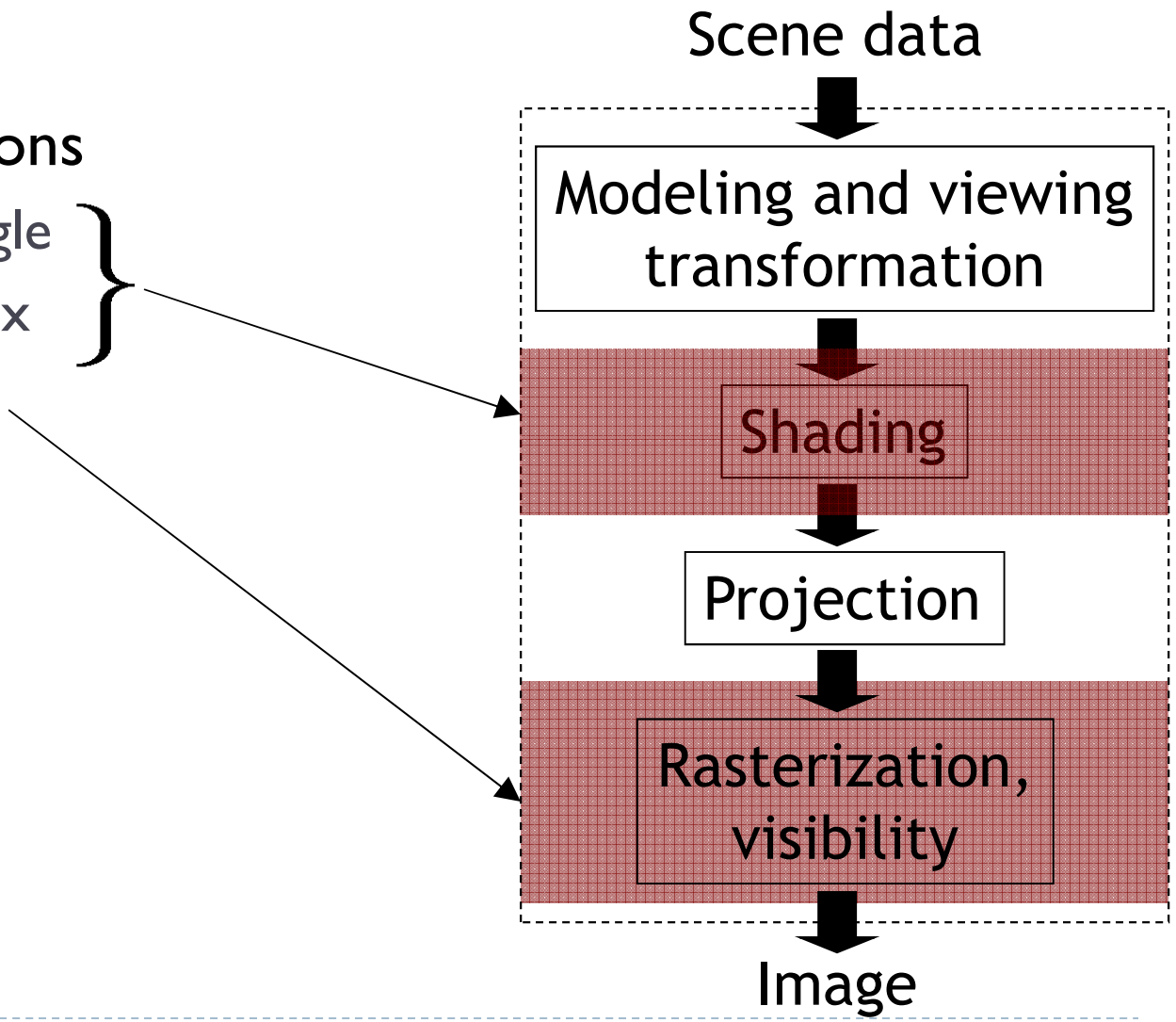
Photograph of spotlight



Spotlights in OpenGL

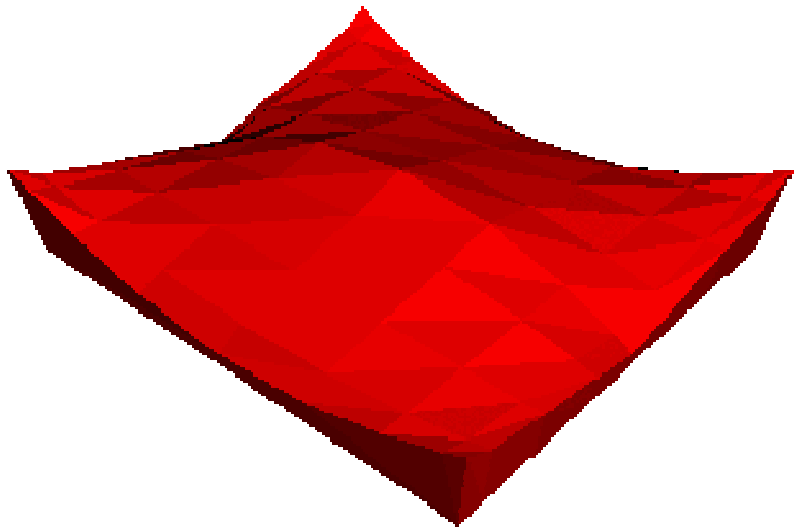
Per-Triangle, -Vertex, -Pixel Shading

- Shading operations
  - Once per triangle
  - Once per vertex
  - Once per pixel



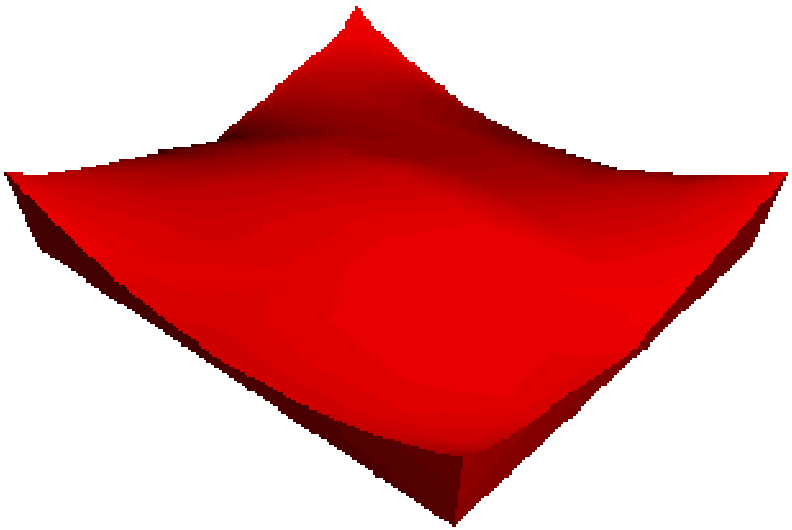
Per-Triangle Shading

- Known as *flat shading*
- Evaluate shading once per triangle
- Advantages
  - Fast
- Disadvantages
  - Faceted appearance



Per-Vertex Shading

- Known as *Gouraud shading* (Henri Gouraud 1971)
- Interpolate vertex colors across triangles
- OpenGL default
- Advantages
  - Fast
  - Smoother than flat shading
- Disadvantages
  - Problems with small highlights

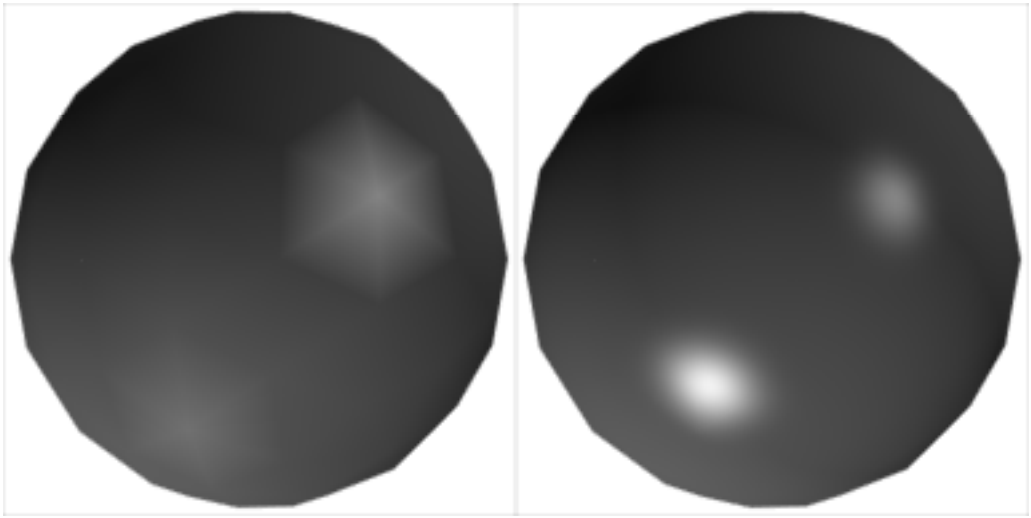


Per-Pixel Shading

- Also known as *Phong interpolation* (not to be confused with *Phong illumination* model)
  - Rasterizer interpolates normals across triangles
  - Illumination model evaluated at each pixel
  - Implemented using *fragment shaders* (later today)
- Advantages
  - Higher quality than Gouraud shading
- Disadvantages
  - Much slower

Gouraud vs. Per-Pixel Shading

- Gouraud has problems with highlights
- More triangles would improve result, but impact frame rate



Gouraud Per-pixel

Shading in OpenGL

```
// Somewhere in the initialization part of your
// program…
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

// Make sure vertex colors are used as material properties
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glColorMaterial(GL_FRONT, GL_SPECULAR);

// Create light components
Glfloat ambientLight[] = { 0.2f, 0.2f, 0.2f, 1.0f };
Glgloat diffuseLight[] = { 0.8f, 0.8f, 0.8, 1.0f };
Glfloat specularLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
Glfloat position[] = { -1.5f, 1.0f, -4.0f, 1.0f };

// Assign created components to GL_LIGHT0
glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
glLightfv(GL_LIGHT0, GL_POSITION, position);
```

Shading in OpenGL

- Shading computations (diffuse, specular, ambient) are performed automatically (unless you use shader programs)

Shading in OpenGL

- Need to provide per vertex normals
- Shading is performed in camera space
  - Position, direction of light sources is transformed by GL_MODELVIEW matrix
- If light sources should be fixed relative to objects
  - Set GL_MODELVIEW to desired object-to-camera transform
  - Choose object space coordinates for light position
  - Will be transformed using current GL_MODELVIEW
- Lots of details, highly recommend OpenGL programming guide
  - http://glprogramming.com/red/chapter05.html
  - http://www.falloutsoftware.com/tutorials/gl/gl8.htm

Transforming Normals

- If the object-to-camera transformation **M** includes shearing or scaling, transforming normals using **M** does not work:
  - Transformed normals are not perpendicular to surfaces any more
- To avoid the problem, we need to transform the normals differently:
  - by transforming the end points of the normal vectors separately
  - or using $\mathbf{M}^{-1T}$
- Find derivation on-line at:
  - http://www.oocities.com/vmelkon/transformingnormals.html
- OpenGL does this automatically for us on the GPU

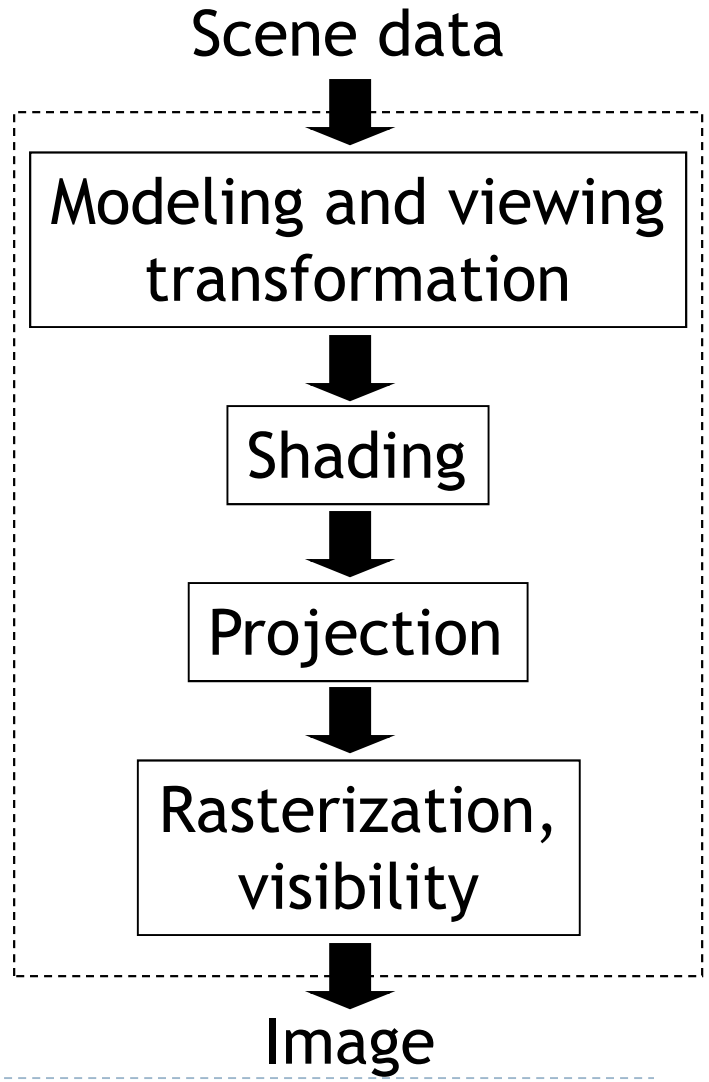
Lecture Overview

- Light Sources
- Shader programming:
  - Vertex shader

Configurable Pipeline

Before programmable shaders:

- APIs (OpenGL, Direct3D) to configure the rendering pipeline
- Enable/disable functionality
  - E.g., lighting, texturing
- Set parameters for given functionality
  - E.g., light direction, texture blending mode



Configurable Pipeline

Disadvantages

- Restricted to preset functionality
  - Limited types of light sources (directional, point, spot)
  - Limited set of reflection models (ambient, diffuse, Phong)
  - Limited use of texture maps
- More flexibility desired for more photorealistic effects

Demo

- NVIDIA Time Machine
- http://www.nvidia.com/object/cool_stuff.html#/demos/256