Computer Graphics

CSC 418/2504

Patricio Simari

November 2, 2011

Figures courtesy of Peter Shirley,
"Fundamentals of Computer Graphics", 2nd Ed.

Topics

- Surface shading
- Raytracing: shadows, reflection
- Texture mapping

Lambertian surfaces

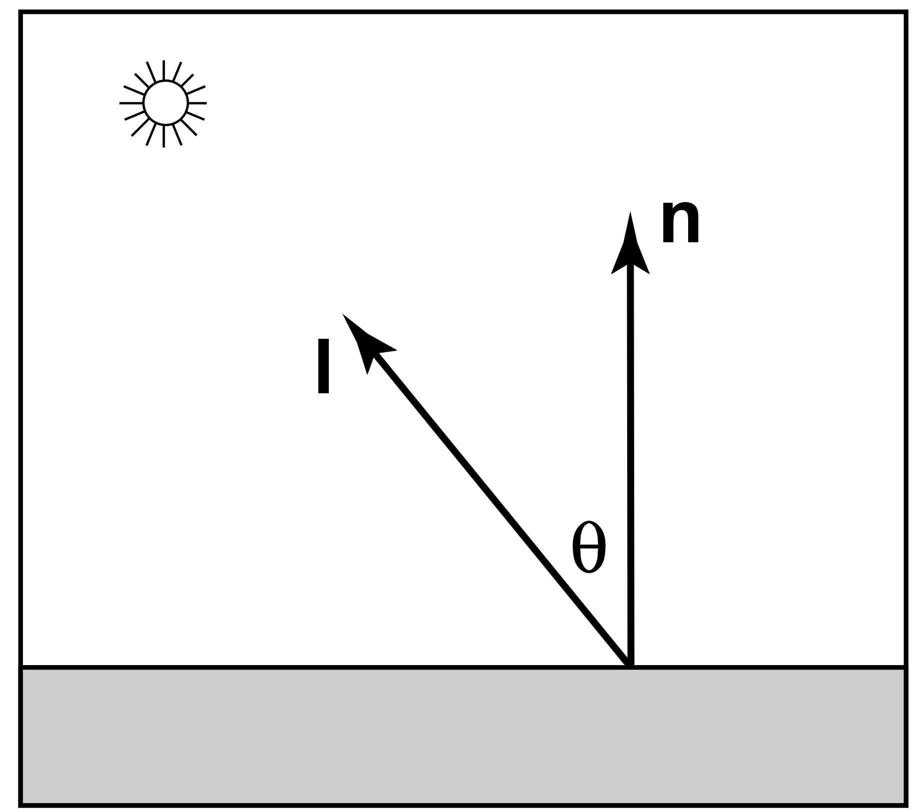
- Matte surfaces: paper, unfinished wood, dry unpolished stones...
- Shading depends on lighting direction, not viewing direction

Lambertian surfaces

Lambert cosine law

$$c \propto \cos\theta$$

$$c \propto \mathbf{n} \cdot \mathbf{l}$$

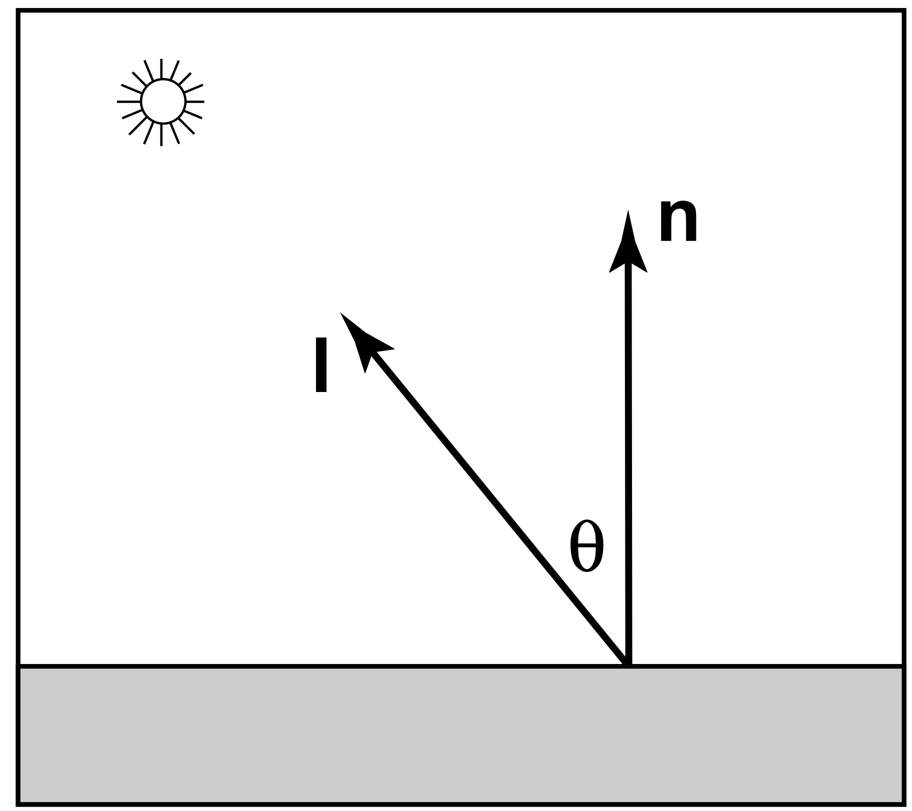


directional light

Lambertian surfaces

Diffuse reflectance: c_r

$$c \propto c_r \, \mathbf{n} \cdot \mathbf{l}$$

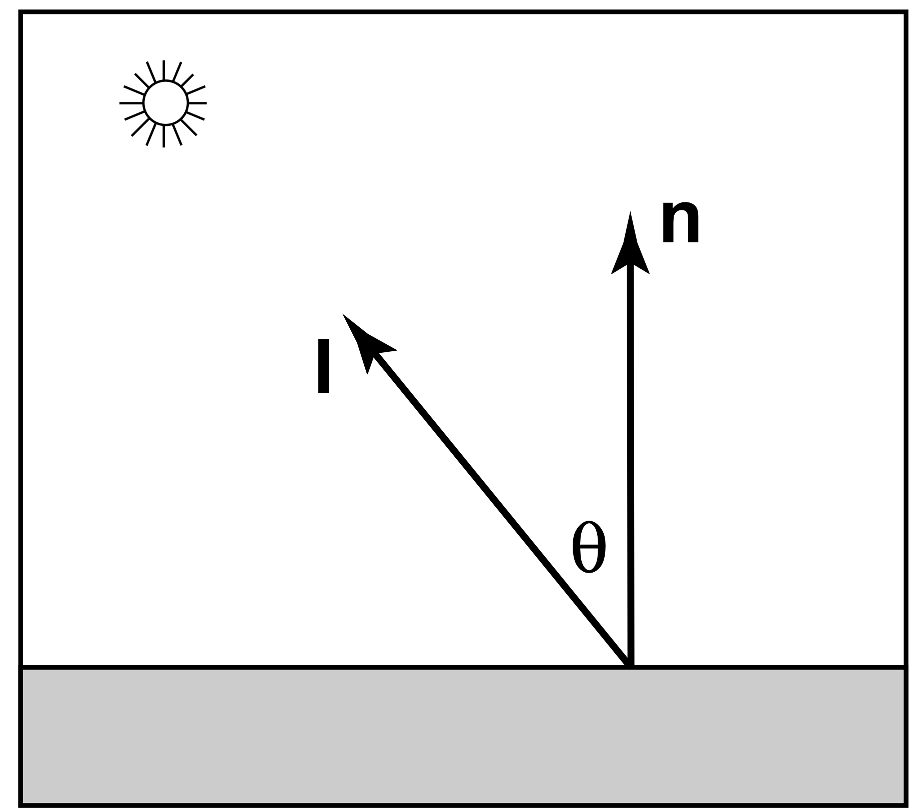


directional light

Lambertian surfaces

Lighting intensity: c_l

$$c = c_r \, c_l \, \mathbf{n} \cdot \mathbf{l}$$

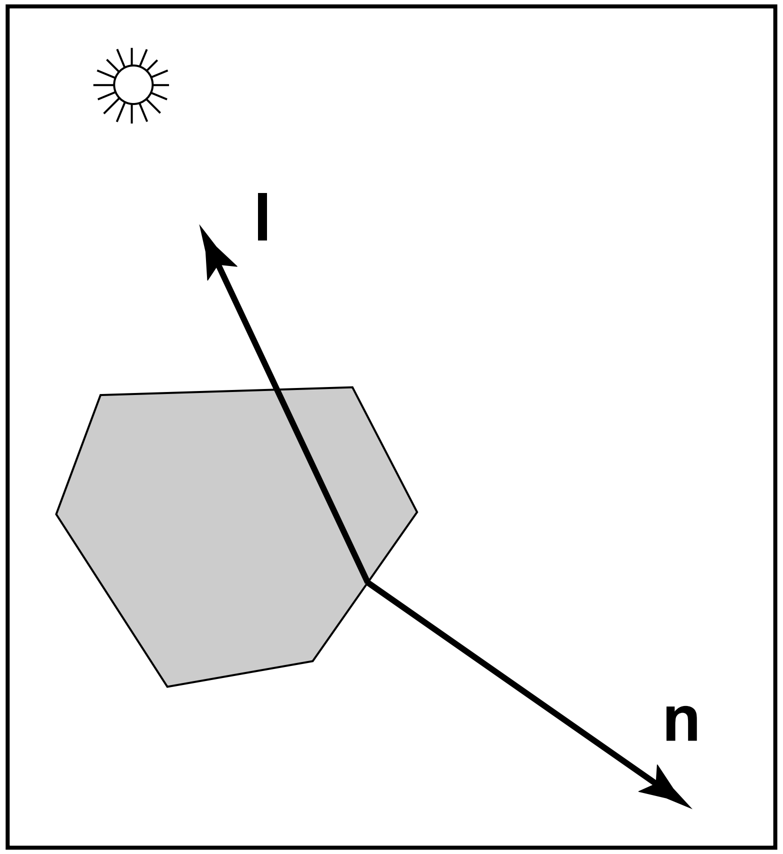


directional light

Lambertian surfaces

Lower clamp to 0

$c = c_r\, c_l \max(0, \mathbf{n} \cdot \mathbf{l})$



normals facing away should not be lit

Lambertian surfaces

Ambient

$$c = c_r \, (c_a + c_l \max(0, \mathbf{n} \cdot \mathbf{l}))$$

Vertex-based diffuse shading

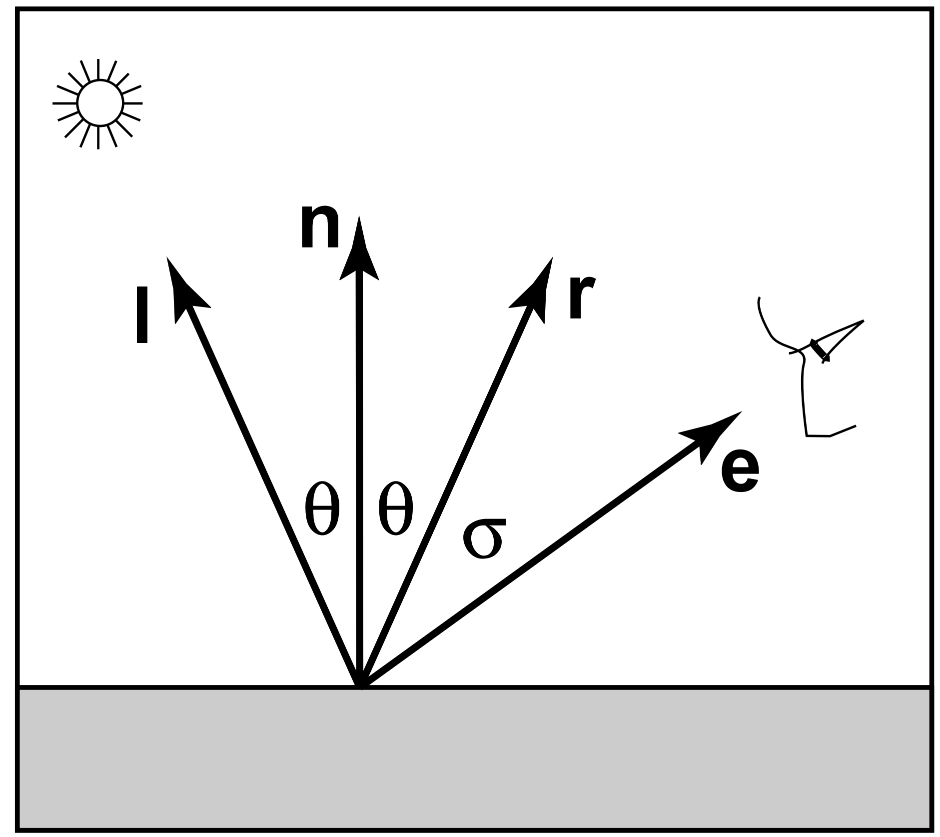
- Flat shading: single color for face
- Gouraud shading: compute value at vertices, barycentric interpolation across faces
- Estimate normals at vertices: average of normals of incident faces (different strategies)

Phong shading

- Matte surfaces + *highlights*: polished tile floors, gloss paint, white boards...
- Highlights depend on light **and** viewing direction

Phong shading

$c = c_l\,(\mathbf{e} \cdot \mathbf{r})$

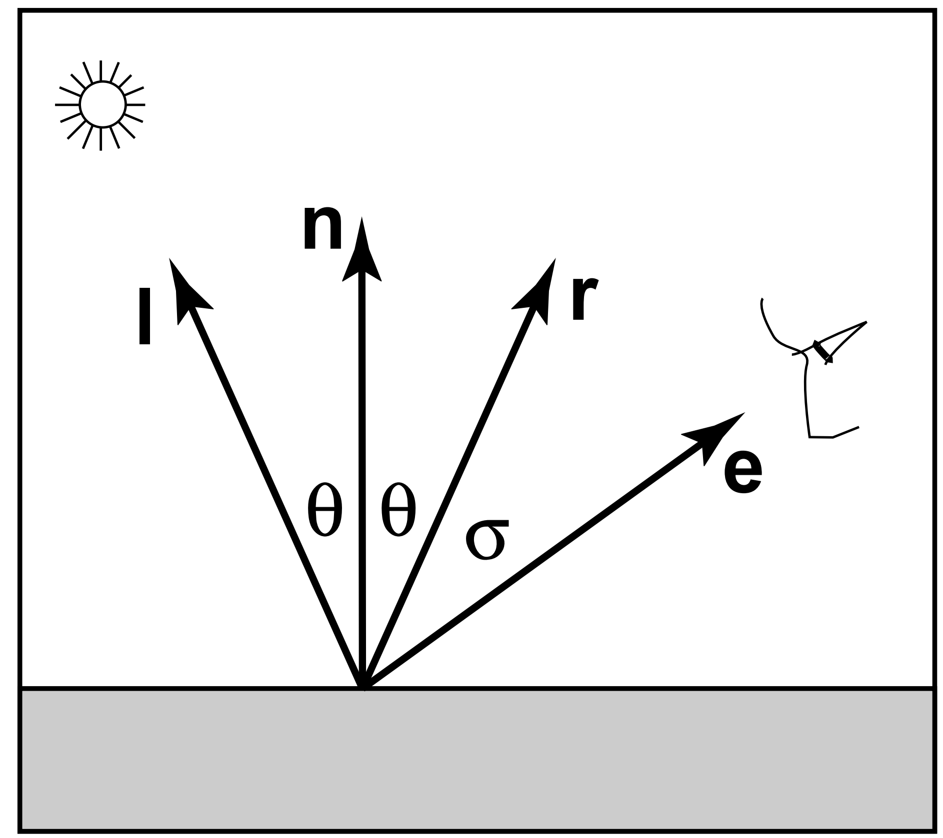


Heuristic: bright when **e** = **r** and falls of gradually

Phong shading

$c = c_l\,(\mathbf{e} \cdot \mathbf{r})$

Avoid negative dot prod with clamping



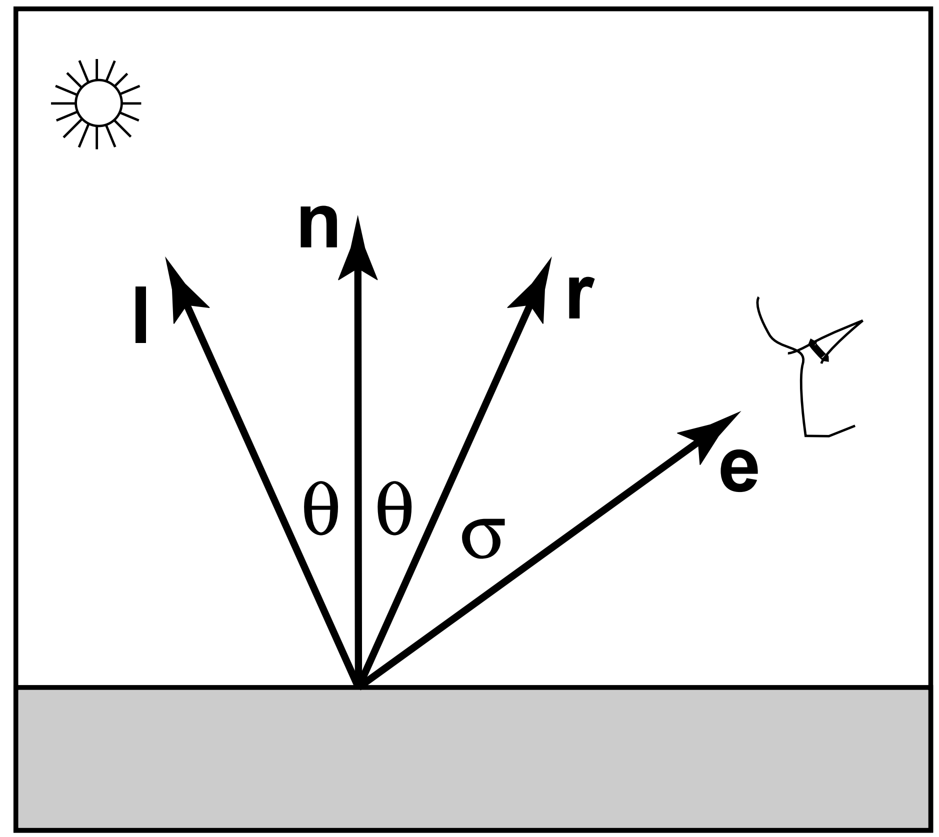
Heuristic: bright when **e** = **r** and falls of gradually

Phong shading

$c = c_l\,(\mathbf{e} \cdot \mathbf{r})$

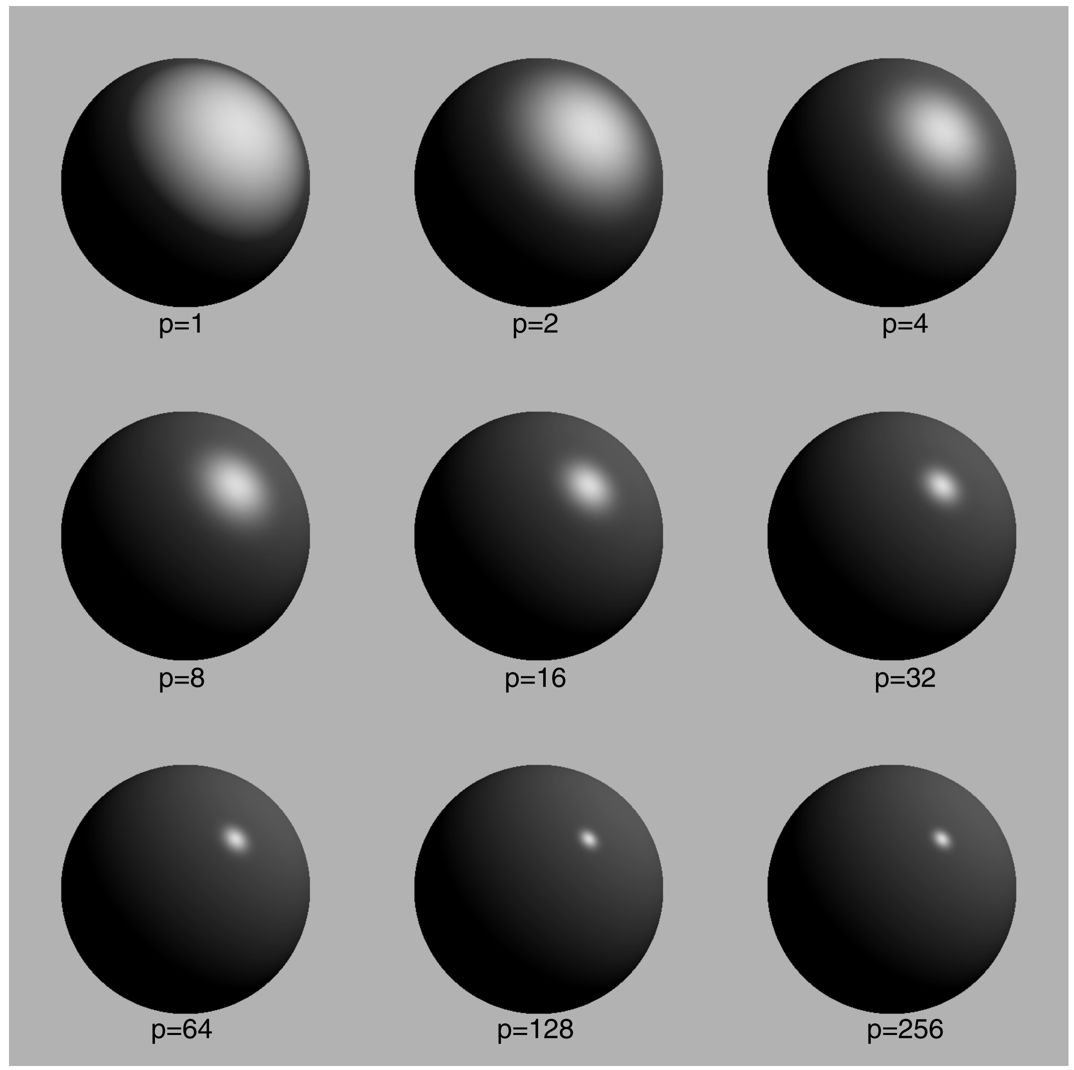
Avoid negative dot prod with clamping

Problem: fall off is not fast enough



Heuristic: bright when $\mathbf{e} = \mathbf{r}$ and falls of gradually

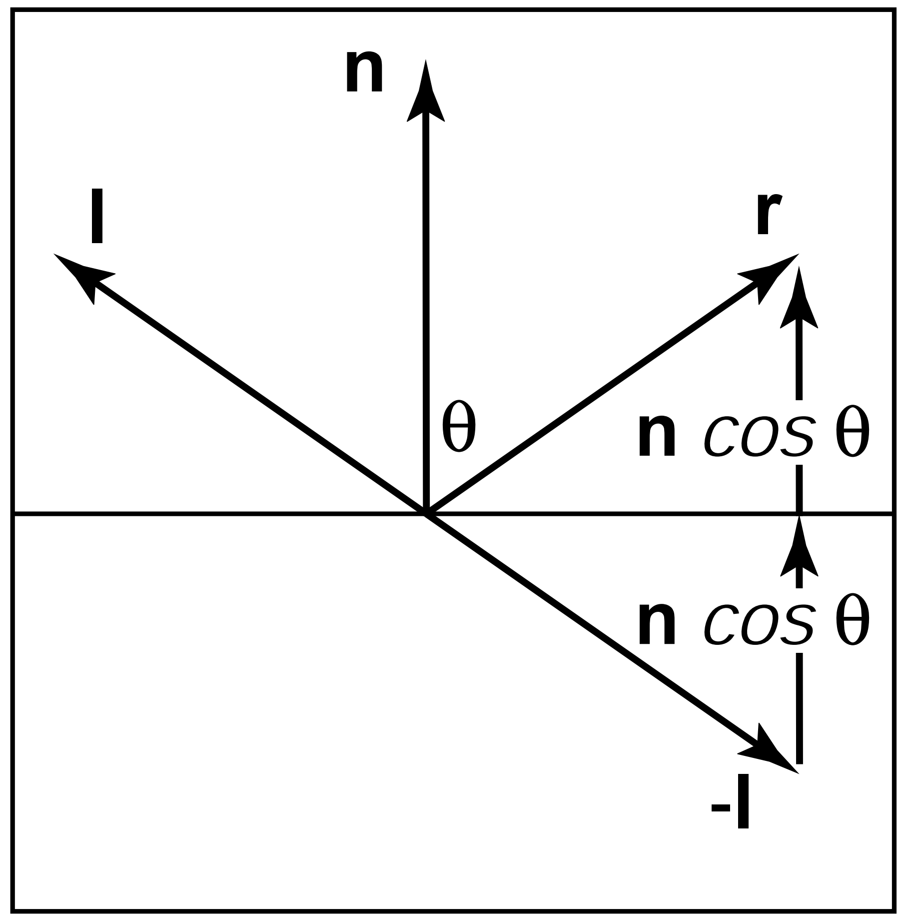
Phong shading



$$c = c_l \max(0, \mathbf{e} \cdot \mathbf{r})^p$$

p: the Phong exponent

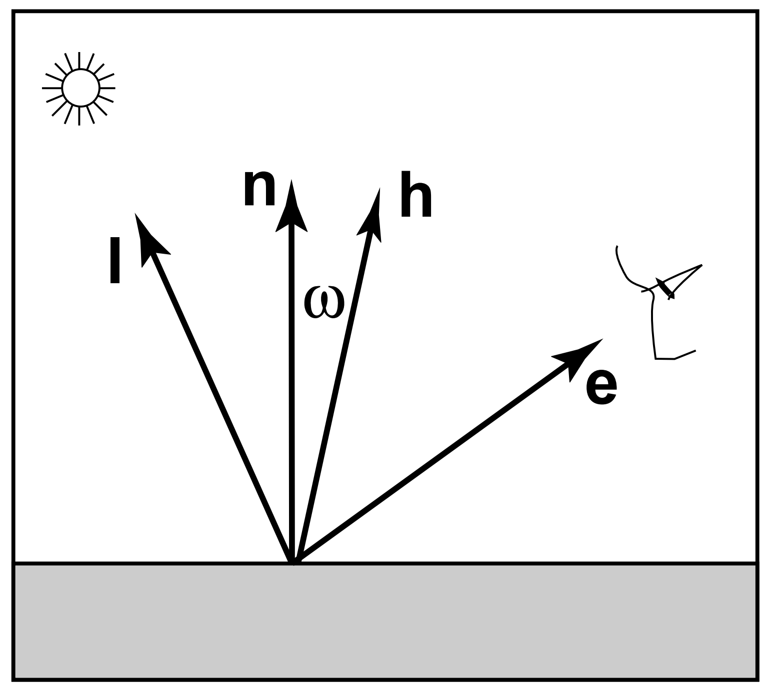
Phong shading



$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n}$$

Phong shading

Alternative:



$$\mathbf{h} = \text{unit}(\mathbf{e} + \mathbf{l}) \qquad c = c_l\,(\mathbf{h} \cdot \mathbf{n})^p$$

Diffuse and highlight

ambient + diffuse + specular

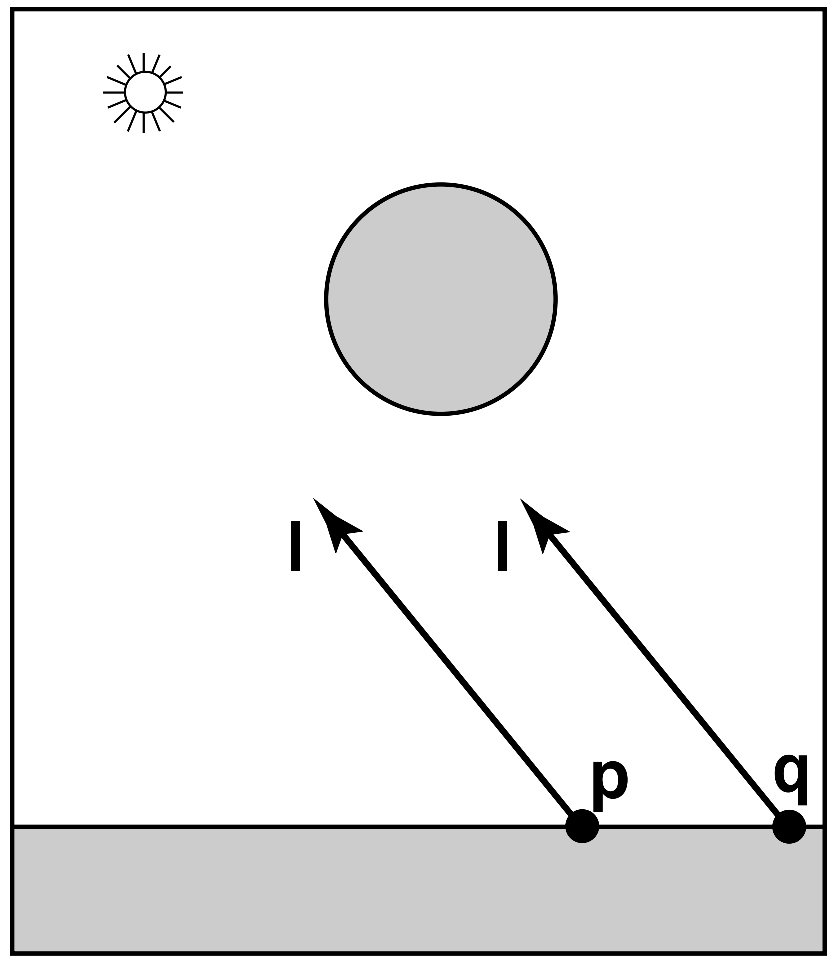
$$c = c_r\, c_a + c_r\, c_l \max(0, \mathbf{n} \cdot \mathbf{l})) + c_p\, c_l\, (\mathbf{h} \cdot \mathbf{n})^p$$

where c_p allows for dimming of the highlight

Phong normal interpolation

- Estimate normals at vertices
- Barycentric interpolation of normals across the triangle
- Use normal to compute shading (rather than interpolate vertex shading)

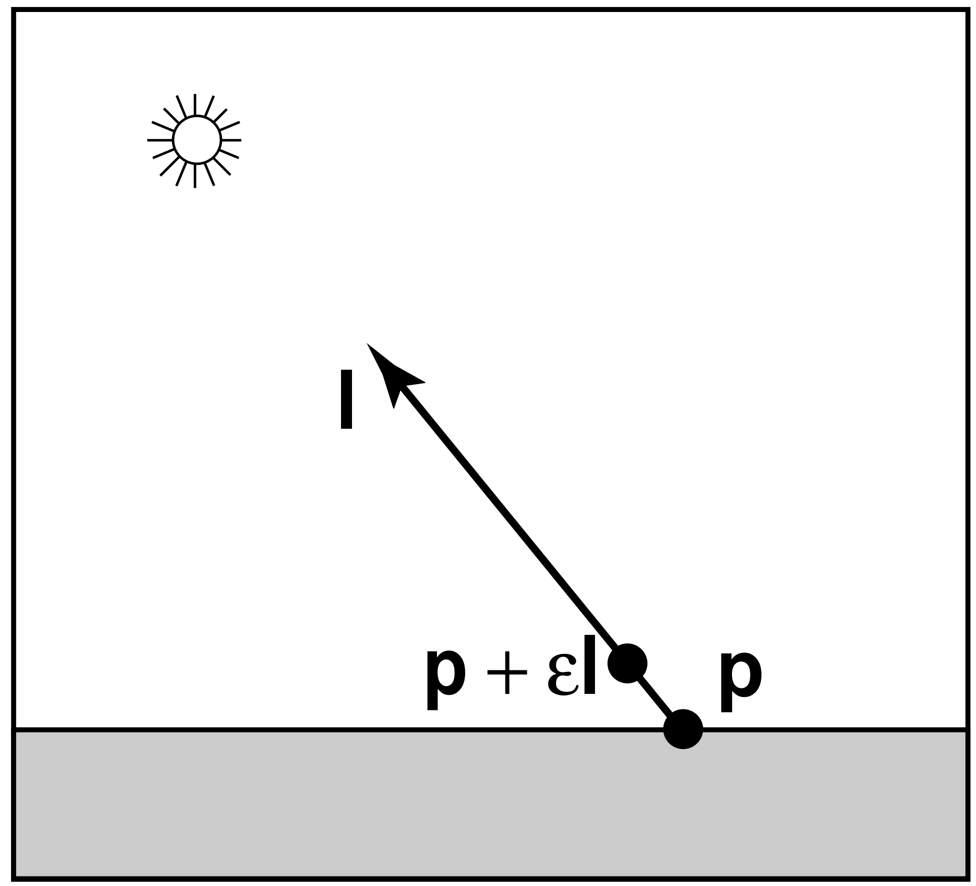
Ray tracing: shadows



Shadow rays

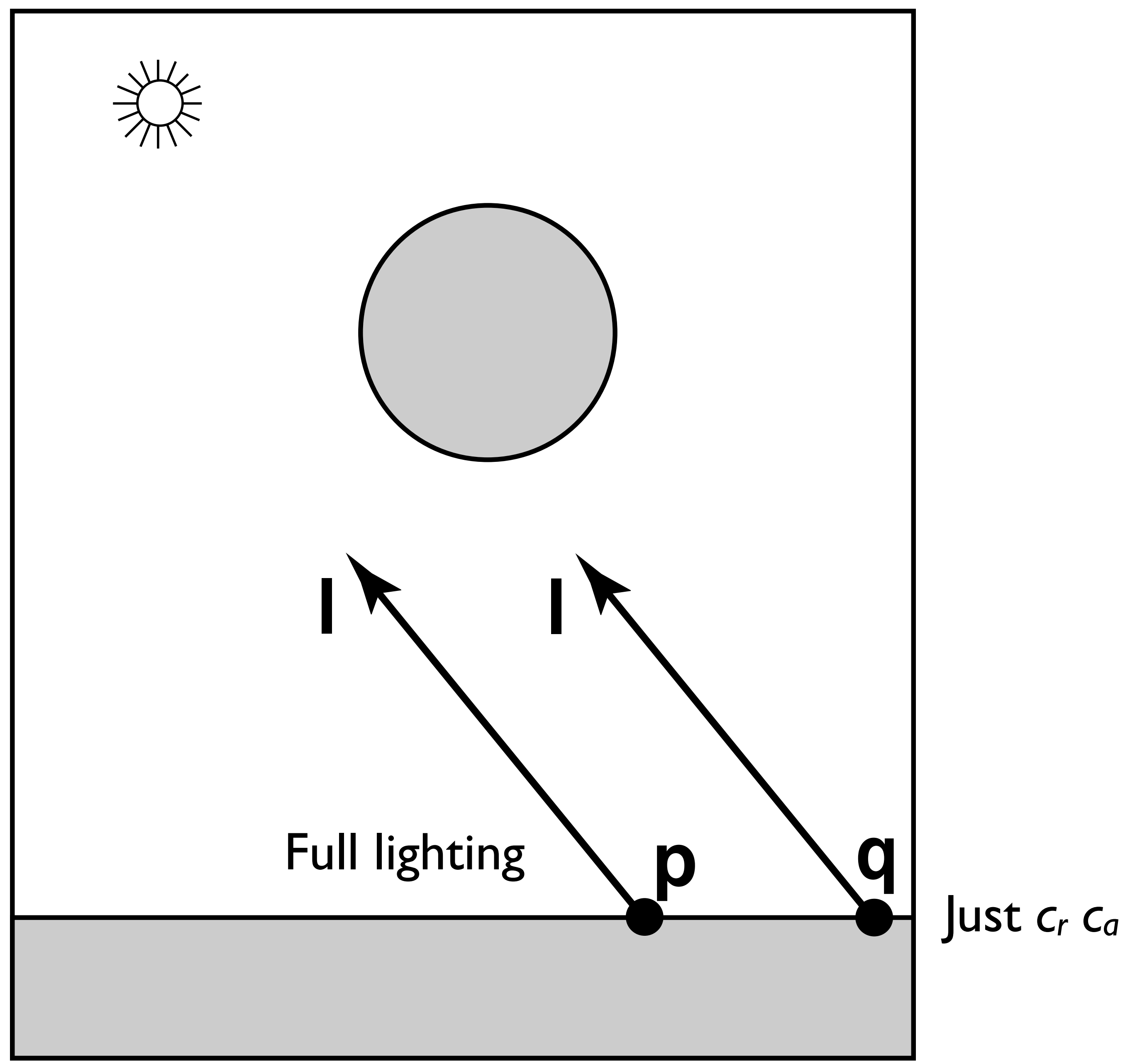
A point is in shadow if when "looking" at the light source form it, there is no occluding object

Ray tracing: shadows

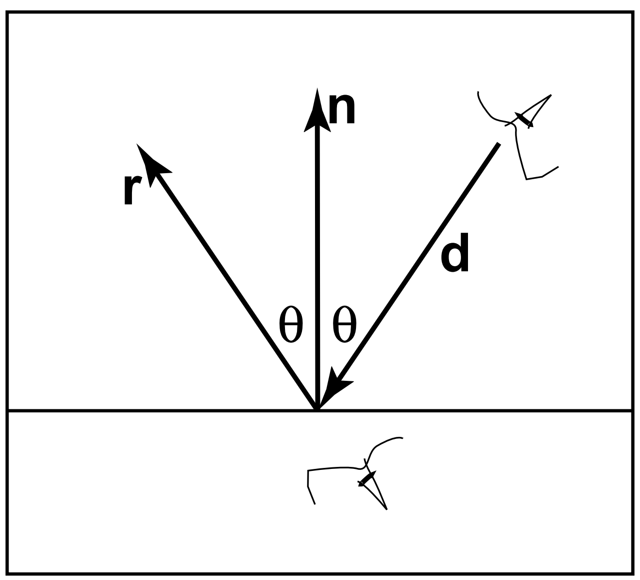


Use an offset to avoid accidental
re-intersecting with surface just hit

Ray tracing: shadows



Ray tracing: Specular Reflection



$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$

Recursively evaluate:

$c = c + c_s \,\text{raycolor}(\mathbf{p} + s\mathbf{r}, \epsilon, \infty)$

where c_s is specular color

Ray tracing: refraction

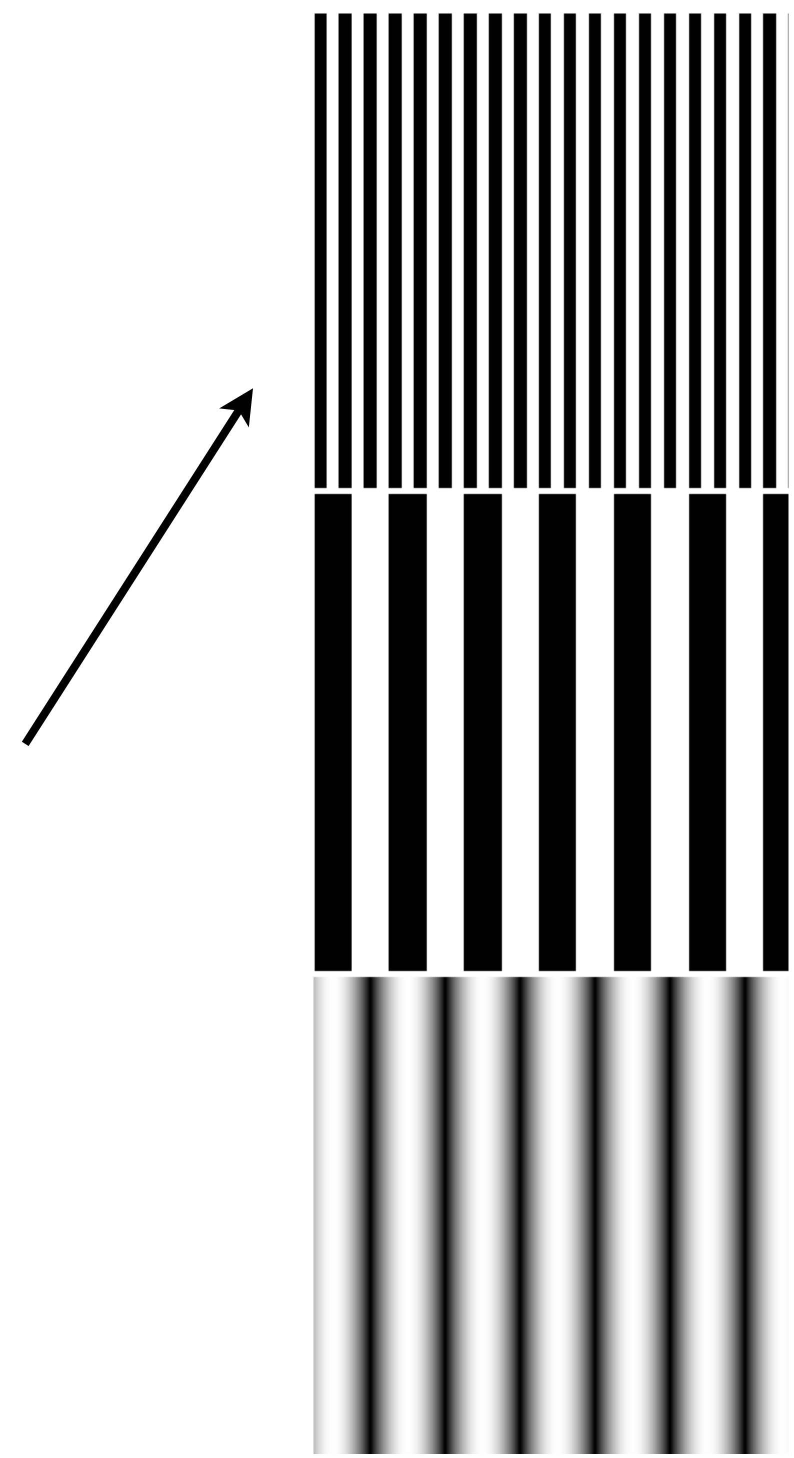
In tutorial

Texture mapping

- Capture variations of reflectance across a surface
- Rather than model detail with small polygons, create a mapping from surface to reflectance values
- Replace c_r with a mapping $c_r(\mathbf{p})$
- Procedural or table look-up (texture map)

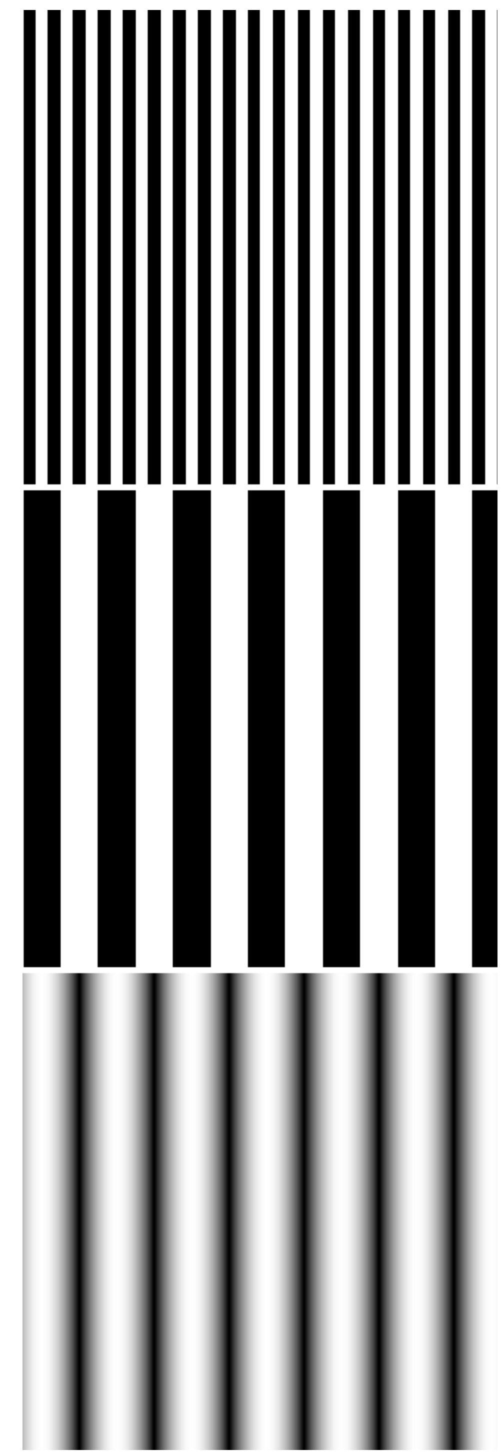
Texture mapping

RGPstripe (point **p**)
if (sin(x_p) > 0) **then**
  return c_0
else
  return c_1



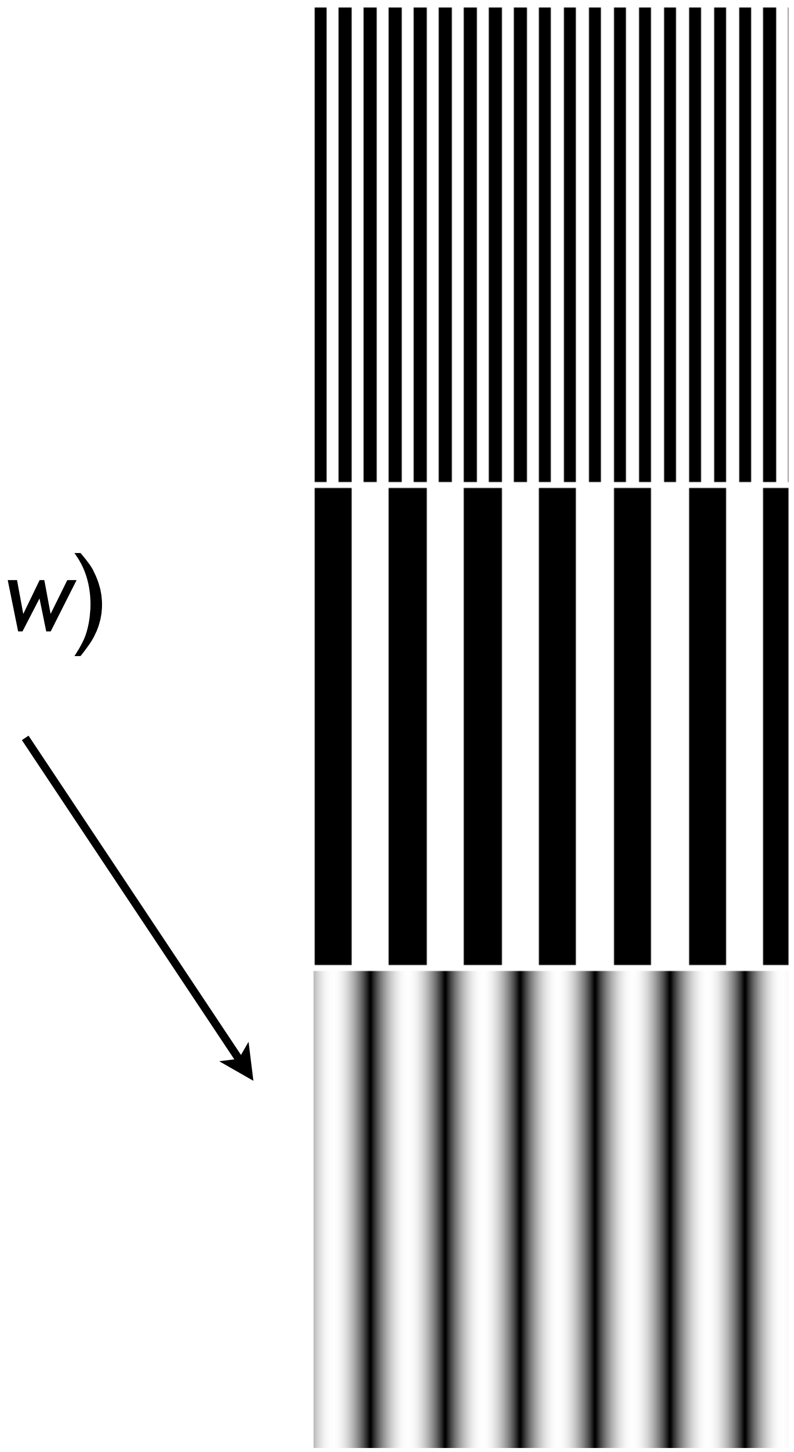
Texture mapping

RGPstripe (point **p**, real *w*)
if ($\sin(\pi x_p / w) > 0$) **then**
    return c_0
else
    return c_1



Texture mapping

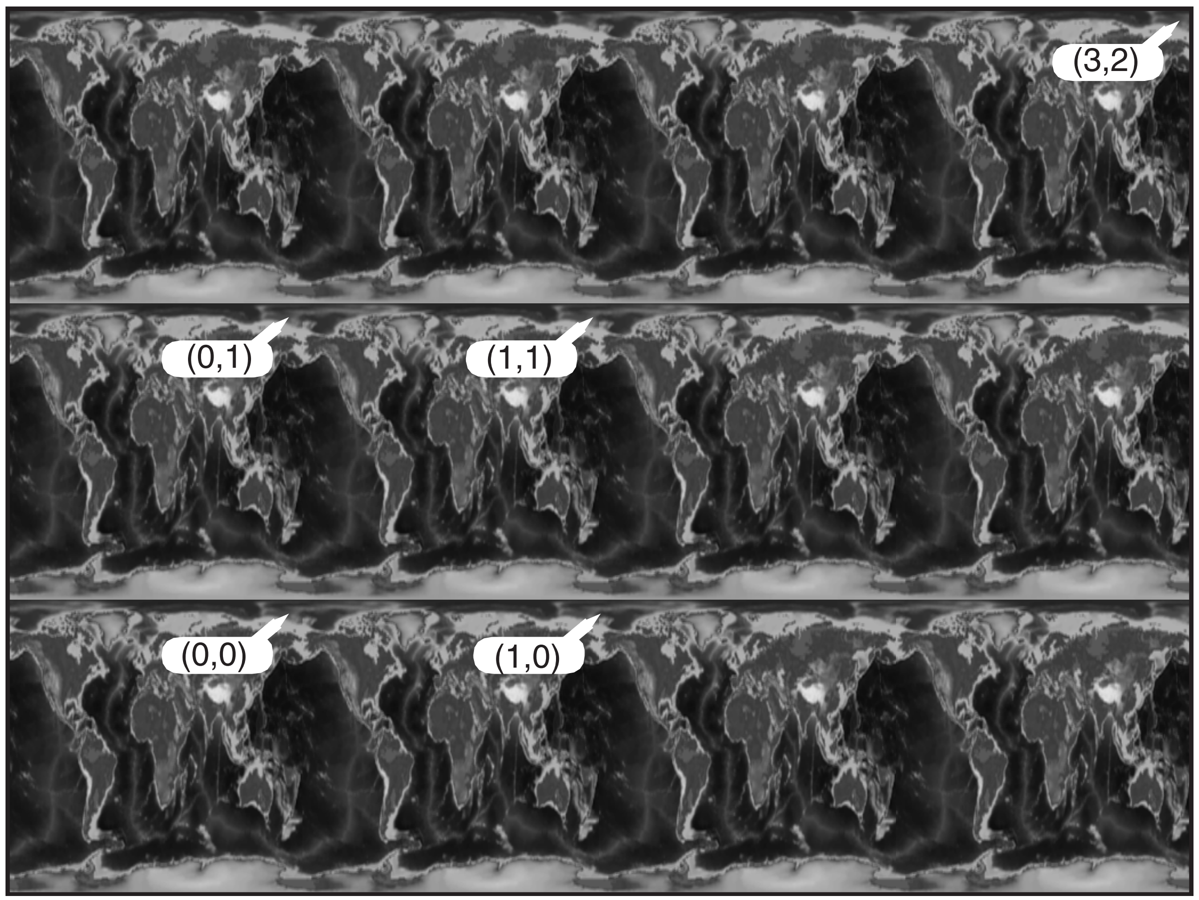
RGPstripe (point **p**, real *w*)
$t = (1 + \sin(\pi x_p / w))/2$
return $(1-t)c_0 + tc_1$



Perlin noise

Tutorial

Texture mapping



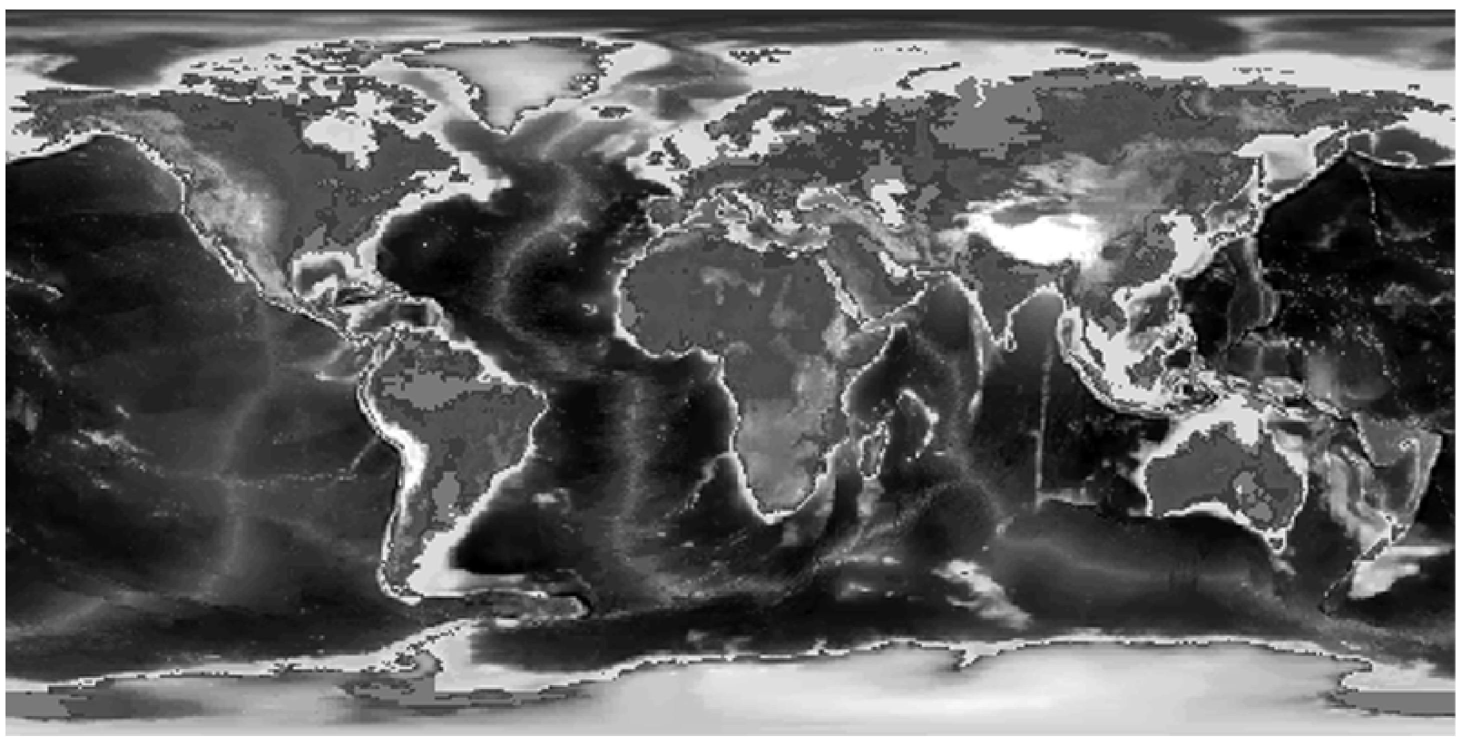
Texture arrays:

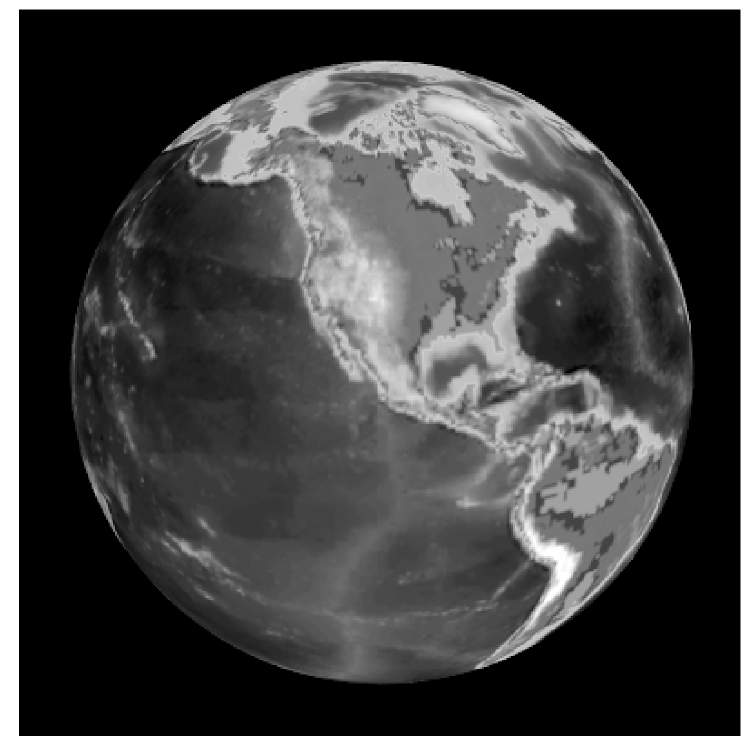
i = floor(un_x),
j = floor(vn_y);

u,v in [0,1]
n_x, n_y image size

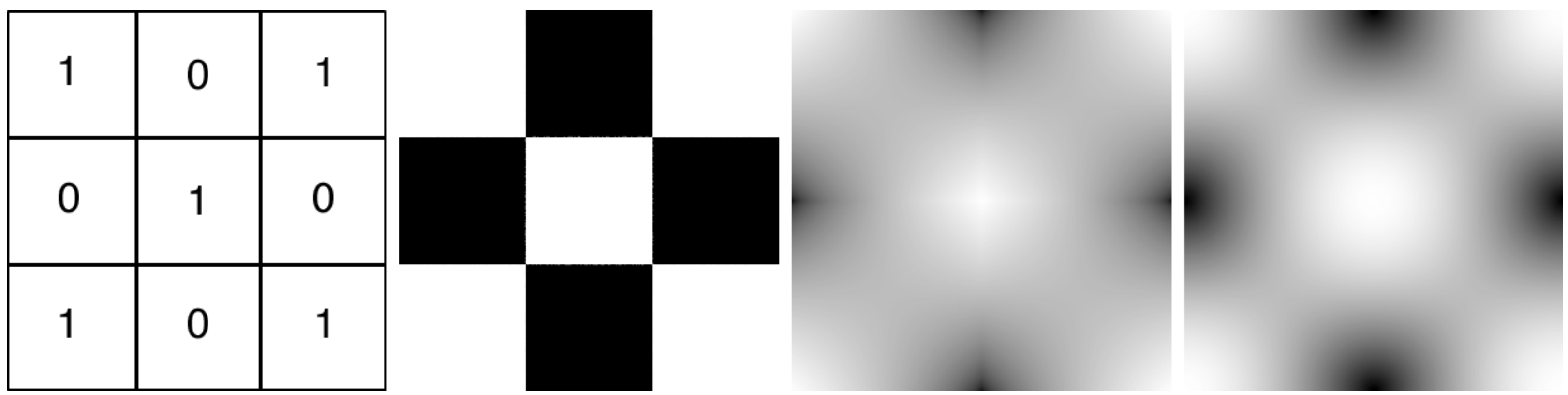
remove integer portion of u,v
results in tiling

Texture mapping





Interpolation



(a) (b) (c) (d)

a) image pixel values
b) nearest neighbor
c) bilinear
d) hermite

Texture mapping

- 3D textures defined in volume: (*x*, *y*, *z*), called only for points on the surface
- 2D textures defined on surface: (*u*, *v*), 2D (local) parametrization of the surface.