3D Graphics Programming

T163 - Game Programming

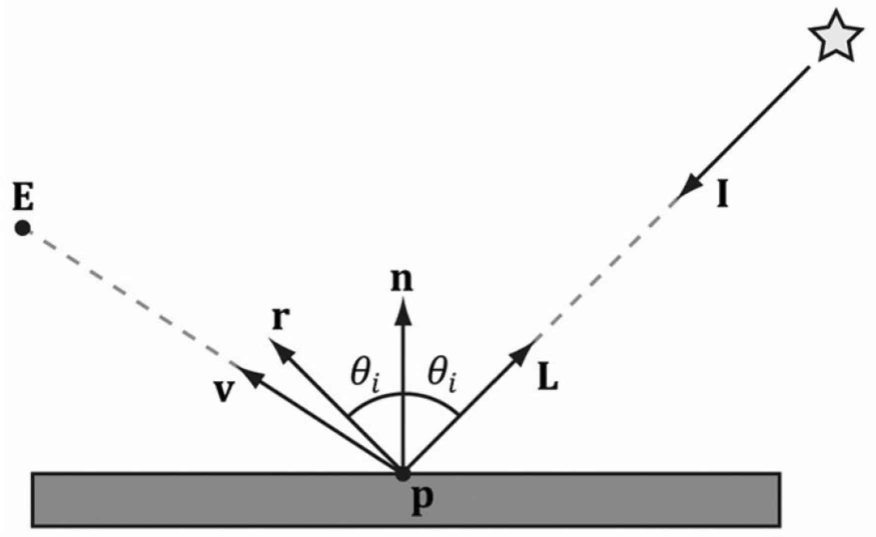
Fall 2020

Week 12

Lighting, cont'd

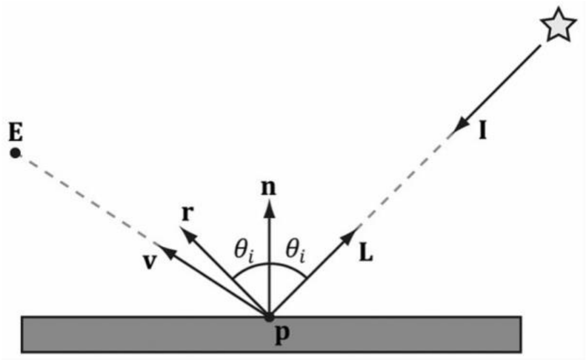
Lighting

- Important Vectors in Lighting
  - Consider the diagram below:

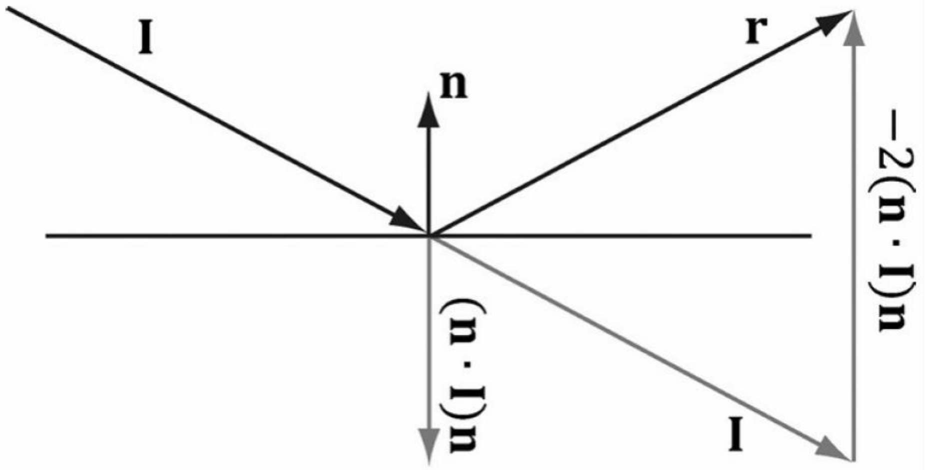


Lighting

- Important Vectors in Lighting
  - **E** is the eye position
  - We are considering the point **p** what the eye sees along the line of site defined by the unit vector **v**
  - At the point **p** the surface has normal **n**, and the point is hit by a ray of light traveling with incident direction **I**
  - The light vector **L** is the unit vector that aims in the opposite direction of the light ray striking the surface point



Lighting



- **r** is the reflection vector is given by **r = I – 2(n·I)n**
- It is assumed that **n** is a unit vector
- We can actually use the HLSL intrinsic **reflect** function to compute **r** for us

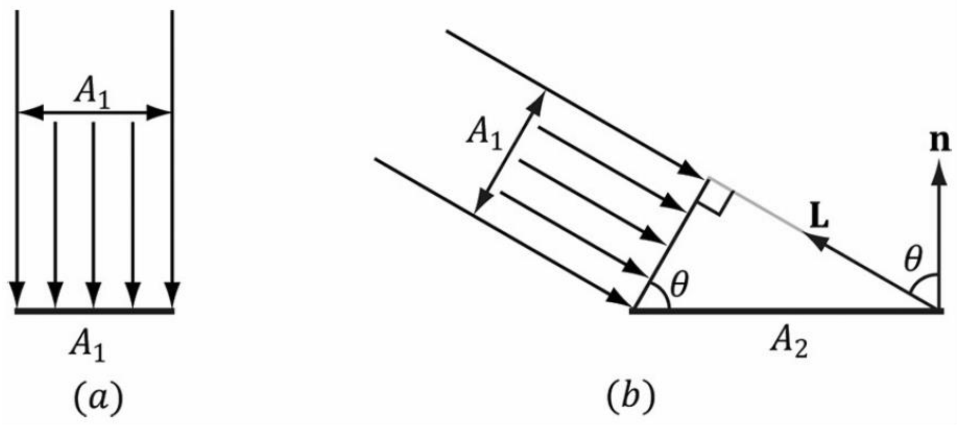
Lighting

- Lambert's Cosine Law
  - The amount of (light) energy emitted per second is called radiant flux
  - The density of radiant flux per area (irradiance) is important because that will determine how much light an area on a surface receives

  - The next slide shows a light beam with cross sectional area A1 strikes a surface head-on (a)
  - A light beam with cross sectional area A1 strikes a surface at an angle to cover a larger area A2 on the surface, thereby spreading the light energy over a larger area, thus making the light appear "dimmer" (b)

Lighting

- Below shows a light beam with cross sectional area A1 strikes a surface head-on (a)
- A light beam with cross sectional area A1 strikes a surface at an angle to cover a larger area A2 on the surface, thereby spreading the light energy over a larger area, thus making the light appear "dimmer" (b)

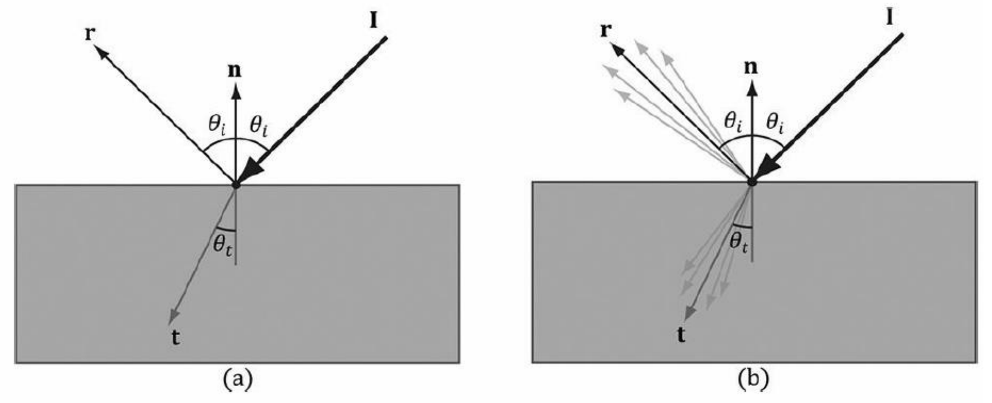


$$f(\theta) = \max(\cos\theta, 0) = \max(\mathbf{L}\cdot\mathbf{n}, 0)$$

Specular Material

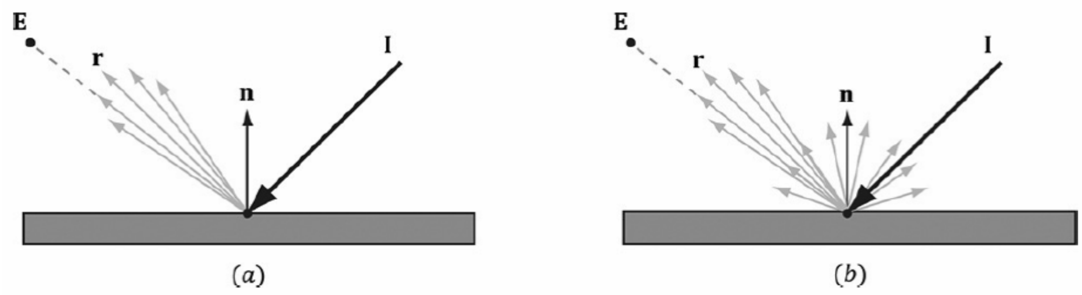
- To get the directional light, we needed to calculate the average normal of each vertex
- Now for specular we calculate the reflection

Specular Material



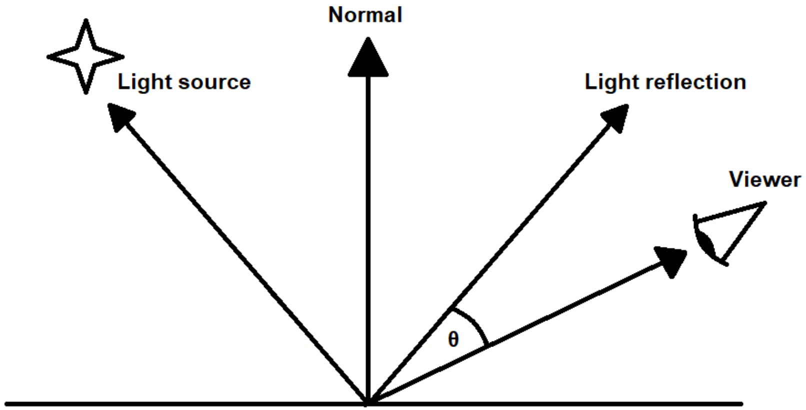
- The Fresnel effect for a perfectly flat mirror with normal **n** (a)
- The incident light **I** is split where some of it reflects in the reflection direction **r** and the remaining light refracts into the medium in the refraction direction **t**
- The angle between the reflection vector and normal is always $\boldsymbol{\theta_i}$, which is the same as the angle between the light vector **L = −I** and normal **n**
- The angle $\boldsymbol{\theta_t}$ between the refraction vector and **−n** depends in the indices of refraction between the two mediums and is specified by Snell's Law (b)

Specular Material



- Specular light of a rough surface spreads about the reflection vector r (a)
- The reflected light that makes it into the eye is a combination of specular reflection and diffuse reflection (b)

Specular Material

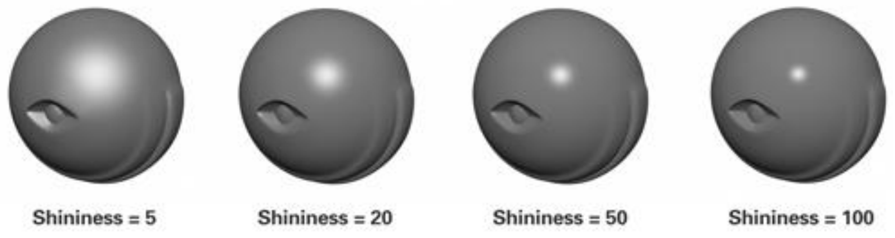


- Simpler explanation
  - Find the angle between viewer and light reflection
  - Smaller θ: more bright
  - Larger θ: more dim

Specular Material

- View vector is just the difference between the fragment position and the viewer (camera) position
- Reflection vector can be obtained with a built-in GLSL function:
  reflect(incident, normal)
  - Incident: vector to reflect
  - Normal: normal vector to reflect around
- Just as with diffuse, use dot product between normalized forms of view and refection vector, to get specular factor

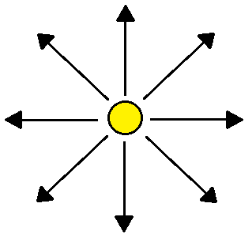
Specular Material



- One last step to alter specular factor: shininess
- Shininess creates a more accurate reflection
  - Higher shine: smaller more compact specular
  - Lower shine: larger, faded specular (matte)
- Previously calculated specular factor to the power of shininess value, thus:
  - **specularFactor = (view · reflection)^{shininess}**

Point Lights

- Before I get into the demo, let's go over how point lights work



- Point lights are lights with a position that emit light in ALL directions
  - It's not as complicated as it sounds though
  - We'll be copying code from the directional light
- Get difference between light position and fragment position, which is a direction that we normalize
- Then apply directional light math to the calculated direction vector
  - As I said…
- Since a lot of maths are the same for both lights, I add functions into the GLSL – because it's programmable

Point Lights

- Attenuation is falloff or drop-off of light from a point
- Linear drop-off is not accurate to real life
- In reality, light intensity initially drops quickly with distance
- But the further you are, the slower it decreases
- The inverse of a quadratic function can do this for us
  - **1/(ax2 + bx + c)**
  - Where x is the distance between the light source and fragment
  - And a is the exponent, b the linear and c the constant

Point Lights

$$\text{Attenuation Factor} = \frac{1.0}{\text{quadratic} * \text{distance}^2 + \text{linear} * \text{distance} + \text{constant}}$$

- distance: distance between light and fragment.
- quadratic: user-defined value, usually the lowest of the three.
- linear: user-defined value, lower than constant.
- constant: usually 1.0 to ensure denominator is always greater than 1.
  - e.g. if denominator is 0.5, then 1.0/0.5 = 2.0, so attenuation will DOUBLE power of light beyond its set value – that's bad!

- For useful values see: http://wiki.ogre3d.org/tiki-index.php?page=-Point+Light+Attenuation
- From the wiki: an accepted group of values for constant, linear, quadratic:
  - **1.0f, 4.5/range, 75.0f/(range*range)**
  - Where range is the absolute range limit of the light, i.e. blackness
- Alternatively, toy around with values!

Week 12

Lab Activities

Fall 2020

Week 12 Lab

- For the lab, see Hooman's material (with video)
- OpenGL examples covered:
  - Different lighting models
  - The rest of Hooman's examples from Week 11

Week 12

End