CENG 477
Introduction to Computer Graphics

Ray Tracing: Shading
Last Week

• Until now we learned:
  – How to create the primary rays from the given camera and image plane parameters
  – How to intersect these rays with various objects:
    • Planes
    • Spheres
    • Triangles
Algorithm So Far

foreach pixel s:
    compute the viewing ray \( r \) (from \( e \) to \( s \))
    \( t_{\text{min}} = \infty \), \( \text{obj} = \text{NULL} \)

foreach object o:
    if \( r \) intersects \( o \):
        if \( t < t_{\text{min}} \):
            \( t_{\text{min}} = t \), \( \text{obj} = o \)
    if \( \text{obj} \) not \( \text{NULL} \):
        pixel color = color of \( \text{obj} \)
    else
        pixel color = color of background
Image So Far

- Two spheres with different colors and a gray background
- So much work for such a bad image!
The image looks bad because we did not use a realistic *shading* algorithm.
We simply set all surface points to the same color.

Two spheres with different colors and a gray background.
So much work for such a bad image!
Reflectance and Power

• We assumed that each object has a color
• This is not how it is in reality
• Each object has a reflectance distribution
• Each light source has a power distribution
• Normally, both are functions of wavelength:
The object color occurs as a result of their interaction.

Same object can appear different under different lights.
Reflectance, Power, Cones

- Color is perceived due to the interaction of this reflected light by our cone pigments
Spectral Ray Tracing

• Accurately modeling this process requires us to represent all objects and light sources using spectral (i.e. wavelength-based) distributions

• Some ray-tracers do this:
  – Indigo renderer (http://www.indigorenderer.com/)
  – Lux renderer (http://www.luxrender.net/)
  – Mental ray (http://www.nvidia-arc.com/products/nvidia-mental-ray.html)
  – ...

CENG 477 – Computer Graphics
RGB Model

• We will make a simplifying assumption and represent all objects and light source with three components

• Reflectance:
  – $R_r$: How much red light it reflects, a value between [0,1]
  – $R_g$: How much green light it reflects, a value between [0,1]
  – $R_b$: How much blue light it reflects, a value between [0,1]

• Power:
  – $I_r$: How much red light it emits, a value between [0,∞)
  – $I_g$: How much green light it emits, a value between [0,∞)
  – $I_b$: How much blue light it emits, a value between [0,∞)
Terms

• The following terms are important when talking about light-surface interaction:
  – Power
  – Intensity
  – Radiance
  – Irradiance
Power

• Power is the time-rate of energy
• It measures how much energy (i.e. in Joules) a light source emits in all possible directions per second
• Power is measured in Watts (W = J/s)
• Power (P) is also known as flux (Φ)

\[ P = \frac{dQ}{dt} \]
Intensity

- Intensity is defined as power per solid angle
- Typically defined for point light sources and measured in W/sr

\[ I = \frac{dP}{dw} \]
Intensity

• Intensity is defined as power per solid angle
• Typically defined for point light sources and measured in W/sr

Here sr stands for steradians – 3D version of an angle (called solid angle)
• Max steradians is $4\pi$ just like a max 2D angle is $2\pi$
Solid Angle

- Solid angle is the 3D generalization of the angle
- It corresponds to a surface area on a unit sphere
  - similar to angle corresponding to a length on a unit circle
  - measured in steradians
Radiance

- Radiance is defined as power per solid angle per projected area
- Measured in Watts per steradian per meter squared \((W/sr/m^2)\)
Radiance

- Radiance is the most important unit in ray tracing
- Radiance of a ray does not change as the ray travels in empty space

The Goal of Ray Tracing

Compute the radiance along each viewing ray
Irradiance

- Irradiance measures the total incident power per unit area
- Measured in W/m$^2$

$$E = \frac{dP}{dA}$$
Irradiance from Radiance

- Irradiance can be computed from radiance:

\[ dE = \frac{dP}{dA} = L \cdot dw \cdot \cos \theta \]
Radiance from Intensity

- Finally, radiance can be computed from intensity:

\[
L = \frac{dI}{dA\cos\theta}
\]
Radiance from Intensity

• Now, imagine that dA was further away
• The same intensity would be spread over a larger area
• This relationship is governed by the inverse square law

\[ L = \frac{dI}{dA \cos \theta} \]
Inverse Square Law

• The irradiance emitted by a point light source is reduced by the square of the distance from the source
Inverse Square Law

- Objects further away from the light source appear dimmer as a result
- This is why the distance of the stars can be judged by their brightness
The Cosine Law

- The cosine law states that a surface can receive (or emit) radiation only in proportion to its area projected in the direction of the light:

\[ \cos \theta = \frac{-\mathbf{w}_i \cdot \mathbf{n}}{|\mathbf{w}_i||\mathbf{n}|} \]
Rendering Equation

• Putting together all these concepts gives us an equation known as the rendering equation

• It models how much light arriving from an incoming direction ($\mathbf{w}_i$) is reflected toward an outgoing direction ($\mathbf{w}_o$)
Rendering Equation

\[ L_o(x, w_o) = L_e(x, w_o) + \int_{\Omega} f(x, w_o, w_i) L_i(x, w_i) \, dw \cos \theta \]

- **Outgoing radiance**
- **Emitted radiance**
- **Reflectance Function (BRDF)**
- **Incoming radiance**
- **Perpendicularly received irradiance**
  - Integration over hemisphere
  - Differential solid angle
  - Area correction factor
Rendering Equation

• Too costly to evaluate
• For each surface point \( x \) we need to integrate over the entire hemisphere surrounding \( x \)
• Simplification:
  – Exclude all directions except light sources
  – Add an ambient term to simulate what was excluded
Surface Normals (Reminder)

• A **surface normal** is a unit vector which is **orthogonal** to the surface and points **outward** from the surface.
Surface Normals (Reminder)

- Depends on the type of the object:
  - For a **plane**, surface normal is given so you don’t have to do anything (except to normalize it if it isn’t normalized).
  - For a **sphere**, compute:
    
    $$ n = \frac{p - c}{R} $$

  - For a **triangle**, compute cross product of two edge vectors:
    
    $$ n = \frac{(b - a) \times (c - a)}{\| (b - a) \times (c - a) \|} $$
Shading Models

• We will approximate the rendering equation as a sum of three simple shading models:
  – Diffuse shading
  – Specular shading
  – Ambient shading

• All models are based on a modified rendering equation:

\[
L_o(x, w_o) = \sum_{i=1}^{l} s(x, w_o, w_i)E_i(x, w_i)
\]
Shading Models

• Compare the surface shading equation with the rendering equation

\[
L_o(x, w_o) = \sum_{i=1}^{l} s(x, w_o, w_i)E_i(x, w_i)
\]

\[
L_o(x, w_o) = L_e(x, w_o) + \int_{\Omega} f(x, w_o, w_i)\cos\theta L_i(x, w_i)dw
\]
Diffuse Shading

• Simulates the phenomenon that a surface can receive radiation only in proportion to its area projected in the direction of the light:

\[ L_d^d(x, wo) = k_d \cos \theta' E_i(x, w_i) \]

- Outgoing radiance
- Diffuse reflectance coefficient
- Perpendicularly received irradiance

\[ \cos \theta' = \max(0, w_i \cdot n) \]
Diffuse Shading

• Note that a diffuse surface reflects equal radiance in all directions (looks equally bright from all viewing directions)
• Such a surface is called a Lambertian surface (1760)

\[ L_d^d(x, wo) = k_d \cos \theta' E_i(x, w_i) \]

Outgoing radiance
Diffuse reflectance coefficient
Received irradiance

\[ \cos \theta' = \max(0, w_i \cdot n) \]
Diffuse Shading

- Also, for a point light source, we must compute irradiance at a distance \( r \) from the light source: inverse square law
- This improves realism

\[
L^d_0(x, wo) = k_d \cos \theta' \frac{I}{r^2}
\]

\[
\cos \theta' = \max(0, w_i \cdot n)
\]
Diffuse Shading Example

- A sphere shaded by diffuse shading:
Ambient Shading

• With diffuse shading parts of the sphere that do not face the light source are all black.
• Ambient shading is used as a very crude approximate of the integral in rendering equation:

\[ L^a_o(x, wo) = k_a I_a \]

- \( L^a_o \): Outgoing radiance
- \( k_a \): Ambient reflectance coefficient
- \( I_a \): Ambient radiance
Ambient Shading

Diffuse only

Diffuse + Ambient
Specular Shading

• Simulates shiny surfaces
• Unlike diffuse shading, specular shading is **view-dependent**
• Several models exist with the most commonly used (and one of the simplest) being **Blinn-Phong shading**
Blinn-Phong (Specular) Shading

- Assumes perfect reflection occurs along the mirror direction of the incoming light
- But for efficiency, a vector known as the half vector is computed
- The angle between the surface normal and the half vector determines the shininess

\[ h = \frac{w_i + w_o}{\|w_i + w_o\|} \]
Blinn-Phong (Specular) Shading

\[ L_s(x, wo) = k_s \cos \alpha' E_i(x, w_i) \]

\[ \cos \alpha' = \max(0, n \cdot h) \]
Blinn-Phong (Specular) Shading

• To control the shininess, an exponent known as the Phong exponent is introduced:

\[ L_s^\alpha(x, w_0) = k_s(\cos\alpha')^p E_i(x, w_i) \]

\[ \cos\alpha' = \max(0, n \cdot h) \]
Blinn-Phong (Specular) Shading

Diffuse + Ambient

Diffuse + Ambient + Specular
Phong Exponent

- Larger Phong exponent makes the highlight more focused, giving the object a shinier appearance

http://cgru.sourceforge.net/
Putting it All Together

• The color of an object at any point can be computed by combining **diffuse**, **ambient**, and **specular** components:

\[
L_o (x, wo) = L_o^d(x, w_o) + L_o^a(x, wo) + L_o^s(x, wo)
\]

• Note that **L** is vector-valued in which each component represents the radiance for each color channel

\[
L_o(x, wo) = \begin{bmatrix} R \\ G \\ B \end{bmatrix}
\]
Multiple Light Sources

- Diffuse and specular contribution of each light source must be accumulated:

\[
L_o(x, w_o) = L_o^a(x, w_o) + \sum_{i=1}^{l} L_o^d(x, w_o)_i + L_o^s(x, w_o)_i
\]

Number of light sources
The Ray Tracing Algorithm – So Far

foreach pixel s:
  compute the viewing ray $r$ (from $e$ to $s$)
  $t_{\text{min}} = \infty$, $\text{obj} = \text{NULL}$
  foreach object $o$:
    if $r$ intersects $o$ at point $x$:
      if $t < t_{\text{min}}$:
        $t_{\text{min}} = t$, $\text{obj} = o$
    if $\text{obj}$ not NULL:
      pixel color = $L_o(x, w_o)$ // $w_o$ is the unit vector from $s$ to $e$
  else
    pixel color = color of background (or black)
Shadows

- Shadows can easily be added by checking if the light source is visible from the intersection point.
- In the configuration below, point $x$ is in shadow:
Shadows

- To check if the light source is visible from the intersection point, we can create shadow rays, one for each light source.

\[ s(t) = x + tw_i \]
Shadows

- Intersect $s(t)$ with all the objects in the scene. If there is no intersection before the light source, the point is not in shadow. Otherwise it is in shadow.

$$s(t) = x + tw_i$$
Shadows

- To avoid self intersection due to floating point precision problems, the origin is offset by a very small amount:

\[ s(t) = (x + w_i \varepsilon) + tw_i \]

Epsilon (\( \varepsilon \)) is a very small number such as 0.0001
The Ray Tracing Algorithm – So Far

\[ \text{foreach pixel } s: \]
\[ \quad \text{compute the viewing ray } r \text{ (from } e \text{ to } s) \]
\[ t_{\text{min}} = \infty, \ obj = \text{NULL} \]
\[ \text{foreach object } o: \]
\[ \quad \text{if } r \text{ intersects } o \text{ at point } x: \]
\[ \quad \quad \text{if } t < t_{\text{min}}:\]
\[ \quad \quad \quad t_{\text{min}} = t, \ obj = o \]
\[ \text{if } obj \text{ not NULL: } // \text{ viewing ray intersected with an object} \]
\[ \quad \text{pixel color} = L_a // \text{ambient shading is not affected by shadows} \]
\[ \text{foreach light } l: \]
\[ \quad \text{compute the shadow ray } s \text{ from } x \text{ to } l \]
\[ \text{foreach object } p: \]
\[ \quad \text{if } s \text{ intersects } p \text{ before the light source}: \]
\[ \quad \quad \text{continue the light loop}; // \text{point is in shadow – no contribution from this light} \]
\[ \quad \text{pixel color} += L_d + L_s // \text{add diffuse and specular components for this light source} \]
\[ \text{else} \]
\[ \quad \text{pixel color} = \text{color of background (or black)} \]

Do not forget clamping the pixel value to \([0, 255]\) range and rounding it to the nearest integer!
Image So Far

- This looks much better! 😊
Recursive Ray Tracing

- Ray tracing is recursive by nature
- Remember the rendering equation

\[
L_o(x, w_o) = L_e(x, w_o) + \int f(x, w_o, w_i)L_i(x, w_i)\cos\theta dw
\]

This term itself could be due to reflection (or refraction) from another surface
Recursive Ray Tracing

- Tracing the incident ray back into the scene (not just directly towards light sources) is called **path tracing**
- This is extremely costly so will only do it for perfect mirrors
Ideal Specular Reflection (Mirrors)

- Mirror-like objects reflect colors of other objects
- Given $w_o$, we must compute $w_r$ (reflection direction)

\[ w_r = -w_o + 2n \cos \theta = -w_o + 2(n \cdot w_o) \]

$n, w_o, w_r$ are all unit vectors

\[ L^m_o(x, w_o) = k_m L_i(x, w_r) \]

Mirror reflection coefficient
Ideal Specular Reflection (Mirrors)

• For these surfaces the total radiance is equal to local shading radiance plus the radiance from the mirrored location

\[ L_o(x, wo) = L^d_o(x, wo) + L^a_o(x, wo) + L^s_o(x, wo) + L^m_o(x, wo) \]
Final Image

- This looks much better! 😊
Ray Tracing

• We have only scratched the surface of ray tracing.
• There is so much more that can be done:
  – Acceleration structures
  – Adding texture
  – Modeling non-point light sources (area lights)
  – Refraction (glass-like materials)
  – Participating media (fog, smoke)
  – Subsurface scattering (candles)
  – Complex surface models
  – Parallel and interactive ray tracing
  – Ray tracing on the GPU
  – Photon mapping
  – Global illumination
  – Image-based lighting
  – ...
Some Ray-traced Images
Some Ray-traced Images

Physically Based Ray Tracer
Some Ray-traced Images
Some Ray-traced Images

Physically Based Ray Tracer
Some Ray-traced Images

Radiance Ray Tracer
Some Ray-traced Images

METU Ray Tracer