Announcements
Announcements

• HW1 is out - due next Monday
Announcements

• HW1 is out - due next Monday
  • HW1 Problems 3 and 4 will help with A2 TODO 9 and 8.
Announcements

• HW1 is out - due next Monday
  • HW1 Problems 3 and 4 will help with A2 TODO 9 and 8.

• A2 is now due Friday 2/7
Ray Tracing: Pseudocode

for each pixel:
  generate a viewing ray for the pixel
  find the closest object it intersects
  determine the color of the object
Ray Tracing: Pseudocode

for each pixel:
    generate a viewing ray for the pixel
    find the closest object it intersects
    determine the color of the object
Shading

What does the color of a pixel depend on?

• surface normal \( \text{stored in or calculated from object} \)

• surface properties (color, shininess, ...) \( \text{stored in object} \)

• eye direction \( \text{calculated from viewing ray and intersection point} \)

• light direction (for each light) \( \text{calculated from light and intersection point} \)
Ray-Sphere: Code Sketch

function ray_intersect(ray, sphere, tmin, tmax):

• Use last lecture's math to find +/- \( t \)

• If no real solutions, return nothing

• Otherwise, return closest \( t \) that lies between \( t_{\text{min}} \) and \( t_{\text{max}} \)

• Also return info needed for shading - store in a HitRecord struct.

In A2: \( t \), intersection point, normal, texture coordinate, object
Ray-Scene: Code Sketch

**Brute force:** check all objects.
There are better ways - more on this later.

```python
find_intersection(ray, scene):
    closest_t = Inf # closest intersection t
    closest_rec = nothing # HitRecord for intersection
    for obj in scene:
        t, rec = ray_intersect(ray, obj, l, closest_t)
        if rec != nothing:
            closest_t = t
            closest_rec = rec
    return closest_t, closest_rec
```
scene = model_scene()
for each pixel (i,j):
    ray = get_view_ray(i, j)
    t, rec = find_intersection(ray, scene)
    if rec != nothing:
        canvas[i,j] = obj.color
    else:
        canvas[i,j] = scenebgcolor
Ray Tracing: Code Sketch

scene = model_scene()
for each pixel (i,j):
    ray = get_view_ray(i, j)
    t, rec = find_intersection(ray, scene)
    if rec != nothing:
        canvas[i,j] = obj.color
    else:
        canvas[i,j] = scene.bgcolor
Ray Tracing: Code Sketch

def model_scene():
    # Initialize the scene model

for each pixel (i, j):
    ray = get_view_ray(i, j)
    t, rec = find_intersection(ray, scene)
    if rec != nothing:
        canvas[i, j] = obj.color
    else:
        canvas[i, j] = scene.bgcolor

Let's work on this.
Light Sources

• Where does light come from?

• Two simple kinds of sources:
  • point source: defined by a 3D position
  • directional source: defined by a 3D direction vector
Diffuse (Lambertian) Reflection

- On a *diffuse* surface, light scatters uniformly in all directions.
- No dependence on view direction.
- Many surfaces are approximately diffuse:
  - matte painted surfaces, projector screens,
  - anything that doesn't look "shiny"
Diffuse (Lambertian) Reflection

- whiteboard
Diffuse (Lambertian) Reflection
Diffuse (Lambertian) Reflection

The top face of a cube receives some amount of light.
Diffuse (Lambertian) Reflection

The top face of a cube receives some amount of light. Rotated 60°, the same face receives half the light.
Diffuse (Lambertian) Reflection

The top face of a cube receives some amount of light.

Rotated 60°, the same face receives half the light.

Light per unit area is proportional to \( \cos \theta = \vec{n} \cdot \vec{l} \).
Diffuse (Lambertian) Shading

• The full model:

\[ L_d = k_d I \max(0, \vec{n} \cdot \vec{l}) \]

- diffuse coefficient
- diffusely reflected light
- why max?
- light intensity

\[ L_d \]
Diffuse (Lambertian) Shading

• The full model:

\[ L_d = k_d I \max(0, \mathbf{n} \cdot \mathbf{l}) \]

- diffuse coefficient
- diffusely reflected light
- light intensity

why max?
Diffuse (Lambertian) Shading

\[ L_d = k_d I \max (\vec{n} \cdot \vec{l}) \]

For colored objects, \( k_d \) is a 3-vector of R, G, and B reflectances.
Exercise: Diffuse Reflection
Let's talk shinies.

- How does a mirror interact with light?

(whiteboard)
Specular Reflection

• What about non-mirror shiny surfaces?

• They appear brighter near "mirror" configuration.

• Phong reflection: specular reflection is a function of angle between $\mathbf{r}$ and $\mathbf{v}$. 
Specular Reflection

- Blinn-Phong: specular reflection is a function of angle between half-way vector between view and light and the normal.

- \( h = \text{bisector}(v, l) \)

- Reflected light proportional to

\[
k_s \max(0, \bar{n} \cdot \bar{h})^p
\]

specular coefficient: determines strength of specularity term

specular exponent: determines shininess
Effect of $p$
Putting it all Together: Blinn-Phong Reflection Model

Usually surfaces have both diffuse \textit{and} specular components, so we'll combine the two:

\[ L = L_d + L_s \]

\[ = k_d I \max(0, \vec{n} \cdot \vec{l}) + k_s I \max(0, \vec{n} \cdot \vec{h})^p \]

- \textit{L} \textit{d} = \text{diffuse reflection}
- \textit{L} \textit{s} = \text{specular reflection}
- \textit{k} \text{d} = \text{diffuse coefficient (surface brightness and color)}
- \textit{k} \text{s} = \text{specular coefficient (strength [and color] of specularity)}
- \textit{I} = \text{light intensity}
- \vec{n} = \text{normal}
- \vec{l} = \text{light direction}
- \vec{h} = \text{half-vector between I and v}
- \text{specular exponent (sharpness of specularity)}
Mirror Reflection

What does a camera see when it looks at a mirror?

Can we do this using the tools we already have?

(Exercise 2)
Mirror Reflection

What does a camera see when it looks at a mirror?

Can we do this using the tools we already have?

find_intersection(ray, scene)
Mirror Reflection

What does a camera see when it looks at a mirror?

compute \( r \)
ray = Ray(x, r)
find_intersection(ray, scene)
Recursion!?

traceray(ray, scene):
    t, rec = find_intersection(ray, scene)
    if rec.obj is a mirror:
        compute $r$, the reflection direction
        mirror_ray = Ray(rec.x, $r$)
        return traceray(mirror_ray, scene)
    # other cases, ...
Mirror Coefficient

- Most surfaces aren't perfect mirrors. Object stores a mirror coefficient \( k_m \) between 0 and 1.

- "Local color" computed as usual

- "reflected color" computed recursively

- Final color: \( k_m \times \text{reflected} + (1-k_m) \times \text{local} \)