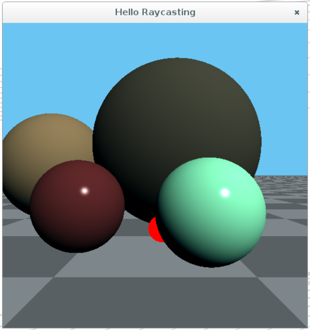
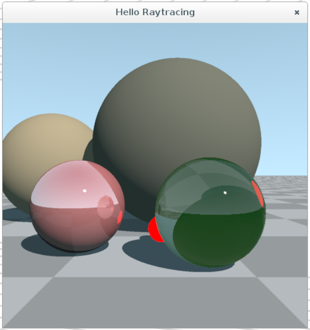
Ray-based graphics, and the rendering equation

Computer Graphics (DT3025)

Martin Magnusson
November 18, 2016





Last time

- Casting real-time shadows.

Last time

- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.

Last time

- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.
  - Shadow volumes: exact hard shadows, but expensive for complex geometries.

Last time

- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.
  - Shadow volumes: exact hard shadows, but expensive for complex geometries.
  - The stencil buffer.

Last time

- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.
  - Shadow volumes: exact hard shadows, but expensive for complex geometries.
  - The stencil buffer.
  - Shadow mapping: noisy and blurry — unless we take good care to improve them — but better for complex geometry.

Last time

- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.
  - Shadow volumes: exact hard shadows, but expensive for complex geometries.
  - The stencil buffer.
  - Shadow mapping: noisy and blurry — unless we take good care to improve them — but better for complex geometry.
  - Tuning the bias for avoiding z-fighting.

Last time

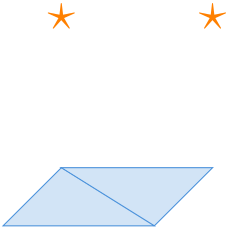
- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.
  - Shadow volumes: exact hard shadows, but expensive for complex geometries.
  - The stencil buffer.
  - Shadow mapping: noisy and blurry — unless we take good care to improve them — but better for complex geometry.
  - Tuning the bias for avoiding z-fighting.
  - Warping and partitioning for avoiding aliasing.

Last time

- Casting real-time shadows.
  - Planar shadows: easy to implement, but not flexible.
  - Shadow volumes: exact hard shadows, but expensive for complex geometries.
  - The stencil buffer.
  - Shadow mapping: noisy and blurry — unless we take good care to improve them — but better for complex geometry.
  - Tuning the bias for avoiding z-fighting.
  - Warping and partitioning for avoiding aliasing.
- (From now on, we'll deal with methods where we get proper shadows "for free"!)

Shadow volumes: how much geometry?

We have the following scene with 2 triangles and 2 omnidirectional point light sources. How many triangles have to be generated for computing the shadow volumes?



1. 10
2. 16
3. 24

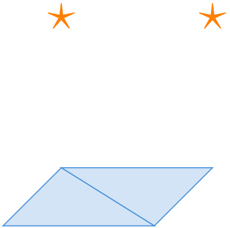
Shadow volumes: how much geometry?

We have the following scene with 2 triangles and 2 omnidirectional point light sources. How many triangles have to be generated for computing the shadow volumes?

1 10

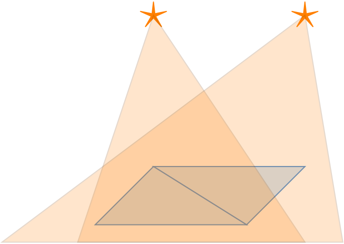
2 16

3 24



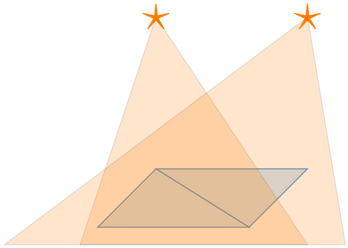
Shadow volumes: how many rendering passes?

We have the following scene with 2 triangles and 2 spot lights. How many rendering passes are required for rendering this scene with *stencil-buffer shadow volumes*?



Shadow mapping: how many rendering passes?

We have the following scene with 2 triangles and 2 spot lights. How many rendering passes are required for rendering this scene with *shadow mapping*?



Today

- Ray casting vs rasterisation
- The rendering equation!
- Ray tracing for shadows and transparency

Reading instructions

- Hughes et al:
  - 15.1–15.2.4, 15.4–15.4.1, 15.4.3
  - 7.8
  - 29

Rasterisation outline

```
for (each triangle)
{
  for (each pixel)
  {
    if (triangle covers pixel)
    {
      update z-buffer
      keep closest hit
    }
  }
}
```

- (Remember Lecture 1.)

Rasterisation outline

```
for (each triangle)
{
  for (each pixel)
  {
    if (triangle covers pixel)
    {
      update z-buffer
      keep closest hit
    }
  }
}
```

- (Remember Lecture 1.)
- Rasterisation, AKA *projective rendering*.

Rasterisation outline

```
for (each triangle)
{
  for (each pixel)
  {
    if (triangle covers pixel)
    {
      update z-buffer
      keep closest hit
    }
  }
}
```

- (Remember Lecture 1.)
- Rasterisation, AKA *projective rendering*.
- Each object is *projected* to screen and *rasterised*.

Ray casting vs rasterisation

Rasterisation

```
for (each triangle)
{
  for (each pixel)
  {
    if (triangle covers pixel)
    {
      update z-buffer
      keep closest hit
    }
  }
}
```

Ray casting

```
for (each pixel)
{
  shoot a ray through pixel
  for (each object)
  {
    if (ray hits object)
    {
      keep closest hit
    }
  }
}
```

Ray casting vs rasterisation

Rasterisation

```
for (each triangle)
{
  for (each pixel)
  {
    if (triangle covers pixel)
    {
      update z-buffer
      keep closest hit
    }
  }
}
```

Ray casting

```
for (each pixel)
{
  shoot a ray through pixel
  for (each object)
  {
    if (ray hits object)
    {
      keep closest hit
    }
  }
}
```

- We just need to swap the for-loops!

Evolution of Lara Croft


kanasoku.info

Polygon counts

- Tom Raider I (1996): 230
- TombRaider III (1998): 300
- Angel of Darkness (2003): 4 400
- Legend (2006): 9 800
- Underworld (2008): 32 816

Speaking of Lara Croft, Schleiner (2001) is worth reading too.

Rasterisation pros and cons

- Can stream over triangles instead of keeping whole data set in memory.

Rasterisation pros and cons

- Can stream over triangles instead of keeping whole data set in memory.
- Allows for better parallelism (and memory management).

Rasterisation pros and cons

- Can stream over triangles instead of keeping whole data set in memory.
- Allows for better parallelism (and memory management).
- But is restricted to *scan-convertible* primitives (i e, triangles).

Rasterisation pros and cons

- Can stream over triangles instead of keeping whole data set in memory.
- Allows for better parallelism (and memory management).
- But is restricted to *scan-convertible* primitives (i e, triangles).
- No natural handling of shadows, reflection, transparency, global illumination. (We can fake it, but it is hard work.)

Rasterisation pros and cons

- Can stream over triangles instead of keeping whole data set in memory.
- Allows for better parallelism (and memory management).
- But is restricted to *scan-convertible* primitives (i e, triangles).
- No natural handling of shadows, reflection, transparency, global illumination. (We can fake it, but it is hard work.)
- Each pixel may need to be touched many times for complex scenes.

Rasterisation pros and cons

- Can stream over triangles instead of keeping whole data set in memory.
- Allows for better parallelism (and memory management).
- But is restricted to *scan-convertible* primitives (i e, triangles).
- No natural handling of shadows, reflection, transparency, global illumination. (We can fake it, but it is hard work.)
- Each pixel may need to be touched many times for complex scenes.
- Ray casting used to be hard to implement in hardware, but with general-purpose GPUs, we are getting there (Lab 4!)

Rasterisation example

Drive Club, Evolution Studios

Ray tracing example

Source: “Now with vitamin R”

Ray casting

```
for every pixel
{
  cast ray from eye through pixel
  for every object
  {
    check intersection point with ray
    if closest
      keep it
  }

}
```

Ray casting

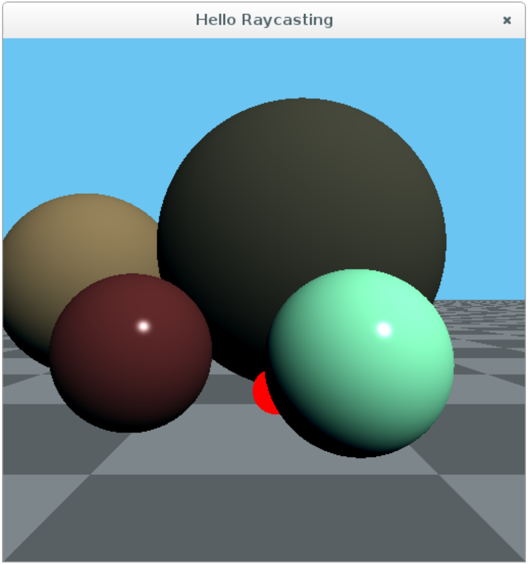
```
for every pixel
{
  cast ray from eye through pixel
  for every object
  {
    check intersection point with ray
    if closest
      keep it
  }
  compute pixel's colour
}
```

Ray casting

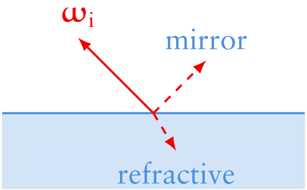
```
for every pixel
{
  cast ray from eye through pixel
  for every object
  {
    check intersection point with ray
    if closest
      keep it
  }
  compute pixel's colour
}
```

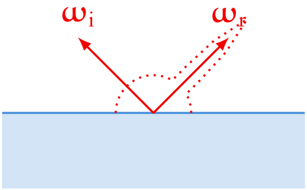
- *How do we compute the colour?*
- With a BSDF, as part of *the rendering equation*.
- Different ways of apprixmating the rendering equation leads to different rendering methods.

Ray casting example



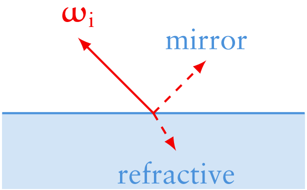
Two types of scattering

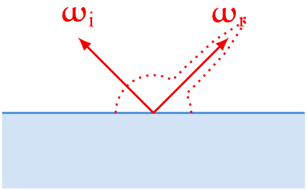




1 Mirror scattering

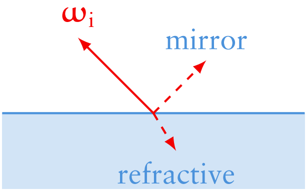
Two types of scattering

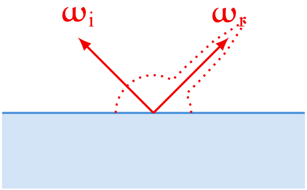




1. Mirror scattering
2. Snell-transmissive scattering (refraction)

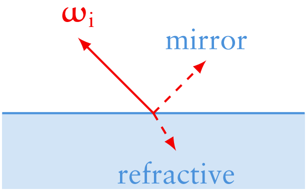
Two types of scattering

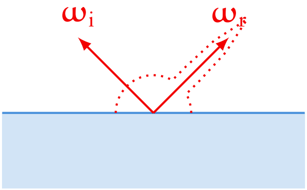




1. Mirror scattering
2. Snell-transmissive scattering (refraction)
3. Three! Three types of scattering!

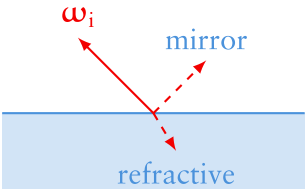
Two types of scattering

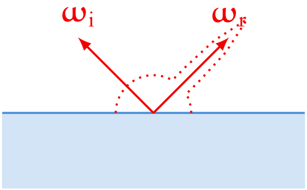




1. Mirror scattering
2. Snell-transmissive scattering (refraction)
3. Everything else

Two types of scattering





1. Impulse scattering: either mirror or refractive (i.e. Snell-transmissive).
2. Everything else (integrals).

Two types of lights

1. Point lights (impulses in incoming light).

Two types of lights

1. Point lights (impulses in incoming light).
2. Everything else (area lights $\implies$ integrals).

Two types of lights

1. Point lights (impulses in incoming light).
2. Everything else (area lights $\implies$ integrals).

- *Luminaire* = light source.

How to compute how light is distributed in the scene?

We are given

$\mathcal{M}$ all surface points in the scene,

How to compute how light is distributed in the scene?

We are given

$$\begin{aligned} \mathcal{M} &\quad \text{all surface points in the scene,} \\ f_{\mathrm{r}}(P, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}) &\quad \text{BRDF for all points } P \in \mathcal{M}, \end{aligned}$$

How to compute how light is distributed in the scene?

We are given

$\mathcal{M}$ all surface points in the scene,

$f_\mathrm{r}(P, \omega_\mathrm{i}, \omega_\mathrm{o})$ BRDF for all points $P \in \mathcal{M}$,

- i.e., how much light arriving at P from ω_i becomes light towards ω_o?

How to compute how light is distributed in the scene?

We are given

$\mathcal{M}$ all surface points in the scene,

$f_\mathrm{r}(P, \omega_\mathrm{i}, \omega_\mathrm{o})$ BRDF for all points $P \in \mathcal{M}$,

- i.e., how much light arriving at P from ω_i becomes light towards ω_o?

L^e a function describing the *illumination*

How to compute how light is distributed in the scene?

We are given

$\mathcal{M}$ all surface points in the scene,

$f_r(P, \omega_i, \omega_o)$ BRDF for all points $P \in \mathcal{M}$,

- i.e., how much light arriving at P from ω_i becomes light towards ω_o?

L^e a function describing the *illumination*

- $L^e(P, \omega_o) =$ how much radiance is emitted from P in direction ω_o

How to compute how light is distributed in the scene?

We are given

$\mathcal{M}$ all surface points in the scene,

$f_\mathrm{r}(P, \omega_\mathrm{i}, \omega_\mathrm{o})$ BRDF for all points $P \in \mathcal{M}$,

- i.e., how much light arriving at P from ω_i becomes light towards ω_o?

L^e a function describing the *illumination*

- $L^\mathrm{e}(P, \omega_\mathrm{o})$ = how much radiance is emitted from P in direction ω_o

R a ray-casting function

How to compute how light is distributed in the scene?

We are given

$\mathcal{M}$ all surface points in the scene,

$f_\mathrm{r}(P, \omega_\mathrm{i}, \omega_\mathrm{o})$ BRDF for all points $P \in \mathcal{M}$,

- i.e., how much light arriving at P from ω_i becomes light towards ω_o?

L^e a function describing the *illumination*

- $L^\mathrm{e}(P, \omega_\mathrm{o}) =$ how much radiance is emitted from P in direction ω_o

R a ray-casting function

- $R(P, \omega) =$ the first point hit when going from P toward ω

The reflectance equation

$$L^{\mathrm{ref}}(P, \omega_{\mathrm{o}}) = \int_{\omega_{\mathrm{i}} \in \mathbf{S}^2_+(P)} L(P, -\omega_{\mathrm{i}}) f_{\mathrm{r}}(P, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}) (\omega_{\mathrm{i}} \cdot \mathbf{n}_P) \, \mathrm{d}\omega_{\mathrm{i}}$$

The reflectance equation

$$L^{\mathrm{ref}}(P, \omega_{\mathrm{o}}) = \int_{\omega_{\mathrm{i}} \in \mathbf{S}^2_+(P)} L(P, -\omega_{\mathrm{i}}) f_{\mathrm{r}}(P, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}) (\omega_{\mathrm{i}} \cdot \mathbf{n}_P) \, \mathrm{d}\omega_{\mathrm{i}}$$

The radiance (light) reflected at P towards ω_{o}

The reflectance equation

$$L^{\text{ref}}(P, \omega_o) = \int_{\omega_i \in \mathbb{S}^2_+(P)} L(P, -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

The radiance (light) reflected at P towards ω_o is computed by

The reflectance equation

$$L^{\text{ref}}(P, \omega_{\text{o}}) = \int_{\omega_{\text{i}} \in \mathbf{S}^2_+(P)} L(P, -\omega_{\text{i}}) f_{\text{r}}(P, \omega_{\text{i}}, \omega_{\text{o}})(\omega_{\text{i}} \cdot \mathbf{n}_P)\, \mathrm{d}\omega_{\text{i}}$$

The radiance (light) reflected at P towards ω_{o} is computed by integrating (summing up)

The reflectance equation

$$L^{\mathrm{ref}}(P, \boldsymbol{\omega}_{\mathrm{o}}) = \int_{\boldsymbol{\omega}_{\mathrm{i}} \in \mathbf{S}^2_+(P)} L(P, -\boldsymbol{\omega}_{\mathrm{i}}) f_{\mathrm{r}}(P, \boldsymbol{\omega}_{\mathrm{i}}, \boldsymbol{\omega}_{\mathrm{o}}) (\boldsymbol{\omega}_{\mathrm{i}} \cdot \mathbf{n}_P) \, \mathrm{d}\boldsymbol{\omega}_{\mathrm{i}}$$

The radiance (light) reflected at P towards $\boldsymbol{\omega}_{\mathrm{o}}$ is computed by integrating (summing up) over all incoming light directions $\boldsymbol{\omega}_{\mathrm{i}}$ *above the surface* at P

The reflectance equation

$$L^{\text{ref}}(P, \omega_\text{o}) = \int_{\omega_\text{i} \in \mathbf{S}^2_+(P)} L(P, -\omega_\text{i}) f_\text{r}(P, \omega_\text{i}, \omega_\text{o}) (\omega_\text{i} \cdot \mathbf{n}_P) \, \mathrm{d}\omega_\text{i}$$

The radiance (light) reflected at P towards ω_o is computed by integrating (summing up) over all incoming light directions ω_i *above the surface* at P the contribution from each direction being

The reflectance equation

$$L^{\mathrm{ref}}(P, \omega_{\mathrm{o}}) = \int_{\omega_{\mathrm{i}} \in \mathbf{S}^2_+(P)} L(P, -\omega_{\mathrm{i}}) f_{\mathrm{r}}(P, \omega_{\mathrm{i}}, \omega_{\mathrm{o}})(\omega_{\mathrm{i}} \cdot \mathbf{n}_P)\, \mathrm{d}\omega_{\mathrm{i}}$$

The radiance (light) reflected at P towards ω_{o} is computed by integrating (summing up) over all incoming light directions ω_{i} *above the surface* at P the contribution from each direction being the incoming irradiance from $-\omega_{\mathrm{i}}$

The reflectance equation

$$L^{\mathrm{ref}}(P, \omega_{\mathrm{o}}) = \int_{\omega_{\mathrm{i}} \in \mathbf{S}^2_+(P)} L(P, -\omega_{\mathrm{i}}) f_{\mathrm{r}}(P, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}) (\omega_{\mathrm{i}} \cdot \mathbf{n}_P)\, \mathrm{d}\omega_{\mathrm{i}}$$

The radiance (light) reflected at P towards ω_{o} is computed by integrating (summing up) over all incoming light directions ω_{i} *above the surface* at P the contribution from each direction being the incoming irradiance from $-\omega_{\mathrm{i}}$ times the BRDF for these particular directions

The reflectance equation

$$L^{\mathrm{ref}}(P,\omega_{\mathrm{o}}) = \int_{\omega_{\mathrm{i}}\in\mathbf{S}^2_+(P)} L(P,-\omega_{\mathrm{i}}) f_{\mathrm{r}}(P,\omega_{\mathrm{i}},\omega_{\mathrm{o}})(\omega_{\mathrm{i}}\cdot\mathbf{n}_P)\,\mathrm{d}\omega_{\mathrm{i}}$$

The radiance (light) reflected at P towards ω_{o} is computed by integrating (summing up) over all incoming light directions ω_{i} *above the surface* at P the contribution from each direction being the incoming irradiance from $-\omega_{\mathrm{i}}$ times the BRDF for these particular directions times the weakening due to the incident angle.

The reflectance equation

$$L^{\mathrm{ref}}(P, \boldsymbol{\omega}_{\mathrm{o}}) = \int_{\boldsymbol{\omega}_{\mathrm{i}} \in \mathbf{S}^2_+(P)} L(P, -\boldsymbol{\omega}_{\mathrm{i}}) f_{\mathrm{r}}(P, \boldsymbol{\omega}_{\mathrm{i}}, \boldsymbol{\omega}_{\mathrm{o}}) (\boldsymbol{\omega}_{\mathrm{i}} \cdot \mathbf{n}_P)\, \mathrm{d}\boldsymbol{\omega}_{\mathrm{i}}$$

The radiance (light) reflected at P towards $\boldsymbol{\omega}_{\mathrm{o}}$ is computed by integrating (summing up) over all incoming light directions $\boldsymbol{\omega}_{\mathrm{i}}$ *above the surface* at P the contribution from each direction being the incoming irradiance from $-\boldsymbol{\omega}_{\mathrm{i}}$ times the BRDF for these particular directions times the weakening due to the incident angle.

Impulses (point lights and mirror reflections) can't really be integrated like this, but let's pretend we can stuff them inside the integral too for now (just make a sum instead of an integral).

The rendering equation (take 1)

$$
\begin{aligned}
L(P, \omega_\mathrm{o}) &= L^\mathrm{e}(P, \omega_\mathrm{o}) + L^\mathrm{ref}(P, \omega_\mathrm{o}) \\
&= L^\mathrm{e}(P, \omega_\mathrm{o}) + \int_{\omega_\mathrm{i} \in \mathbf{S}^2_+(P)} L(P, -\omega_\mathrm{i}) f_\mathrm{r}(P, \omega_\mathrm{i}, \omega_\mathrm{o})(\omega_\mathrm{i} \cdot \mathbf{n}_P)\, \mathrm{d}\omega_\mathrm{i}
\end{aligned}
$$

The rendering equation (take 1)

$$
\begin{aligned}
L(P, \boldsymbol{\omega}_\mathrm{o}) &= L^\mathrm{e}(P, \boldsymbol{\omega}_\mathrm{o}) + L^\mathrm{ref}(P, \boldsymbol{\omega}_\mathrm{o}) \\
&= L^\mathrm{e}(P, \boldsymbol{\omega}_\mathrm{o}) + \int_{\boldsymbol{\omega}_\mathrm{i} \in \mathbf{S}^2_+(P)} L(P, -\boldsymbol{\omega}_\mathrm{i}) f_\mathrm{r}(P, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) (\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n}_P) \, \mathrm{d}\boldsymbol{\omega}_\mathrm{i}
\end{aligned}
$$

- This is "the *reflectance-only* rendering equation."

The rendering equation (take 1)

$$L(P,\omega_\mathrm{o}) = L^\mathrm{e}(P,\omega_\mathrm{o}) + L^\mathrm{ref}(P,\omega_\mathrm{o})$$
$$= L^\mathrm{e}(P,\omega_\mathrm{o}) + \int_{\omega_\mathrm{i}\in \mathbf{S}^2_+(P)} L(P,-\omega_\mathrm{i}) f_\mathrm{r}(P,\omega_\mathrm{i},\omega_\mathrm{o})(\omega_\mathrm{i}\cdot \mathbf{n}_P)\,\mathrm{d}\omega_\mathrm{i}$$

- This is “the *reflectance-only* rendering equation.”
- Reflectance + emission (from light sources).

The rendering equation (take 1)

$$L(P, \omega_o) = L^e(P, \omega_o) + L^{ref}(P, \omega_o)$$
$$= L^e(P, \omega_o) + \int_{\omega_i \in \mathbf{S}^2_+(P)} L(P, -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

- This is “the *reflectance-only* rendering equation.”
- Reflectance + emission (from light sources).
- We are given L^e and f_r, and want to compute the unknown function L.

The rendering equation (take 1)

$$
\begin{aligned}
L(P, \omega_o) &= L^e(P, \omega_o) + L^{ref}(P, \omega_o) \\
&= L^e(P, \omega_o) + \int_{\omega_i \in \mathbf{S}^2_+(P)} L(P, -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i
\end{aligned}
$$

- This is "the *reflectance-only* rendering equation."
- Reflectance + emission (from light sources).
- We are given L^e and f_r, and want to compute the unknown function L.
- But L appears on both sides of the equation.

The rendering equation (take 1)

$$L(P, \omega_o) = L^e(P, \omega_o) + L^{ref}(P, \omega_o)$$
$$= L^e(P, \omega_o) + \int_{\omega_i \in S_+^2(P)} L(P, -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

- This is "the *reflectance-only* rendering equation."
- Reflectance + emission (from light sources).
- We are given L^e and f_r, and want to compute the unknown function L.
- But L appears on both sides of the equation.
- This is an *integral equation* and is difficult to solve exactly.

The rendering equation (take 1)

$$L(P, \omega_o) = L^e(P, \omega_o) + L^{ref}(P, \omega_o)$$
$$= L^e(P, \omega_o) + \int_{\omega_i \in \mathbf{S}^2_+(P)} L(P, -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

- This is "the *reflectance-only* rendering equation."
- Reflectance + emission (from light sources).
- We are given L^e and f_r, and want to compute the unknown function L.
- But L appears on both sides of the equation.
- This is an *integral equation* and is difficult to solve exactly.
- We can only hope to approximate the solution.

The transport equation

- Trouble with the rendering equation:

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),
  - we need to compute $L(P, -\omega_i)$ (light that arrives at P).

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_\mathrm{o})$ (light that reflects from P),
  - we need to compute $L(P, -\omega_\mathrm{i})$ (light that arrives at P).
- But light that arrives along $-\omega_\mathrm{i}$ must have departed from some other point $Q \in \mathcal{M}$ in direction $-\omega_\mathrm{i}$.

$$L(P, -\omega_\mathrm{i}) = L\big(R(P, \omega_\mathrm{i}), -\omega_\mathrm{i}\big)$$

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),
  - we need to compute $L(P, -\omega_i)$ (light that arrives at P).
- But light that arrives along $-\omega_i$ must have departed from some other point $Q \in \mathcal{M}$ in direction $-\omega_i$.

$$L(P, -\omega_i) = L\big(R(P, \omega_i),\ -\omega_i\big)$$

- In other words, the light arriving at P from $-\omega_i$

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),
  - we need to compute $L(P, -\omega_i)$ (light that arrives at P).
- But light that arrives along $-\omega_i$ must have departed from some other point $Q \in \mathcal{M}$ in direction $-\omega_i$.

$$L(P, -\omega_i) = L\big(R(P, \omega_i), -\omega_i\big)$$

- In other words, the light arriving at P from $-\omega_i$

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),
  - we need to compute $L(P, -\omega_i)$ (light that arrives at P).
- But light that arrives along $-\omega_i$ must have departed from some other point $Q \in \mathcal{M}$ in direction $-\omega_i$.

$$L(P, -\omega_i) = L\big(R(P, \omega_i), -\omega_i\big)$$

- In other words, the light arriving at P from $-\omega_i$ is the same as the light

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),
  - we need to compute $L(P, -\omega_i)$ (light that arrives at P).
- But light that arrives along $-\omega_i$ must have departed from some other point $Q \in \mathcal{M}$ in direction $-\omega_i$.

$$L(P, -\omega_i) = L\big(R(P, \omega_i), -\omega_i\big)$$

- In other words, the light arriving at P from $-\omega_i$ is the same as the light that leaves from the closest surface point when looking to ω_i

The transport equation

- Trouble with the rendering equation:
  - To compute $L(P, \omega_o)$ (light that reflects from P),
  - we need to compute $L(P, -\omega_i)$ (light that arrives at P).
- But light that arrives along $-\omega_i$ must have departed from some other point $Q \in \mathcal{M}$ in direction $-\omega_i$.

$$L(P, -\omega_i) = L\big(R(P, \omega_i), -\omega_i\big)$$

- In other words, the light arriving at P from $-\omega_i$ is the same as the light that leaves from the closest surface point when looking to ω_i with direction $-\omega_i$.

The rendering equation (take 2)

$$L(P, \omega_o) = L^e(P, \omega_o) + L^{ref}(P, \omega_o)$$

$$= L^e(P, \omega_o) + \int_{\omega_i \in S^2_+(P)} L(R(P, \omega_i), -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

The rendering equation (take 2)

$$
\begin{aligned}
L(P, \omega_o) &= L^e(P, \omega_o) + L^{\mathrm{ref}}(P, \omega_o) \\
&= L^e(P, \omega_o) + \int_{\omega_i \in S^2_+(P)} L(R(P, \omega_i), -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i
\end{aligned}
$$

- Are we happier?

The rendering equation (take 2)

$$L(P,\omega_o) = L^e(P,\omega_o) + L^{ref}(P,\omega_o)$$
$$= L^e(P,\omega_o) + \int_{\omega_i \in S^2_+(P)} L(R(P,\omega_i), -\omega_i) f_r(P,\omega_i,\omega_o)(\omega_i \cdot \mathbf{n}_P)\,d\omega_i$$

- Are we happier?
- Yes, because now the surface radiance function $L(P,\omega_o)$ is expressed in terms of

The rendering equation (take 2)

$$
\begin{aligned}
L(P,\omega_o) &= L^e(P,\omega_o) + L^{ref}(P,\omega_o) \\
&= L^e(P,\omega_o) + \int_{\omega_i \in S^2_+(P)} L(R(P,\omega_i), -\omega_i) f_r(P,\omega_i,\omega_o)(\omega_i \cdot \mathbf{n}_P)\,d\omega_i
\end{aligned}
$$

- Are we happier?
- Yes, because now the surface radiance function $L(P,\omega_o)$ is expressed in terms of
  - *known* light sources L^e and

The rendering equation (take 2)

$$L(P,\boldsymbol{\omega}_\mathrm{o}) = L^\mathrm{e}(P,\boldsymbol{\omega}_\mathrm{o}) + L^\mathrm{ref}(P,\boldsymbol{\omega}_\mathrm{o})$$
$$= L^\mathrm{e}(P,\boldsymbol{\omega}_\mathrm{o}) + \int_{\boldsymbol{\omega}_\mathrm{i}\in S^2_+(P)} L(R(P,\boldsymbol{\omega}_\mathrm{i}), -\boldsymbol{\omega}_\mathrm{i}) f_\mathrm{r}(P,\boldsymbol{\omega}_\mathrm{i},\boldsymbol{\omega}_\mathrm{o})(\boldsymbol{\omega}_\mathrm{i}\cdot\mathbf{n}_P)\,\mathrm{d}\boldsymbol{\omega}_\mathrm{i}$$

- Are we happier?
- Yes, because now the surface radiance function $L(P,\boldsymbol{\omega}_\mathrm{o})$ is expressed in terms of
  - *known* light sources L^e and
  - an integral over *known* BRDFs for all surface points f_r,

The rendering equation (take 2)

$$
\begin{aligned}
L(P,\boldsymbol{\omega}_\mathrm{o}) &= L^\mathrm{e}(P,\boldsymbol{\omega}_\mathrm{o}) + L^\mathrm{ref}(P,\boldsymbol{\omega}_\mathrm{o}) \\
&= L^\mathrm{e}(P,\boldsymbol{\omega}_\mathrm{o}) + \int_{\boldsymbol{\omega}_\mathrm{i}\in\mathbb{S}^2_+(P)} L(R(P,\boldsymbol{\omega}_\mathrm{i}),-\boldsymbol{\omega}_\mathrm{i}) f_\mathrm{r}(P,\boldsymbol{\omega}_\mathrm{i},\boldsymbol{\omega}_\mathrm{o})(\boldsymbol{\omega}_\mathrm{i}\cdot\mathbf{n}_P)\,\mathrm{d}\boldsymbol{\omega}_\mathrm{i}
\end{aligned}
$$

- Are we happier?
- Yes, because now the surface radiance function $L(P,\boldsymbol{\omega}_\mathrm{o})$ is expressed in terms of
  - *known* light sources L^e and
  - an integral over *known* BRDFs for all surface points f_r,
  - a (*known*) ray-casting function R,

The rendering equation (take 2)

$$L(P, \omega_o) = L^e(P, \omega_o) + L^{ref}(P, \omega_o)$$
$$= L^e(P, \omega_o) + \int_{\omega_i \in S^2_+(P)} L(\textcolor{red}{R(P, \omega_i)}, -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

- Are we happier?
- Yes, because now the surface radiance function $L(P, \omega_o)$ is expressed in terms of
  - *known* light sources L^e and
  - an integral over *known* BRDFs for all surface points f_r,
  - a (*known*) ray-casting function R,
  - and the radiance from another point. (Just a recursive function.)

What about $R(P, \omega)$?

$$L(P, \omega_\mathrm{o}) =$$
$$L^\mathrm{e}(P, \omega_\mathrm{o}) + \int_{\omega_\mathrm{i} \in \mathbb{S}^2_+(P)} L(R(P, \omega_\mathrm{i}), -\omega_\mathrm{i}) f_\mathrm{r}(P, \omega_\mathrm{i}, \omega_\mathrm{o})(\omega_\mathrm{i} \cdot \mathbf{n}_P)\, \mathrm{d}\omega_\mathrm{i}$$

- L^e is easy.

What about $R(P, \omega)$?

$$L(P, \boldsymbol{\omega}_\mathrm{o}) = L^\mathrm{e}(P, \boldsymbol{\omega}_\mathrm{o}) + \int_{\boldsymbol{\omega}_\mathrm{i} \in \mathbf{S}^2_+(P)} L(R(P, \boldsymbol{\omega}_\mathrm{i}), -\boldsymbol{\omega}_\mathrm{i}) f_\mathrm{r}(P, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) (\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n}_P) \, \mathrm{d}\boldsymbol{\omega}_\mathrm{i}$$

- L^e is easy.
- We covered f_r in Lecture 3.

What about $R(P, \omega)$?

$$L(P, \boldsymbol{\omega}_\mathrm{o}) = L^\mathrm{e}(P, \boldsymbol{\omega}_\mathrm{o}) + \int_{\boldsymbol{\omega}_\mathrm{i} \in \mathbf{S}^2_+(P)} L(R(P, \boldsymbol{\omega}_\mathrm{i}), -\boldsymbol{\omega}_\mathrm{i}) f_\mathrm{r}(P, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) (\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n}_P) \, \mathrm{d}\boldsymbol{\omega}_\mathrm{i}$$

- L^e is easy.
- We covered f_r in Lecture 3.
- The last factor is just the cosine of the incident angle.

What about $R(P, \omega)$?

$$L(P, \omega_o) = L^e(P, \omega_o) + \int_{\omega_i \in S^2_+(P)} L(R(P, \omega_i), -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

- L^e is easy.
- We covered f_r in Lecture 3.
- The last factor is just the cosine of the incident angle.
- The remaining item is the ray-casting function $R(P, \omega)$

What about $R(P, \omega)$?

$$L(P, \omega_o) = L^e(P, \omega_o) + \int_{\omega_i \in \mathbb{S}^2_+(P)} L(R(P, \omega_i), -\omega_i) f_r(P, \omega_i, \omega_o)(\omega_i \cdot \mathbf{n}_P)\, d\omega_i$$

- L^e is easy.
- We covered f_r in Lecture 3.
- The last factor is just the cosine of the incident angle.
- The remaining item is the ray-casting function $R(P, \omega)$
- (…and that integral).

What about $R(P, \omega)$?

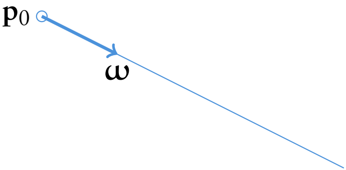
$$L(P, \boldsymbol{\omega}_\mathrm{o}) =$$
$$L^\mathrm{e}(P, \boldsymbol{\omega}_\mathrm{o}) + \int_{\boldsymbol{\omega}_\mathrm{i} \in \mathbb{S}^2_+(P)} L(R(P, \boldsymbol{\omega}_\mathrm{i}), -\boldsymbol{\omega}_\mathrm{i}) f_\mathrm{r}(P, \boldsymbol{\omega}_\mathrm{i}, \boldsymbol{\omega}_\mathrm{o}) (\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{n}_P)\, \mathrm{d}\boldsymbol{\omega}_\mathrm{i}$$

- L^e is easy.
- We covered f_r in Lecture 3.
- The last factor is just the cosine of the incident angle.
- The remaining item is the ray-casting function $R(P, \boldsymbol{\omega})$
- (…and that integral).
- For a ray with starting point P and direction $\boldsymbol{\omega}$, how to compute which surface point it hits first?

Intersection testing

- How to represent a ray?
- Given start point $\mathbf{p}_0$ and (normalised) direction $\boldsymbol{\omega}$,

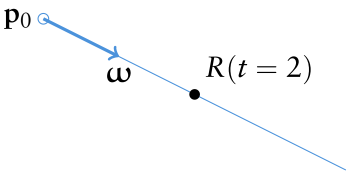
$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}.$$



Intersection testing

- How to represent a ray?
- Given start point $\mathbf{p}_0$ and (normalised) direction $\boldsymbol{\omega}$,

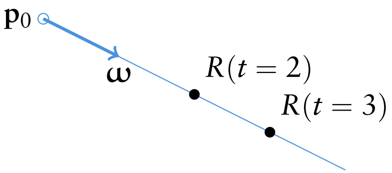
$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}.$$



Intersection testing

- How to represent a ray?
- Given start point $\mathbf{p}_0$ and (normalised) direction $\boldsymbol{\omega}$,

$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}.$$



Plane representations

- Infinite plane defined by point $\mathbf{p}_0$ in the plane and normal $\mathbf{n}$,

Plane representations

- Infinite plane defined by point $\mathbf{p}_0$ in the plane and normal $\mathbf{n}$,
- or as the normal $\mathbf{n} = (a, b, c)$ and a distance d from the origin.

Plane representations

- Infinite plane defined by point $\mathbf{p}_0$ in the plane and normal $\mathbf{n}$,
- or as the normal $\mathbf{n} = (a, b, c)$ and a distance d from the origin.
- Implicit plane equation

$$\begin{aligned} H(\mathbf{p}) &= ax + by + cz + d = 0 \\ &= \mathbf{n} \cdot \mathbf{p} + d = 0 \end{aligned}$$

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
    - "Generates points on the ray."

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
  - "Generates points on the ray."
  - Easy to find out where a point on the ray is, given distance from the start.

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
  - “Generates points on the ray.”
  - Easy to find out where a point on the ray is, given distance from the start.
  - Hard to verify that any point is on the ray.

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
  - "Generates points on the ray."
  - Easy to find out where a point on the ray is, given distance from the start.
  - Hard to verify that any point is on the ray.
- Plane equation is *implicit*: $H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
  - "Generates points on the ray."
  - Easy to find out where a point on the ray is, given distance from the start.
  - Hard to verify that any point is on the ray.
- Plane equation is *implicit*: $H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$
  - Easy to verify if point $\mathbf{p}$ is in plane.

Explicit vs implicit

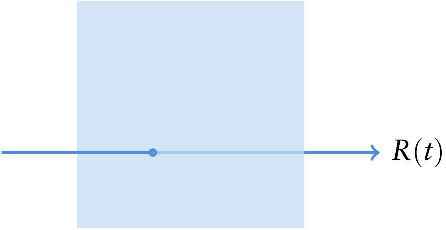
- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
  - "Generates points on the ray."
  - Easy to find out where a point on the ray is, given distance from the start.
  - Hard to verify that any point is on the ray.
- Plane equation is *implicit*: $H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$
  - Easy to verify if point $\mathbf{p}$ is in plane.
  - Does not generate points.

Explicit vs implicit

- Ray equation is *explicit*: $R(t) = \mathbf{p}_0 + t\boldsymbol{\omega}$
  - "Generates points on the ray."
  - Easy to find out where a point on the ray is, given distance from the start.
  - Hard to verify that any point is on the ray.
- Plane equation is *implicit*: $H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$
  - Easy to verify if point $\mathbf{p}$ is in plane.
  - Does not generate points.
- Now: how to check if ray intersects plane?

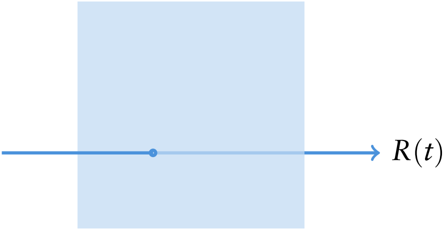
Ray–plane intersection

- Intersection iff both equations are satisifed.



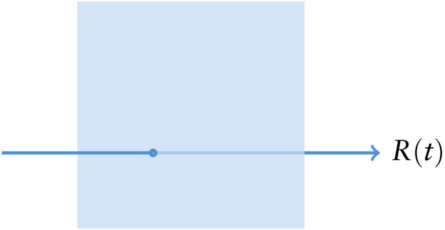
Ray–plane intersection

- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).



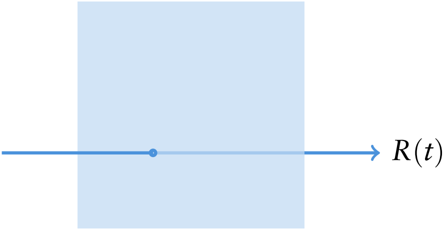
Ray–plane intersection

- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- “Which t generates a point that satisfies the plane equation?”



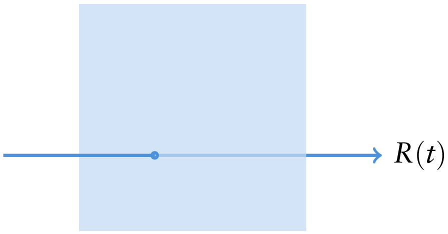
Ray–plane intersection

- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- “Which t generates a point that satisfies the plane equation?”



Ray–plane intersection

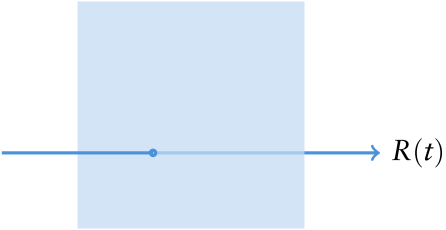
- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- “Which t generates a point that satisfies the plane equation?”



$$R(t) = \mathbf{p}_0 + \boldsymbol{\omega} t$$

Ray–plane intersection

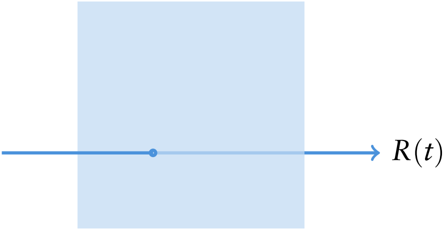
- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- "Which t generates a point that satisfies the plane equation?"



$$R(t) = \mathbf{p}_0 + \boldsymbol{\omega} t$$
$$H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$$

Ray–plane intersection

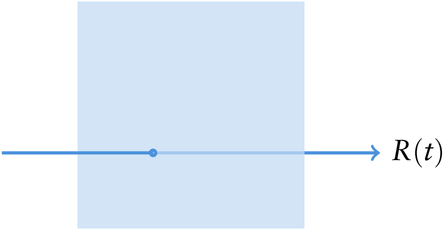
- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- "Which t generates a point that satisfies the plane equation?"



$$R(t) = \mathbf{p}_0 + \boldsymbol{\omega} t$$
$$H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$$
$$\mathbf{n} \cdot (\mathbf{p}_0 + \boldsymbol{\omega} t) + d = 0$$

Ray–plane intersection

- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- "Which t generates a point that satisfies the plane equation?"



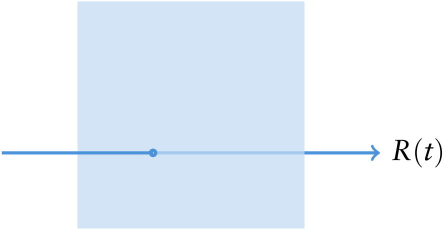
$$R(t) = \mathbf{p}_0 + \boldsymbol{\omega} t$$

$$H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$$

$$\mathbf{n} \cdot (\mathbf{p}_0 + \boldsymbol{\omega} t) + d = 0$$

$$t = -\frac{d + \mathbf{n} \cdot \mathbf{p}_0}{\mathbf{n} \cdot \boldsymbol{\omega}}$$

Ray–plane intersection



- Intersection iff both equations are satisifed.
- So: insert explicit ray equation into implicit plane equation and solve for t (i.e., how far along the ray they intersect).
- "Which t generates a point that satisfies the plane equation?"

$$R(t) = \mathbf{p}_0 + \boldsymbol{\omega} t$$

$$H(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d = 0$$

$$\mathbf{n} \cdot (\mathbf{p}_0 + \boldsymbol{\omega} t) + d = 0$$

$$t = -\frac{d + \mathbf{n} \cdot \mathbf{p}_0}{\mathbf{n} \cdot \boldsymbol{\omega}}$$

voilà!

What do we do with t?

- If we have more than one object, we'll run intersection tests for the ray with all of them.

What do we do with t?

- If we have more than one object, we'll run intersection tests for the ray with all of them.
- Check if t is closer than previous intersection along this ray:

$$R(t) < t_{\text{closest}}.$$

What do we do with t?

- If we have more than one object, we'll run intersection tests for the ray with all of them.
- Check if t is closer than previous intersection along this ray:

$$R(t) < t_{\text{closest}}.$$

- Check that t is not behind us:

$$R(t) > t_{\text{min}}.$$

What do we do after verifying t?

- Finally, we also need to compute the normal at the point $R(t)$, for computing *lighting and secondary rays*.

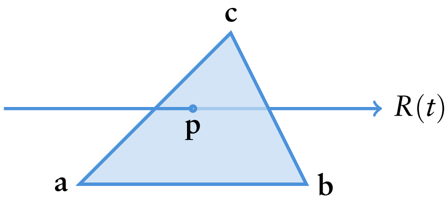
What do we do after verifying t?

- Finally, we also need to compute the normal at the point $R(t)$, for computing *lighting and secondary rays*.
- For plane, normal is constant for all points.

Ray–triangle intersection

Two options:

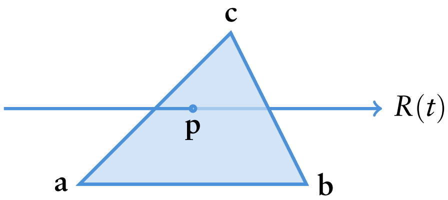
1. First ray–plane intersection, then test if $R(t)$ is inside triangle (as with rasterisation).



Ray–triangle intersection

Two options:

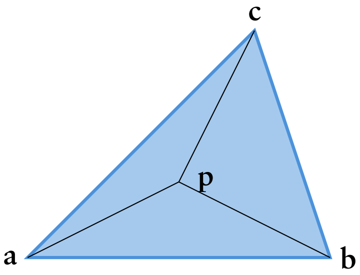
1. First ray–plane intersection, then test if $R(t)$ is inside triangle (as with rasterisation).
2. Better: use barycentric coordinates.



Triangle representation

- Implicit triangle-patch equation

$$
\begin{aligned}
T(\alpha, \beta, \gamma) &= \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c} \\
\alpha + \beta + \gamma &= 1 \\
0 \leq \alpha, \beta, \gamma &\leq 1
\end{aligned}
$$



Just the same as when doing texture interpolation, etc.

Simplified triangle representation

- Since $\alpha + \beta + \gamma = 1$ we can use that $\alpha = 1 - \beta - \gamma$ and get rid of α.

$$T(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

Simplified triangle representation

- Since $\alpha + \beta + \gamma = 1$ we can use that $\alpha = 1 - \beta - \gamma$ and get rid of α.

$$
\begin{aligned}
T(\alpha, \beta, \gamma) &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\
T(\beta, \gamma) &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\
0 \leq \beta - \gamma &\leq 1
\end{aligned}
$$

Ray–triangle intersection

- So, does our ray intersect the triangle?

$$T(\beta, \gamma) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{1}$$

I think this version is conceptually simpler than the book's.

Ray–triangle intersection

- So, does our ray intersect the triangle?

$$T(\beta, \gamma) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{1}$$

I think this version is conceptually simpler than the book's.

Ray–triangle intersection

- So, does our ray intersect the triangle?

$$T(\beta, \gamma) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{1}$$

$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \tag{2}$$

I think this version is conceptually simpler than the book's.

Ray–triangle intersection

- So, does our ray intersect the triangle?

$$T(\beta,\gamma) = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \tag{1}$$

$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega} = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \tag{2}$$

$$t\boldsymbol{\omega} - \beta(\mathbf{b}-\mathbf{a}) - \gamma(\mathbf{c}-\mathbf{a}) = \mathbf{a} - \mathbf{p}_0 \tag{3}$$

- (Write out as three equations, for $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]$ and $\mathbf{p}_0 = [p_x, p_y, p_z]$.)

I think this version is conceptually simpler than the book's.

Ray–triangle intersection

- So, does our ray intersect the triangle?

$$T(\beta,\gamma) = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \tag{1}$$

$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega} = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \tag{2}$$

$$t\boldsymbol{\omega} - \beta(\mathbf{b}-\mathbf{a}) - \gamma(\mathbf{c}-\mathbf{a}) = \mathbf{a} - \mathbf{p}_0 \tag{3}$$

- (Write out as three equations, for $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]$ and $\mathbf{p}_0 = [p_x, p_y, p_z]$.)
- Solve

$$\begin{bmatrix} \omega_x & A_x - B_x & A_x - C_x \\ \omega_y & A_y - B_y & A_y - C_y \\ \omega_z & A_z - B_z & A_z - C_z \end{bmatrix} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} A_x - p_x \\ A_y - p_y \\ A_z - p_z \end{bmatrix}$$

I think this version is conceptually simpler than the book's.

Ray–triangle intersection

- So, does our ray intersect the triangle?

$$T(\beta,\gamma) = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \tag{1}$$
$$R(t) = \mathbf{p}_0 + t\boldsymbol{\omega} = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \tag{2}$$
$$t\boldsymbol{\omega} - \beta(\mathbf{b}-\mathbf{a}) - \gamma(\mathbf{c}-\mathbf{a}) = \mathbf{a} - \mathbf{p}_0 \tag{3}$$

- (Write out as three equations, for $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]$ and $\mathbf{p}_0 = [p_x, p_y, p_z]$.)
- Solve

$$\begin{bmatrix} \omega_x & A_x - B_x & A_x - C_x \\ \omega_y & A_y - B_y & A_y - C_y \\ \omega_z & A_z - B_z & A_z - C_z \end{bmatrix} \begin{bmatrix} t \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} A_x - p_x \\ A_y - p_y \\ A_z - p_z \end{bmatrix}$$

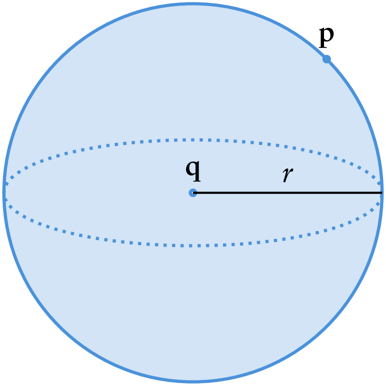
- Intersection if $\beta + \gamma < 1$ and $\beta, \gamma > 0$ and $t > 0$.

I think this version is conceptually simpler than the book's.

Sphere representation

- Implicit sphere equation:
- Sphere centred at $\mathbf{q}$ with radius r. Point $\mathbf{p}$ is on sphere iff

$$(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}) = r^2.$$



Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$
\begin{aligned}
(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}) &= r^2 \\
((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) \cdot ((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) &= r^2
\end{aligned}
$$

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}) = r^2$$
$$((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) \cdot ((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) = r^2$$

- Using $\mathbf{p} - \mathbf{q} \equiv \mathbf{v}$, this boils down to

$$(\mathbf{v} \cdot \mathbf{v} - r^2) + t(2\boldsymbol{\omega} \cdot \mathbf{v}) + t^2(\boldsymbol{\omega} \cdot \boldsymbol{\omega}) = 0$$

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q}) = r^2$$
$$((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q})\cdot((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) = r^2$$

- Using $\mathbf{p}-\mathbf{q} \equiv \mathbf{v}$, this boils down to

$$(\mathbf{v}\cdot\mathbf{v} - r^2) + t(2\boldsymbol{\omega}\cdot\mathbf{v}) + t^2(\boldsymbol{\omega}\cdot\boldsymbol{\omega}) = 0$$

- which is quadratic in t.

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}) = r^2$$
$$((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) \cdot ((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) = r^2$$

- Using $\mathbf{p} - \mathbf{q} \equiv \mathbf{v}$, this boils down to

$$(\mathbf{v} \cdot \mathbf{v} - r^2) + t(2\boldsymbol{\omega} \cdot \mathbf{v}) + t^2(\boldsymbol{\omega} \cdot \boldsymbol{\omega}) = 0$$

- which is quadratic in t.
- Quadratic polynomials can have zero, one, or two solutions.

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$\begin{aligned}(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q}) &= r^2 \\ ((\mathbf{p}_0+t\boldsymbol{\omega})-\mathbf{q})\cdot((\mathbf{p}_0+t\boldsymbol{\omega})-\mathbf{q}) &= r^2\end{aligned}$$

- Using $\mathbf{p}-\mathbf{q}\equiv\mathbf{v}$, this boils down to

$$(\mathbf{v}\cdot\mathbf{v}-r^2)+t(2\boldsymbol{\omega}\cdot\mathbf{v})+t^2(\boldsymbol{\omega}\cdot\boldsymbol{\omega})=0$$

- which is quadratic in t.
- Quadratic polynomials can have zero, one, or two solutions.
- What does that mean, and which t do we choose?

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$\begin{aligned}(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q}) &= r^2\\ ((\mathbf{p}_0+t\boldsymbol{\omega})-\mathbf{q})\cdot((\mathbf{p}_0+t\boldsymbol{\omega})-\mathbf{q}) &= r^2\end{aligned}$$

- Using $\mathbf{p}-\mathbf{q}\equiv\mathbf{v}$, this boils down to

$$(\mathbf{v}\cdot\mathbf{v}-r^2)+t(2\boldsymbol{\omega}\cdot\mathbf{v})+t^2(\boldsymbol{\omega}\cdot\boldsymbol{\omega})=0$$

- which is quadratic in t.
- Quadratic polynomials can have zero, one, or two solutions.
- What does that mean, and which t do we choose?
  - No solution: ray misses sphere.

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q}) = r^2$$
$$((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q})\cdot((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) = r^2$$

- Using $\mathbf{p}-\mathbf{q} \equiv \mathbf{v}$, this boils down to

$$(\mathbf{v}\cdot\mathbf{v} - r^2) + t(2\boldsymbol{\omega}\cdot\mathbf{v}) + t^2(\boldsymbol{\omega}\cdot\boldsymbol{\omega}) = 0$$

- which is quadratic in t.
- Quadratic polynomials can have zero, one, or two solutions.
- What does that mean, and which t do we choose?
  - No solution: ray misses sphere.
  - One solution: ray *just* touches sphere.

Ray–sphere intersection

- Insert explicit ray equation into implicit sphere equation and solve for t.

$$(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q}) = r^2$$
$$((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q})\cdot((\mathbf{p}_0 + t\boldsymbol{\omega}) - \mathbf{q}) = r^2$$

- Using $\mathbf{p}-\mathbf{q} \equiv \mathbf{v}$, this boils down to

$$(\mathbf{v}\cdot\mathbf{v} - r^2) + t(2\boldsymbol{\omega}\cdot\mathbf{v}) + t^2(\boldsymbol{\omega}\cdot\boldsymbol{\omega}) = 0$$

- which is quadratic in t.
- Quadratic polynomials can have zero, one, or two solutions.
- What does that mean, and which t do we choose?
  - No solution: ray misses sphere.
  - One solution: ray *just* touches sphere.
  - Two solutions: ray pierces sphere. Choose smallest positive t.

Solving a quadratic equation

- Quadratic polynomial $at^2 + bt + c = 0$ can be solved with the standard “quadratic formula”

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Solving a quadratic equation

- Quadratic polynomial $at^2 + bt + c = 0$ can be solved with the standard “quadratic formula”

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

- Ray–sphere intersection is

$$(\mathbf{v} \cdot \mathbf{v} - r^2) + t(2\boldsymbol{\omega} \cdot \mathbf{v}) + t^2(\boldsymbol{\omega} \cdot \boldsymbol{\omega}) = 0$$

so in this case

$$a = \boldsymbol{\omega} \cdot \boldsymbol{\omega},$$
$$b = 2\boldsymbol{\omega} \cdot \mathbf{v},$$
$$c = \mathbf{v} \cdot \mathbf{v} - r^2.$$

Solving a quadratic equation

- Quadratic polynomial $at^2 + bt + c = 0$ can be solved with the standard "quadratic formula"

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

- Ray–sphere intersection is

$$(\mathbf{v} \cdot \mathbf{v} - r^2) + t(2\boldsymbol{\omega} \cdot \mathbf{v}) + t^2(\boldsymbol{\omega} \cdot \boldsymbol{\omega}) = 0$$

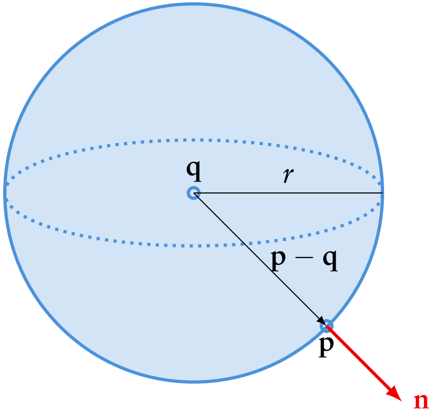
so in this case

$$a = \boldsymbol{\omega} \cdot \boldsymbol{\omega},$$
$$b = 2\boldsymbol{\omega} \cdot \mathbf{v},$$
$$c = \mathbf{v} \cdot \mathbf{v} - r^2.$$

- (Useful in Lab 4.)

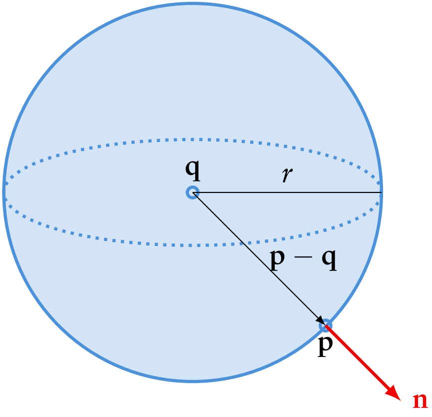
Sphere normal

- ... and then we need the normal at $\mathbf{p}$ too.



Sphere normal

- …and then we need the normal at $\mathbf{p}$ too.
- $\mathbf{n} = \mathbf{p} - \mathbf{q} / \left\|\mathbf{p} - \mathbf{q}\right\|$



Light transport

- Light travels through medium

Light transport

- Light travels through medium
- Light hits object and scatters:

Light transport

- Light travels through medium
- Light hits object and scatters:
  - reflected

Light transport

- Light travels through medium
- Light hits object and scatters:
  - reflected
  - absorbed

Light transport

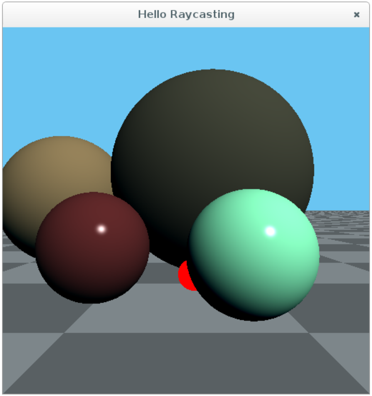
- Light travels through medium
- Light hits object and scatters:
  - reflected
  - absorbed
  - transmitted

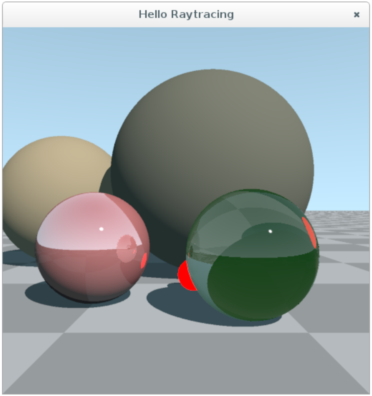
Light transport

- Light travels through medium
- Light hits object and scatters:
  - reflected
  - absorbed
  - transmitted
- Sooner or later, light reaches eye.

Ray casting vs ray tracing

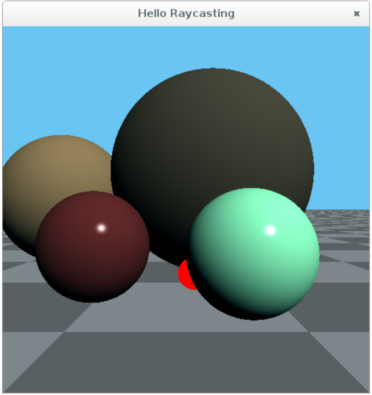
- Ray casting: stop at first intersection.

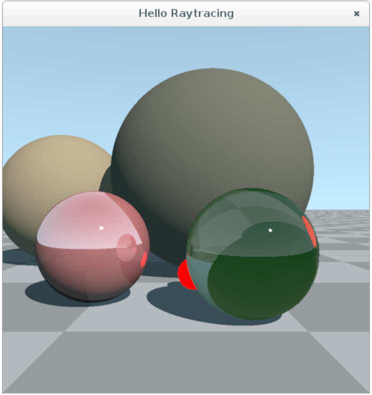




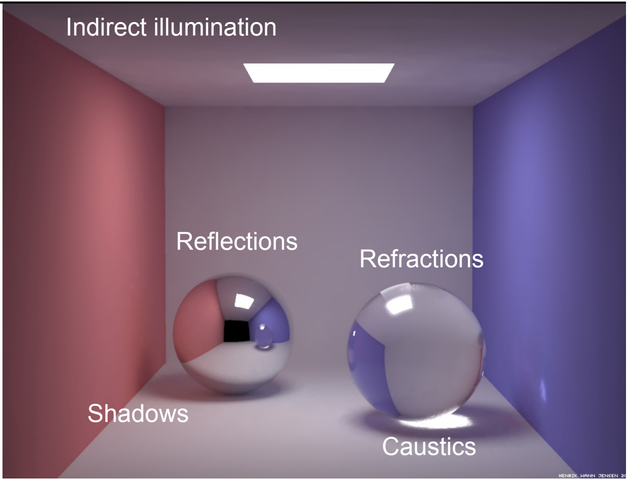
Ray casting vs ray tracing

- Ray casting: stop at first intersection.
- Ray tracing: secondary rays for reflections, shadows, etc.



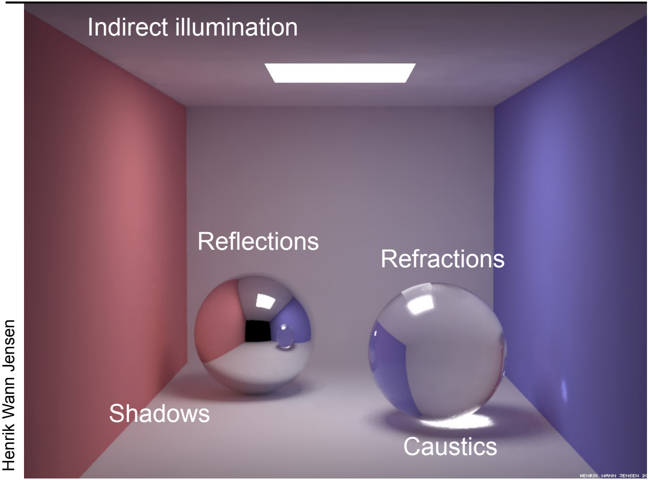


Physically based rendering



Henrik Wann Jensen

Physically based rendering



With ray tracing, we get (hard) shadows, reflections, refractions, but *not* indirect illumination and caustics.

Shadow rays

- AKA “shadow feelers”.

Shadow rays

- AKA “shadow feelers”.
- After hitting an object at $\mathbf{p}$, shoot shadow ray $\mathbf{p} + t(\mathbf{q} - \mathbf{p})$ towards light source at location $\mathbf{q}$.

Shadow rays

- AKA "shadow feelers".
- After hitting an object at $\mathbf{p}$, shoot shadow ray $\mathbf{p} + t(\mathbf{q} - \mathbf{p})$ towards light source at location $\mathbf{q}$.
- If this rays intersects an object, $\mathbf{p}$ is in shadow.

Shadow rays

- AKA “shadow feelers”.
- After hitting an object at $\mathbf{p}$, shoot shadow ray $\mathbf{p} + t(\mathbf{q} - \mathbf{p})$ towards light source at location $\mathbf{q}$.
- If this rays intersects an object, $\mathbf{p}$ is in shadow.
- Precision issues (z-fighting): start ray at $\mathbf{p} + \epsilon(\mathbf{q} - \mathbf{p})$ instead of $\mathbf{p}$.

Reflection rays

- If BRDF contains mirror-like reflection,

Reflection rays

- If BRDF contains mirror-like reflection,
- shoot ray in mirror direction from $\mathbf{p} + \boldsymbol{\omega}_\mathrm{r}$.

Reflection rays

- If BRDF contains mirror-like reflection,
- shoot ray in mirror direction from $\mathbf{p} + \boldsymbol{\omega}_{\mathrm{r}}$.
- See where this ray hits, and compute the colour for that point.

Reflection rays

- If BRDF contains mirror-like reflection,
- shoot ray in mirror direction from $\mathbf{p} + \boldsymbol{\omega}_{\mathrm{r}}$.
- See where this ray hits, and compute the colour for that point.
- Add that colour to the colour of $\mathbf{p}$.

```
R = 0.5;
colour = 0;
for (all light sources)
{
  colour += brdf(light_direction, view_direction, normal) * cosine;
}
mirror = reflect(view_direction, normal);
reflected_colour = raycast(point + mirror * eps, mirror);
colour = R * reflected_colour + (1 - R) * colour;
```

Reflection rays

- If BRDF contains mirror-like reflection,
- shoot ray in mirror direction from $\mathbf{p} + \boldsymbol{\omega}_r$.
- See where this ray hits, and compute the colour for that point.
- Add that colour to the colour of $\mathbf{p}$.

```
R = 0.5;
colour = 0;
for (all light sources)
{
  colour += brdf(light_direction, view_direction, normal) * cosine;
}
mirror = reflect(view_direction, normal);
reflected_colour = raycast(point + mirror * eps, mirror);
colour = R * reflected_colour + (1 - R) * colour;
```

- How does this fit with the rendering equation?

Reflection rays

- If BRDF contains mirror-like reflection,
- shoot ray in mirror direction from $\mathbf{p} + \boldsymbol{\omega}_{\mathrm{r}}$.
- See where this ray hits, and compute the colour for that point.
- Add that colour to the colour of $\mathbf{p}$.

```
R = 0.5;
colour = 0;
for (all light sources)
{
  colour += brdf(light_direction, view_direction, normal) * cosine;
}
mirror = reflect(view_direction, normal);
reflected_colour = raycast(point + mirror * eps, mirror);
colour = R * reflected_colour + (1 - R) * colour;
```

- How does this fit with the rendering equation?
- *We just took one more sample from the integral.*

Refraction rays

- If surface has transparency (not just a BRDF, but a full BSDF),

Refraction rays

- If surface has transparency (not just a BRDF, but a full BSDF),
- shoot ray into surface, with a new direction depending on the refraction indices.

Refraction rays

- If surface has transparency (not just a BRDF, but a full BSDF),
- shoot ray into surface, with a new direction depending on the refraction indices.
- Add colour from that ray…

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,
- E eye,

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,
- E eye,
- D diffuse reflection,

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,
- E eye,
- D diffuse reflection,
- G glossy reflection,

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,
- E eye,
- D diffuse reflection,
- G glossy reflection,
- S (perfectly) specular reflection.

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,
- E eye,
- D diffuse reflection,
- G glossy reflection,
- S (perfectly) specular reflection.
- LDE: light leaves source, scatters from a diffuse surface, arrives at eye.

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- L light source,
- E eye,
- D diffuse reflection,
- G glossy reflection,
- S (perfectly) specular reflection.
- LDE: light leaves source, scatters from a diffuse surface, arrives at eye.
    - That's how light *actually travels*.

Classification of light-transport paths

- Convenient notation for describing ray casting, ray tracing, and the many other simulation methods that exist (due to Heckbert, Veach, and others):
- *L* light source,
- *E* eye,
- *D* diffuse reflection,
- *G* glossy reflection,
- *S* (perfectly) specular reflection.
- *LDE*: light leaves source, scatters from a diffuse surface, arrives at eye.
    - That's how light *actually travels*.
- In ray casting, we go backwards: *EDL*.

More expressive classification

- We can add *regular expression* syntax to make it more expressive.

More expressive classification

- We can add *regular expression* syntax to make it more expressive.
- For example:

More expressive classification

- We can add *regular expression* syntax to make it more expressive.
- For example:
  - $(D|G)$: either diffuse or glossy reflection

More expressive classification

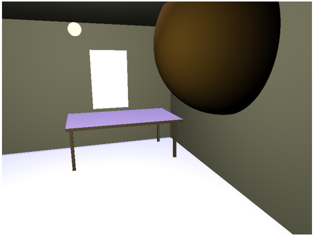
- We can add *regular expression* syntax to make it more expressive.
- For example:
  - $(D|G)$: either diffuse or glossy reflection
  - D^+: one or several diffuse reflections

More expressive classification

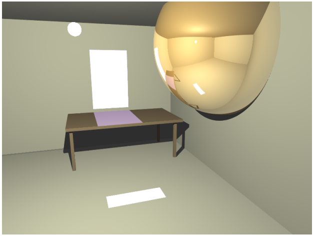
- We can add *regular expression* syntax to make it more expressive.
- For example:
  - $(D|G)$: either diffuse or glossy reflection
  - D^+: one or several diffuse reflections
  - D^*: zero or more diffuse reflections

A taxonomy of tracing

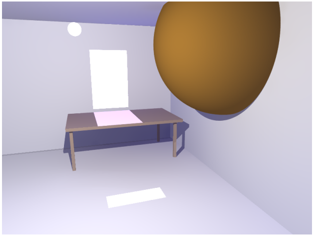
- Ray casting $E(D|G)L$



Ray casting



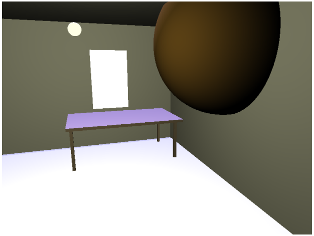
Ray tracing



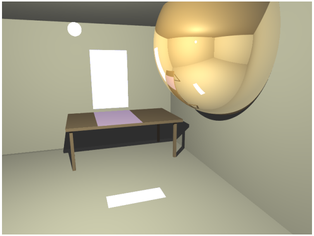
Radiosity

A taxonomy of tracing

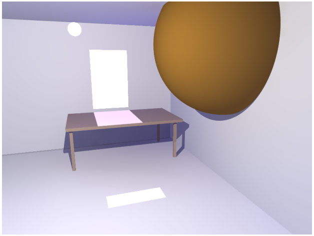
- Ray casting $E(D|G)L$
- Ray tracing $E[S^*](D|G)L$



Ray casting



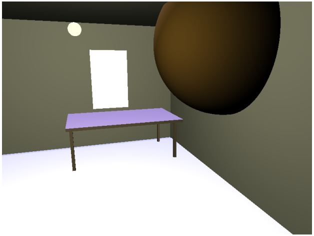
Ray tracing



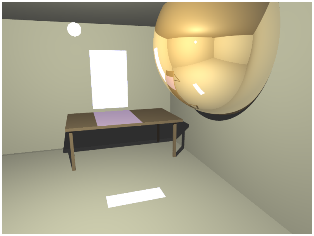
Radiosity

A taxonomy of tracing

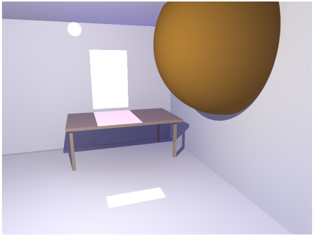
- Ray casting $E(D|G)L$
- Ray tracing $E[S^*](D|G)L$
- Path tracing $E[((D|G|S)^+)D|G]L$ (lecture 8)



Ray casting



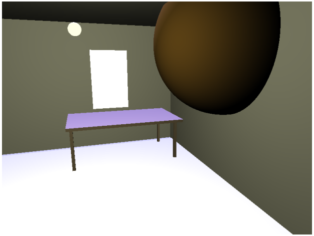
Ray tracing



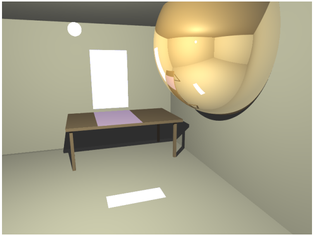
Radiosity

A taxonomy of tracing

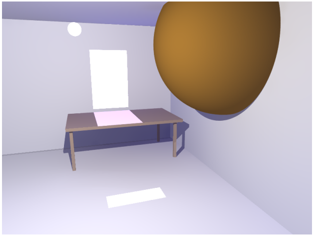
- Ray casting $E(D|G)L$
- Ray tracing $E[S^*](D|G)L$
- Path tracing $E[((D|G|S)^+)D|G]L$ (lecture 8)
- Radiosity ED^*L (lecture 8)



Ray casting



Ray tracing



Radiosity

Summary

- Ray casting is "just rasterisation with a different order of the for loops"

Summary

- Ray casting is "just rasterisation with a different order of the for loops"
- The rendering equation: all-purpose solution for computing what light comes from a point

Summary

- Ray casting is "just rasterisation with a different order of the for loops"
- The rendering equation: all-purpose solution for computing what light comes from a point
  - Different rendering techniques approximate it in different ways

Summary

- Ray casting is “just rasterisation with a different order of the for loops”
- The rendering equation: all-purpose solution for computing what light comes from a point
  - Different rendering techniques approximate it in different ways
- Ray–object intersections: planes, triangles, spheres.

Summary

- Ray casting is “just rasterisation with a different order of the for loops”
- The rendering equation: all-purpose solution for computing what light comes from a point
  - Different rendering techniques approximate it in different ways
- Ray–object intersections: planes, triangles, spheres.
- Ray tracing: emit secondary rays to render shadows and transparency (incl refraction)

Summary

- Ray casting is "just rasterisation with a different order of the for loops"
- The rendering equation: all-purpose solution for computing what light comes from a point
  - Different rendering techniques approximate it in different ways
- Ray–object intersections: planes, triangles, spheres.
- Ray tracing: emit secondary rays to render shadows and transparency (incl refraction)
- Classification of light-transport paths to describe different types of rendering methods

Next lecture: more reflections, transmission, materials

Time and place

- Mon 21 Nov, 13.15–15.00
- T-141

Reading material

- We'll stick to Chapter 29, mostly.

References

Anne-Marie Schleiner (2001). “Does Lara Croft Wear Fake Polygons? Gender and Gender-Role Subversion in Computer Adventure Games”. In: *Leonardo* 34.3, pp. 221–226.