The Road So Far...
Direct / Local Illumination

- Apply a lighting model for a fragment. We consider only the light source.
Indirect Lighting

- Not only does the light source illuminate the scene. The scene itself illuminates the scene.
Cornell Box

- This scene is called the **Cornell Box scene**.
- It is often used to test global illumination.
- Left wall **red**, right wall **green**.
- Other walls are **white / gray**.
- Light source in the ceiling.
- Objects inside the room.

By LockRickard:
[http://lockrikard.deviantart.com/art/Minecraft-Cornell-Box-356384680](http://lockrikard.deviantart.com/art/Minecraft-Cornell-Box-356384680)
Indirect Lighting

- The light bounces around more than once...
Indirect Lighting

- The light bounces around more than once...

- What is wrong with this picture?
Indirect Lighting

- It is more like...
Global Illumination

• Take into account both direct (from the light source) and indirect lighting.
Global Illumination

- Take into account both direct (from the light source) and indirect lighting.
- Alternatively, think of all the surfaces as light sources.
Path Tracing (with Direct Illum.)

- Algorithm based on ray trace rendering.
Path Tracing (with Direct Illum.)

- Algorithm based on ray trace rendering.
- For each hit, send random ray(s) to sample indirect illumination from the scene.
Path Tracing (with Direct Illum.)

• Did just that, got this result:

• That is correct, it just one sample.
Path Tracing (with Direct Illum.)

• Because the rays are random, we need to sample a lot of them, and average all results.
Path Tracing (with Direct Illum.)

- That was just one bounce. We can do more.
Path Tracing (with Direct Illum.)

- There are differences.

80 samples, 1 bounce versus 3 bounces
Path Tracing

- There are other path tracing techniques also.
Path Tracing

- There are other path tracing techniques also.
- Without direct illumination:
  - Only bounce, until we reach the light source.
  - Light source intensity should be more than 1.
  - Direct into the light source at a random bounce.
Path Tracing

- There are other path tracing techniques also.
- Without direct illumination:
  - Only bounce, until we reach the light source.
  - Light source intensity should be more than 1.
  - Direct into the light source at a random bounce.
- Create a number of random rays at the first bounce.
Path Tracing

- With direct illumination
Path Tracing

- Without direct illumination (direct rays to light, after a random number of bounces).
Path Tracing

• Examples:
  • Thorough project and overview: https://github.com/erichlof/THREE.js-PathTracing-Renderer
  • Nice interactive version: http://madebyevan.com/webgl-path-tracing/
  • Non-parallel implementation on CPU: https://github.com/hunterloftis/pathtracer
  • Shadertoy search (slow on laptops): https://www.shadertoy.com/results?query=tag%3Dpathtracer
Photon Mapping

- First shoot rays from the light source.

Construction phase.
Photon Mapping

- First shoot rays from the light source.
- Create a map of photons, that those rays distribute.

Construction phase.
Photon Mapping

- Then shoot rays from the camera, find nearest photons for the hit points.

Gathering phase.
Photon Mapping

- Illumination of the surface can be estimated, by considering the nearest photons, divided by the area of the minimum sphere covering those.
Photon Mapping

- Illumination of the surface can be estimated, by considering the nearest photons, divided by the area of the minimum sphere covering those.
- Some illumination (direct, specular) can be calculated without the photon map.
Photon Mapping

- Illumination of the surface can be estimated, by considering the nearest photons, divided by the area of the minimum sphere covering those.

- Some illumination (direct, specular) can be calculated without the photon map.

- Gathering phase is much slower, then the construction.
Photon Mapping

- Illumination of the surface can be estimated, by considering the nearest photons, divided by the area of the minimum sphere covering those.
- Some illumination (direct, specular) can be calculated without the photon map.
- Gathering phase is much slower, then the construction.
- How to store the photon map for fast nearest neighbour search?
More Cool Stuff

- Photon Mapping (In Estonian) by Hendrik Eerikson:
  + http://tume-maailm.pri.ee/ylikool/CG/2017/slides/12/PhotonMapping.png

- Photon mapping:
  https://github.com/erlandranvinge/photons

- Real-time GPU Path Tracing in Quake 2
  https://www.youtube.com/watch?v=x19sIlR0qU

- WebGL Path Tracing
  http://madebyevan.com/webgl-path-tracing/
What important things you learned?

What more would you like to know?

Next time: Global Illumination continued
The Rendering Equation

- Mathematical formulation for the visible color (outgoing light) of the surface.

\[
L_{out}(x, \omega_o) = 
L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i
\]
The Rendering Equation

\[ L_{out}(x, \omega_o) = \]
\[ = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
The Rendering Equation

\[ L_{out}(x, \omega_o) = \]

\[ = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
- \( \omega_o \) – Viewing (outgoing light) angle / vector
The Rendering Equation

\[ L_{out}(x, \omega_o) = \]
\[ = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
- \( \omega_o \) – Viewing (outgoing light) angle / vector
- \( \omega_i \) – Incoming light angle / vector
The Rendering Equation

\[ L_{out}(x, \omega_o) = \]
\[ = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
- \( \omega_o \) – Viewing (outgoing light) angle / vector
- \( \omega_i \) – Incoming light angle / vector
- \( L_{out} \) – Amount of outgoing light
The Rendering Equation

\[ L_{out}(x, \omega_o) = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
- \( \omega_o \) – Viewing (outgoing light) angle / vector
- \( \omega_i \) – Incoming light angle / vector
- \( L_{out} \) – Amount of outgoing light
- \( L_{emit} \) – Amount of emitted light
The Rendering Equation

\[ L_{out}(x, \omega_o) = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
- \( \omega_o \) – Viewing (outgoing light) angle / vector
- \( \omega_i \) – Incoming light angle / vector
- \( L_{out} \) – Amount of outgoing light
- \( L_{emit} \) – Amount of emitted light
- \( f_{brdf} \) – Amount of reflected light (BRDF)

Bidirectional Reflectance Distribution Function
The Rendering Equation

\[ L_{\text{out}}(x, \omega_o) = L_{\text{emit}}(x, \omega_o) + \int_{\Omega} f_{\text{brdf}}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]

- \( x \) – Surface point
- \( \omega_o \) – Viewing (outgoing light) angle / vector
- \( \omega_i \) – Incoming light angle / vector
- \( L_{\text{out}} \) – Amount of outgoing light
- \( L_{\text{emit}} \) – Amount of emitted light
- \( f_{\text{brdf}} \) – Amount of reflected light (BRDF)
- \( L_i(\omega_i \cdot n) \) – Amount of incoming light per surface
The Rendering Equation

- Some surfaces can emit light on their own (light sources eg car headlights, fluorescent materials)

\[ L_{out}(x, \omega_o) = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]
The Rendering Equation

- Outgoing light depends on the light coming in from all directions. Integrate over the hemisphere.

\[
L_{out}(x, \omega_o) = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i
\]
The Rendering Equation

- Light reaching the surface unit is proportional to

\[
\cos(\angle(\omega_i, n)) = (\omega_i \cdot n)
\]

\[
L_{out}(x, \omega_o) = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i
\]
The Rendering Equation

- Reflected light depends on the incoming light $L_i$ and the material properties represented by $f_{brdf}$.
- BRDF – Bidirectional Reflectance Distribution Function

\[ L_{out}(x, \omega_o) = \]
\[ = L_{emit}(x, \omega_o) + \int_{\Omega} f_{brdf}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i \]
The Rendering Equation

- BRDF for Phong would be:

\[ f_{brdf} = M_{\text{diffuse}} + M_{\text{specular}} \cdot (v \cdot r)^{\text{shininess}} \]
The Rendering Equation

• BRDF for Phong would be:

\[ f_{brdf} = M_{diffuse} + M_{specular} \cdot \left( v \cdot r \right)^{\text{shininess}} \]

• Notice, that we are also multiplying the specular term with \( \omega_i \cdot n \).
The Rendering Equation

• BRDF for Phong would be:

\[ f_{\text{brdf}} = M_{\text{diffuse}} + M_{\text{specular}} \cdot (v \cdot r)^{\text{shininess}} \]

• Notice, that we are also multiplying the specular term with \( \omega_i \cdot n \).

• The specular highlight intensity also depends on the angle light is reaching the surface.
The Rendering Equation

- BRDF for Phong would be:

\[ f_{brdf} = M_{\text{diffuse}} + M_{\text{specular}} \cdot (v \cdot r)^{\text{shininess}} \]

- Notice, that we are also multiplying the specular term with \( \omega_i \cdot n \).
- The specular highlight intensity also depends on the angle light is reaching the surface.
- Or:

\[ f_{brdf} = M_{\text{diffuse}} + \frac{M_{\text{specular}} \cdot (v \cdot r)^{\text{shininess}}}{n \cdot l} \]

See this later in the Disney's BRDF Explorer
Radiosity

- Divides our geometry into (small) patches.
Radiosity

- Divides our geometry into (small) patches.
- Tries to approximate the rendering equation by replacing the integral with a finite sum.
Radiosity

- Divides our geometry into (small) patches.
- Tries to approximate the rendering equation by replacing the integral with a finite sum.
- Assumes that each patch radiates light equally in all directions (diffuse reflection).

The BRDF is constant
Radiosity

- Divides our geometry into (small) patches.
- Tries to approximate the rendering equation by replacing the integral with a finite sum.
- Assumes that each patch radiates light equally in all directions (diffuse reflection).

\[
L_{\text{emit}}(x, \omega_o) + \int_{\Omega} f_{\text{brdf}}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i
\]

\[
L_{\text{emit}}(x, \omega_o) + \sum_j f_{\text{brdf}}(x, \omega_j, \omega_o) \cdot L_i(x_j) \cdot (\omega_j \cdot n)
\]
Radiosity

- Divides our geometry into (small) patches.
- Tries to approximate the rendering equation by replacing the integral with a finite sum.
- Assumes that each patch radiates light equally in all directions (diffuse reflection).

\[
L_{\text{emit}}(x, \omega_o) + \int_\Omega f_{\text{brdf}}(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) \, d\omega_i
\]

\[
L_{\text{emit}}(x, \omega_o) + \sum_j f_{\text{brdf}}(x, \omega_j, \omega_o) \cdot L_i(x_j) \cdot (\omega_j \cdot n)
\]

- For each patch \( k \):
  \[
  E_k + \rho_k \sum_j F_{k, j} \cdot L_j
  \]
Radiosity

- For each patch $k$:

$$ L_k = E_k + \rho_k \sum_j F_{k,j} \cdot L_j $$

How much this patch emits light
Radiosity

- For each patch $k$:

$$L_k = E_k + \rho_k \sum_j F_{k,j} \cdot L_j$$

How much this patch emits light

How much this patch reflects light (its color)
Radiosity

- For each patch $k$:

$$L_k = E_k + \rho_k \sum_j F_{k,j} \cdot L_j$$

- How much this patch emits light
- How much this patch reflects light (its color)
- View factor
  How much this patch $k$ receives light from patch $j$
Radiosity

• For each patch $k$:

$$L_k = E_k + \rho_k \sum_j F_{k,j} \cdot L_j$$

- How much this patch emits light
- How much this patch reflects light (its color)
- View factor
- How much this patch $k$ receives light from patch $j$
- How much patch $j$ emits light
Radiosity

- For each patch $k$:

$$L_k = E_k + \rho_k \sum_j F_{k,j} \cdot L_j$$

$$F = \begin{pmatrix}
0 & F_{0,1} & F_{0,2} & \ldots & F_{0,k} \\
F_{1,0} & 0 & F_{1,2} & \ldots & F_{1,k} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
F_{k,0} & F_{k,1} & F_{k,2} & \ldots & 0
\end{pmatrix}$$

View factors in each row / column need to sum up to 1.
We can not have more than 100% energy entering or exiting a patch!
Radiosity

- Vector equation for the entire scene

\[ L = E + \rho F \cdot L \]

\[
\begin{pmatrix}
L_0 \\
L_1 \\
\vdots \\
L_k
\end{pmatrix} =
\begin{pmatrix}
E_0 \\
E_1 \\
\vdots \\
E_k
\end{pmatrix} + \rho
\begin{pmatrix}
0 & F_{0,1} & F_{0,2} & \cdots & F_{0,k} \\
F_{1,0} & 0 & F_{1,2} & \cdots & F_{1,k} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
F_{k,0} & F_{k,1} & F_{k,2} & \cdots & 0
\end{pmatrix}
\begin{pmatrix}
L_0 \\
L_1 \\
\vdots \\
L_k
\end{pmatrix}
\]
Radiosity

- Vector equation for the entire scene

\[ L = E + \rho F \cdot L \]
\[ -E = \rho F \cdot L - L \]
Radiosity

- Vector equation for the entire scene

\[ L = E + \rho F \cdot L \]
\[ -E = \rho F \cdot L - L \]
\[ -E = (\rho F - I)L \]
Radiosity

- Vector equation for the entire scene

\[ L = E + \rho F \cdot L \]
\[ -E = \rho F \cdot L - L \]
\[ -E = (\rho F - I) L \]
\[ E = (I - \rho F) L \]
Radiosity

• Vector equation for the entire scene

\[ L = E + \rho F \cdot L \]
\[-E = \rho F \cdot L - L \]
\[-E = (\rho F - I) L \]
\[ E = (I - \rho F) L \]
\[ L = (I - \rho F)^{-1} E \]
Radiosity

- Vector equation for the entire scene

\[
L = E + \rho F \cdot L \\
-E = \rho F \cdot L - L \\
-E = (\rho F - I) L \\
E = (I - \rho F) L \\
L = (I - \rho F)^{-1} E
\]

- Unfortunately, finding the solution this way is \( O(k^3) \)

\( k \) – Number of patches...
Radiosity

- Jacobi iteration method solves $Ax = b$, when $A$ is strictly diagonally dominant.
Radiosity

- Jacobi iteration method solves $Ax = b$, when $A$ is strictly **diagonally dominant**.

Diagonal element has greater absolute value, then the sum of absolute values of other elements in a row (or column).

$$\left| a_{i,i} \right| > \sum_{i \neq j} \left| a_{i,j} \right|$$
Radiosity

• Jacobi iteration method solves $Ax = b$, when $A$ is strictly diagonally dominant.

\[(I - \rho F) L = E\]
Radiosity

- Jacobi iteration method solves $Ax = b$, when $A$ is strictly diagonally dominant.

$$\left( I - \rho F \right) L = E$$

$$A = \begin{pmatrix}
0 & F_{0,1} & F_{0,2} & \ldots & F_{0,k} \\
F_{1,0} & 0 & F_{1,2} & \ldots & F_{1,k} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
F_{k,0} & F_{k,1} & F_{k,2} & \ldots & 0
\end{pmatrix} \begin{pmatrix}
L_0 \\
L_1 \\
\vdots \\
L_k
\end{pmatrix} = \begin{pmatrix}
E_0 \\
E_1 \\
\vdots \\
E_k
\end{pmatrix}$$
Radiosity

- Jacobi iteration method solves $Ax = b$, when $A$ is strictly diagonally dominant.

\[
(I - \rho F) L = E
\]

- Is $(I - \rho F)$ strictly diagonally dominant? When?
Radiosity

• Jacobi iteration method finds an approximation:

\[ L^{(r+1)} = D^{-1} \left( E - R \cdot L' \right) \]
Radiosity

- Jacobi iteration method finds an approximation:

\[
L^{(r+1)} = D^{-1}(E - R \cdot L^r)
\]

\(D\) – diagonal of \((I - \rho F)\)

What is that value?
Radiosity

- Jacobi iteration method finds an approximation:

\[ L^{(r+1)} = D^{-1} (E - R \cdot L') \]

\[ R = (I - \rho F) - D = -\rho F \]

The "remainder" of \( A \) (\( A \) minus the diagonal)
Radiosity

- Jacobi iteration method finds an approximation:

\[ L^{(r+1)} = D^{-1} (E - R \cdot L^r) \]

\[ L^{(r+1)} = E - R \cdot L^r \]

\[ L^{(r+1)} = E + \rho F \cdot L^r \]

This is exactly, what we started with...
Radiosity

- Jacobi iteration method finds an approximation:

\[
L^{(r+1)} = D^{-1}(E - R \cdot L^r)
\]

\[
L^{(r+1)} = E - R \cdot L^r
\]

\[
L^{(r+1)} = E + \rho F \cdot L^r
\]

This is exactly, what we started with...

\(L^r\) – \(r\)-th approximation

\(L^0\) – initial random guess
Radiosity

• Difficult part is to **find the view factors** for all the patches.

\[ O(k^2) \]

Naive way

\[ O(k \cdot \log k) \]

With BSP tree
Radiosity

- Difficult part is to **find the view factors for all the patches**. One way to do it, is to:
  1) Create an unit hemisphere around the patch
  2) Project another patch onto the hemisphere
  3) Then project that onto the base of the hemisphere
  4) Ratio between the area of the last projection and the area of the circular base, is the view factor.
Radiosity

- Because finding the actual hemisphere projection for each patch is again complicated, we can create a hemicube.

- We create a rectangular grid on the hemicube and pre-calculate the hemisphere projections of each cell.

- Then we project other patches onto the cube, and see what cells they cover.
Global Illumination

- Different combinations of the algorithms, also many algorithms for real time approximations.
- Global Illumination in a Nutshell:
  http://www.thepolygoners.com/tutorials/GIIntro/GIIntro.htm
- Voxel Octree Cone Tracing – Real time, by Nvidia, 2011:
- Lightmass – Unreal Engine 4:
  https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/Lightmass/index.html
BRDF

- Disney BRDF explorer, to see different BRDF-s (incl. Lambert and Phong) in action: http://www.disneyanimation.com/technology/brdf.html

- Simon's Tech Blog, post about BRDF-s, includes interactive demos: http://simonstechblog.blogspot.com/2011/12/microfacet-brdf.html
What ideas you had today?

What more would you like to know?

Next time: Shadows, Conclusion!