PHOTOREALISM

Al Chee by Robert Bechtle (1962)

Photorealism is the genre of painting based on using cameras and photographs to gather visual information and then from this creating a painting that appears to be photographic. The term is primarily applied to paintings from the United States art movement that began in the late 1960s and early 1970s.


Photorealism by Robert Bechtle (1965)

PHOTOREALISM

View of New York from Staten Island on the Ferry by Richard Estes

Local illumination models cannot generate images like these

Images © Steven Collins

LECTURE OVERVIEW

Global Illumination
  - Reflectance
  - Transmission
  - Environment Maps
  - A brief look at: Radiosity

POV Ray Tips:
  - Area Lights
  - Depth of Field

THE RENDERING EQUATION

A mathematical representation of light transport in the scene, describing the amount of light going from any point $x'$ to another point $x'$:

$$ I(x, x') = g(x, x') I(x, x') + \frac{1}{4} \rho(x, x', x''') I(x', x''') \ dx''' $$

- $I(x, x')$ = intensity of light passing from $x$ to $x'$
- $g(x, x') = \begin{cases} 1 & \text{if $x$ and $x'$ are not mutually visible} \\ \frac{1}{4} \rho(x, x', x'') & \text{otherwise} \end{cases}$
- $\rho(x, x', x''')$ = bi-directional reflectance factor for light passing from $x'''$ to $x$ by reflecting off $x'$$
- $S = \text{all surfaces in the scene}$

Please don't worry, you don't have to remember this equation!
Two point Light Transport

\[ I(x, x') = g(x, x') [e(x, x') + \int \rho(x, x', x'') I(x', x'') dx''] \]

Geometry Factor: Is B visible from A?

\[ g(x, x') = \begin{cases} 0 & \text{if } x \text{ and } x' \text{ are not mutually visible} \\ \frac{1}{r^2} & \text{where } r \text{ is the distance between } x \text{ and } x' \end{cases} \]

Emittance: Is B' a light source?

\[ I(x, x') = g(x, x') [e(x, x') + \int \rho(x, x', x'') I(x', x'') dx''] \]

Effect of Distance: Inverse Square Law

From a point light source, light is radiated equally in all directions (analogous to a growing sphere). As the light gets further, the sphere has a bigger area across which the same light is distributed thus it gets dimmer.

The amount by which it gets dimmer is \( \frac{1}{r^2} \), where \( r \) is the distance the light has travelled (the radius of the sphere).

Reflectance of B: How much incoming light does B reflect out in A’s direction

\[ I(x, x') = g(x, x') [e(x, x') + \int \rho(x, x', x'') I(x', x'') dx''] \]
Practically we are interested in how much and in what directions light is reflected or scattered from a point: this is a feature of the material.

The Rendering Equation implies a possibly infinite recursive path of light transfer. We need to make some simplifying assumptions in order to solve this equation:

- **Global Illumination** algorithms account for light transport between several points on the scene before reaching the eye (several bounces).
  - Thus they can account for reflection, shadows, reflections etc.
  - More expensive, usually used in off-line rendering
  - E.g. animated movies (~2-4 hours per frame !)

- **Local Illumination** algorithms assume light only "bounces" once travelling from light source to a point in the scene and then to the eye.
  - Less realistic, usually used in real-time rendering
  - E.g. in games, we need to render at ~0.05seconds per frame and on top of that deal with, sound, animation, physics, interaction, A.I. etc.
**LOCAL VS. GLOBAL ILLUMINATION**

Local
Illumination depends on local object & light sources only.

Global
Illumination at a point can depend on any other point in the scene.

---

**RAY TRACING**

A global illumination algorithm for photo-realistic image synthesis.

First implementation of ray tracing in computer graphics = Appel (IBM 1968)

Current ray tracing methods are attributed to Turner Whitted (Bell Labs, 1980), also called:
- Whitted ray Tracing
- Recursive Ray-Tracing
- the Whitted Illumination Model

---

**RAY-TRACING**

\[ I(P) = I_{\text{LOCAL}}(P) + k_{\text{REF}}(P) + k_{\text{TR}}(P) \]

The Local term is for direct diffuse lighting and is calculated using Phong Illumination.

In the global ray-traced terms, diffusion of light due to surface imperfections is totally ignored.

---

**RAY TRACING FROM EYE**

Starting at the light position implies tracing many rays that never reach the eye. Thus the traditional ray-tracing method is to start at the eye and trace rays backwards to the source.

---

**EYE RAY AND OBJECT INTERSECTION**

![Image](image.png)

Cast a ray from camera position through each pixel into the scene. Calculate where this intersects with objects in the scene. Get the first intersection.
DIFFUSE TERM & SHADOW RAY

Extend a "Shadow Feeler" and see if it is occluded by an object in the scene.

If so, the object is in shadow from this light source (this point doesn’t contribute to the colour of the pixel).

Otherwise solve the phong model to calculate the contribution of this point to the colour of the pixel.

If object is diffuse stop here.

REFLECTION

If object is specular THEN spawn a ray in the direction of perfect specular reflection.

For this new ray, we will again check whether it intersects an object.

If so, is the object in shadow?

If not, what is the contribution of this object to the colour of the pixel?

If not diffuse, we keep spawning new rays (bouncing the light).

Each ray contributes to the colour of the pixel it originated from.

REFRACTION

Similarly, if the object is transmissive then spawn a transmitted (or refracted) ray and repeat...

THE ANGLE OF REFRACTION

When light passes from a material of one optical density to another it changes direction.

The amount by which the direction changes is determined by the optical densities of the two media (Snell’s Law).

Optical density (and thus the amount of bending) is related to a value we call the index of refraction (ior) of the material.

INDEX OF REFRACTION (IOR)

<table>
<thead>
<tr>
<th>Material</th>
<th>Index of Refraction</th>
<th>&lt;--lowest optical density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum</td>
<td>1.0000</td>
<td>&lt;--lowest optical density</td>
</tr>
<tr>
<td>Air</td>
<td>1.0003</td>
<td></td>
</tr>
<tr>
<td>Ice</td>
<td>1.31</td>
<td></td>
</tr>
<tr>
<td>Water</td>
<td>1.33</td>
<td></td>
</tr>
<tr>
<td>Ethyl Alcohol</td>
<td>1.36</td>
<td></td>
</tr>
<tr>
<td>Plexiglas</td>
<td>1.51</td>
<td></td>
</tr>
<tr>
<td>Crown Glass</td>
<td>1.52</td>
<td></td>
</tr>
<tr>
<td>Light Flint Glass</td>
<td>1.58</td>
<td></td>
</tr>
<tr>
<td>Dense Flint Glass</td>
<td>1.66</td>
<td></td>
</tr>
<tr>
<td>Zircon</td>
<td>1.923</td>
<td></td>
</tr>
<tr>
<td>Diamond</td>
<td>2.417</td>
<td></td>
</tr>
<tr>
<td>Ruby</td>
<td>2.907</td>
<td></td>
</tr>
<tr>
<td>Gallium phosphide</td>
<td>3.50                &lt;--highest optical density</td>
<td></td>
</tr>
</tbody>
</table>

These refraction index values should work in POVRay.
RAY TRACING SUMMARY

1. Cast a ray
2. Determine intersections
3. For closest intersection:
   a) Extend light/shadow feeler ray + calculate local term
   b) Spawn Transmitted Ray (step 1)
   c) Spawn Reflected Ray (step 1)

RAY TREE

The spawning of reflected and refracted rays can be represented in tree structure.

TERMINATING RECURSION

All of the spawned rays contribute to the pixel that the tree originated from.
However each new ray contributes less and less to the pixel.
Unlike in the real world, we can’t keep bouncing around for ever so we stop the recursion at some stage: recursion clipping
H Stop after a set number of bounces/refractions.
H Or stop when the contribution becomes less than a certain value.

RECURSION CLIPPING

max level = 1
max level = 1
max level = 2
max level = 2
max level = 4
max level = 4
max level = 3
max level = 3

Very high maximum recursion level

OBJECT INTERSECTION

Intersection testing is solved mathematically. But is an inherently expensive geometrical problem. (out of the scope of current lecture)

REFLECTION

Pov-Ray does the ray tracing for you. All you need to do is specify if your object is reflective. This is done in the finish modifier.
The reflection parameter with a value between 0 and 1 specifies how reflective an object is.

RAY TREE

Ray Tree
eye ray
specular
diffuse
eye ray
R
R
R
T
T
R
R

Recursion clipping
max level = 1
max level = 2
max level = 3
max level = 4

finish
{ reflection 0.5 //phong etc... }
**TRANSLUCENCY**

N.B. A filter value of 0 is fully opaque, 1 is fully transparent.

**REFRACTION**

**CAUSTICS**

Caustics are a phenomenon created by a lens effect of transmissive objects. Light is bent in such away that multiple rays are focused in particular areas creating brightly flares.

**ASIDE: RADIOSITY**

**RADIOSITY: “DIFFUSE ILLUMINATION”**

This diffuse scene (no highly specular reflectors) looks dark as any surface not directly hit by light is coloured in black.

**RADIOSITY**

The surfaces of the scene to be rendered are much divided up into one or more smaller surfaces (patches). Instead of casting rays we calculate how much light is transferred from patch to patch. This is very computationally expensive
The patch resolution affects the quality of the image and inversely also the rendering time.

**RADIOSITY EXAMPLE**

**RADIOSITY IN POV-RAY**

Parameters of radiosity are quite complex and somewhat outside the scope of this lecture. For details see:
- http://www.alexandre.eletrica.ufu.br/alexandre/cg/tutorial.pdf#page=175
- http://www.povray.org/documentation/view/3.6.0/270/
- http://wiki.povray.org/content/How_To:Use_radiosity

If you just want to try it out, add the following code to the top of your POV file:

```plaintext
#include "rad_def.inc"
global_settings{
    radiosity{
        Rad_Settings(Radiosity_Normal, off, off)
    }
}
```

Basic radiosity with default settings:
- Slightly more complex settings (see the file "rad_def.inc" in your Documents\POV-Ray\v3.7\Include folder for details and some other options).

N.B. this will take longer to render.

**LIGHT AND MATERIAL COLOR**

**RECAP + POV-RAY TIPS**
ENVIRONMENT MAPS

Shiny objects need something to reflect – often a sky sphere or skybox is used to enhance the realism of a scene with highly reflective (or refractive objects)

SKY_SPHERE

POVray has a built in object for a spherical sky environment map.

```
sky_sphere
{
  pigment
  {
    agate
turbulence .3
color_map
  {
    [0.0 rgb <0, 0, 1>]
    [1.0 rgb <.7, .6, 1>]
  }
}
```

```
sky_sphere
{
  pigment
  {
    bozo
turbulence 1
color_map
  {
    [0.0 rgb <.0, .0, .91>]
    [0.9 rgb <.81, .81, 1>]
    [1.0 rgb <1, 1, 1>]
  }
scale <.3, .21, .21>
}
```

```
#include "skies.inc"
sky_sphere
{
  pigment
  {
    P_Cloud1
  }
}
```

```
#declare Count=0;
#while (Count < 8)
object
  {
    MY_OBJECT
    scale 0.5
    translate <-8, 0, -4>
    translate <2 * Count, 0, 0>
  }
#declare Count=Count+1;
#end
```

```
#declare CountX=0;
#while (CountX < 8)
  #declare CountZ=0;
  #while (CountZ < 8)
    object
      {
        MY_OBJECT
        scale 0.5
        translate <-8, 0, -4>
        translate <2 * CountX, 0, 2*CountZ>
      }
  #declare CountZ=CountZ+1;
  #end
#declare CountX=CountX+1;
#end
```

SOME CODING: A WHILE LOOP

NESTED LOOP EXAMPLE
MATH FUNCTIONS

```plaintext
#declare Count=0;
#declare theta = 0;
#while (Count < 18)
    #declare theta = theta + radians(30);
    object
        {
            MY_OBJECT
                scale .3
                translate <sin(theta),0 , cos(theta)>
                scale pow(.94,Count)
        }
    #declare Count=Count+1;
#end
```

Other functions:
`abs( FLOAT )` | `acos( FLOAT )` | `acosh( FLOAT )` | `asin( FLOAT )` | `asinh( FLOAT )` | `atan( FLOAT)` | `atanh( FLOAT)` | `atan2( FLOAT , FLOAT )` | `ceil( FLOAT )` | `cos( FLOAT )` | `cosh( FLOAT )`
| `degrees( FLOAT )` | `exp( FLOAT )` | `floor( FLOAT )` | `int( FLOAT )` | `ln (Float)` | `log( FLOAT )` | `max( FLOAT , FLOAT, ... )` | `min( FLOAT , FLOAT, ... )` | `mod( FLOAT , FLOAT )` | `pow( FLOAT , FLOAT )` | `radians( FLOAT )` | `sin( FLOAT )` | `sinh( FLOAT )` | `sqrt( FLOAT )` | `tan( FLOAT )` | `tanh( FLOAT )` | `select( FLOAT , FLOAT , FLOAT [, FLOAT] )`

CAMERA

Minimum requirements: position and target

```plaintext
camera
    {
        location <0, 0, -3>
        look_at <0, 0, 0>
    }
```

Camera:
- `camera{ 
  [CAMERA_ITEMS...] }
- `CAMERA_ITEM`: `CAMERA_TYPE` | `CAMERA_VECTOR` | `CAMERA_MODIFIER` | `CAMERA_IDENTIFIER`
- `CAMERA_TYPE`: `perspective` | `orthographic` | `fisheye` | `ultra_wide_angle` | `omnimax` | `panoramic` | `cylinder CylinderType` | `spherical`
- `CAMERA_VECTOR`: `location <Location>` | `right <Right>` | `up <Up>` | `direction <Direction>` | `sky <Sky>`
- `CAMERA_MODIFIER`: `angle HORIZONTAL` [`VERTICAL`] | `look_at <Look_At>` | `blur_samples Num_of_Samples` | `aperture Size` | `focal_point <Point>` | `confidence Blur_Confidence` | `variance Blur_Variance` | `NORMAL` | `TRANSFORMATION`

For more details, see:

OtherSettings:

DEPTH OF FIELD

Depth of Field effects simulate blur in real lenses in cameras that are not out of the focal range of the camera. N.B. Rendering depth of field effects takes quite a while more for each frame.

```plaintext
{camera
    {
        location <3, 3, -5>
        look_at <0, 0, 0>
        aperture 0.3
        blur_samples 50
        focal_point <5,5,5>
    }
}
```

This point will be in sharp focus with increased blurring away from it.

Overall quality of blurring ~ bigger numbers are better but slower.
Between 0 and 1: affects how big the sharply focussed region is. Higher numbers mean more blurry regions.

For details:
- http://www.alexandre.eletrica.ufu.br/cg/tutorial.pdf#page=49

SOFT SHADOWS: POINT VS. AREA LIGHT SOURCE

```plaintext
shadow
```

```plaintext
soft shadow
```

```plaintext
full shadow
```

```plaintext
area light
```

```plaintext
light_source
    {
        <5, 5, -5>, rgb < 1, 1, 1>
        area_light
        <4, 4, 4>, <8, 8, 8>, 16, 16
    }
```

```plaintext
light_source
    {
        <5, 5, -5>, rgb < 1, 1, 1>
    }
```

AREA LIGHT SOURCE — SOFT SHADOWS
```
light_source
{
<10, 7, -5>  // the position of the light source
color <1, 1, 1>  // the color (here it is white)
area_light
<1, 0, 0; 0, 1, 0>,
4, 4
}
```

Note: This will increase rendering time.