

Course overview

Digital Image Synthesis
Yung-Yu Chuang

with slides by Mario Costa Sousa, Pat Hanrahan and Ravi Ramamoorthi

Logistics



- Meeting time: 2:20pm-5:20pm, Thursday
- Classroom: CSIE Room 111
- Instructor: Yung-Yu Chuang (cyy@csie.ntu.edu.tw)
- TA: 陳育聖
- Webpage:
<http://www.csie.ntu.edu.tw/~cyy/rendering>
id/password
- Mailing list: rendering@cmlab.csie.ntu.edu.tw
Please subscribe via
<https://cmlmail.csie.ntu.edu.tw/mailman/listinfo/rendering/>

Prerequisites



- C++ programming experience is required.
- Basic knowledge on algorithm and data structure is essential.
- Knowledge on linear algebra, probability, calculus and numerical methods is a plus.
- Though not required, it is recommended that you have background knowledge on computer graphics.

Requirements (subject to change)

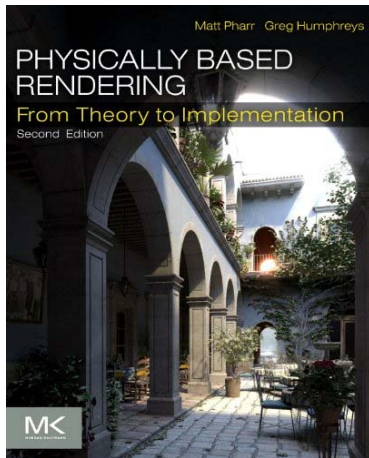


- 3 programming assignments (60%)
- Class participation (5%)
- Final project (35%)

Textbook



[Physically Based Rendering from Theory to Implementation](#),
2nd ed, by Matt Pharr and Greg Humphreys



- Authors have a lot of experience on ray tracing
- Complete (educational) code, more concrete
- Has been used in many courses and papers
- Implement some advanced or difficult-to-implement methods: subdivision surfaces, Metropolis sampling, BSSRDF, PRT.
- 3rd edition is coming next year!

pbrrt won Oscar 2014



- To Matt Pharr, Greg Humphreys and Pat Hanrahan for their formalization and reference implementation of the concepts behind physically based rendering, as shared in their book *Physically Based Rendering*.

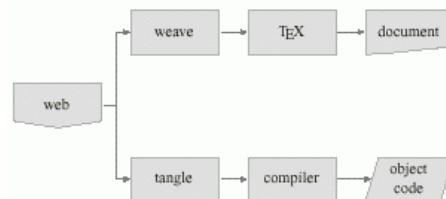
Physically based rendering has transformed computer graphics lighting by more accurately simulating materials and lights, allowing digital artists to focus on cinematography rather than the intricacies of rendering. First published in 2004, Physically Based Rendering is both a textbook and a complete source-code implementation that has provided a widely adopted practical roadmap for most physically based shading and lighting systems used in film production.



Literate programming



- A programming paradigm proposed by Knuth when he was developing Tex.
- Programs should be written more for people's consumption than for computers' consumption.
- The whole book is a long literate program. That is, when you read the book, you also read the complete program.



Processing a WEB

Features



- Mix prose with source: description of the code is as important as the code itself
- Allow presenting the code to the reader in a different order than to the compiler
- Easy to make index
- Traditional text comments are usually not enough, especially for graphics
- This decomposition lets us present code a few lines at a time, making it easier to understand.
- It looks more like pseudo code.

LP example



@\section{Selection Sort: An Example for LP}

We use {\it selection sort} to illustrate the concept of {\it iterate programming}.

Selection sort is one of the simplest sorting algorithms. It first find the smallest element in the array and exchange it with the element in the first position, then find the second smallest element and exchange it the element in the second position, and continue in this way until the entire array is sorted.

The following code implement the procedure for selection sort assuming an external array [[a]].

```
<<*>>=  
<<external variables>>  
void selection_sort(int n) {  
  <<init local variables>>  
  for (int i=0; i<n-1; i++) {  
    <<find minimum after the ith element>>  
    <<swap current and minimum>>  
  }  
}
```

LP example



```
<<find minimum after the ith element>>=  
min=i;  
for (int j=i+1; j<n; j++) {  
  if (a[j]<a[min]) min=j;  
}
```

```
<<init local variables>>=  
int min;
```

@ To swap two variables, we need a temporary variable [[t]] which is declared at the beginning of the procedure.

```
<<init local variables>>=  
int t;
```

@ Thus, we can use [[t]] to preserve the value of [[a[min]]] so that the swap operation works correctly.

```
<<swap current and minimum>>=  
t=a[min]; a[min]=a[i]; a[i]=t;
```

```
<<external variables>>=  
int *a;
```

LP example (tangle)



```
int *a;
```

```
void selection_sort(int n) {  
  int min;  
  
  int t;  
  
  for (int i=0; i<n-1; i++) {  
    min=i;  
    for (int j=i+1; j<n; j++) {  
      if (a[j]<a[min]) min=j;  
    }  
  
    t=a[min]; a[min]=a[i]; a[i]=t;  
  }  
}
```

LP example (weave)



1 Selection Sort: An Example for LP

We use *selection sort* to illustrate the concept of *iterate programming*. *Selection sort* is one of the simplest sorting algorithms. It first find the smallest element in the array and exchange it with the element in the first position, then find the second smallest element and exchange it the element in the second position, and continue in this way until the entire array is sorted. The following code implement the procedure for selection sort assuming an external array *a*.

```
1a < * 1a)≡  
  <external variables 1f)  
  void selection_sort(int n) {  
    <init local variables 1c)  
    for (int i=0; i<n-1; i++) {  
      <find minimum after the ith element 1b)  
      <swap current and minimum 1e)  
    }  
  }  
  
1b <find minimum after the ith element 1b)≡ (1a)  
  min=i;  
  for (int j=i+1; j<n; j++) {  
    if (a[j]<a[min]) min=j;  
  }
```

pbrt



- Pbrt is designed to be
 - Complete: includes features found in commercial high-quality renderers.
 - Illustrative: select and implement elegant methods.
 - Physically based
- Efficiency was given a lower priority (the unofficial fork [luxrender](http://www.luxrender.net) could be more efficient)
- [Source code browser](#)

LuxRender (<http://www.luxrender.net>)



LuxRender
GPL PHYSICALLY BASED RENDERER

Home About Download Documentation Community Development

Home

Development News
More...

New hair support coming

A much improved hair support is being added to LuxRender. The first results are quite promising.

...

SLG renderer

The SLG renderer branch has just been merged with mainline. This means from now on LuxRender has a full GPU accelerated render mode, it will be available in

Overview

LuxRender is a physically based and unbiased rendering engine. Based on state of the art algorithms, LuxRender simulates the flow of light according to physical equations, thus producing realistic images of photographic quality.

Get LuxRender

To get started with LuxRender, choose a package:

Blender 2.49h Blender 2.6x 3DS Max
SketchUp C4D DAZ Studio

Community News
More...

New exporter for Carrara

Luxus, a commercial exporter for Carrara to LuxRender, has just been officially released.

...

LuxRender advertized with Poser 10

A card advertizing Reality and LuxRender is present in every Poser 10 box thanks to RuntimeDNA.

...

Mitsuba (<http://www.mitsuba-renderer.org/>)



mitsuba-renderer.org about documentation download misc -bugs +blog

Mitsuba
PHYSICALLY BASED RENDERER

ABOUT

Mitsuba is a research-oriented rendering system in the style of **PBRt**, from which it derives much inspiration. It is written in portable C++, implements unbiased as well as biased techniques, and contains heavy optimizations targeted towards current CPU architectures. Mitsuba is extremely modular: it consists of a small set of core libraries and over 100 different plugins that implement functionality ranging from materials and light sources to complete rendering algorithms.

In comparison to other open source renderers, Mitsuba places a strong emphasis on experimental rendering techniques, such as path-based formulations of Metropolis Light Transport and volumetric modeling approaches. Thus, it may be of genuine interest to those who would like to experiment with such techniques that haven't yet

INTEGRATORS

A wide range of rendering techniques are available, including:

- Ambient occlusion
- Direct illumination
- Monte-Carlo path tracer which solves the full Radiative Transfer Equation
- Photon mapper with irradiance gradients
- Adjoint particle tracer
- Bidirectional path tracer
- Instant Radiosity (hardware-accelerated)
- Progressive Photon Mapper
- Stochastic Progressive Photon Mapper
- Path Space Metropolis Light Transport
- Primary Sample Space Metropolis Light Transport
- Energy redistribution path tracer

New features of pbrt2



- Remove plug-in architecture, but still an *extensible* architecture
- Add multi-thread support (automatic or --ncores)
- OpenEXR is recommended, not required
- HBV is added and becomes default
- Can be full spectral, do it at compile time
- Animation is supported
- Instant global illumination, extended photon map, extended infinite light source
- Improved irradiance cache

New features of pbrt2



- BSSRDF is added
- Metropolis light transport
- Precomputed radiance transfer
- Support measured BRDF

Reference books



References



- SIGGRAPH proceedings
- SIGGRAPH Asia proceedings
- Proceedings of Eurographics Symposium on Rendering
- Eurographics proceedings
- Most can be found at [this link](#).

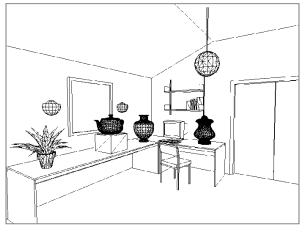
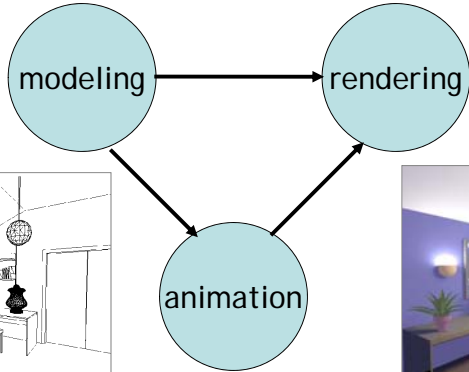
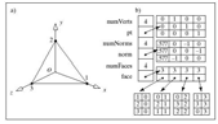
Image synthesis (Rendering)



- Create a 2D picture of a 3D world



Computer graphics





Physically-based rendering



uses physics to simulate the interaction between matter and light, realism is the primary goal



Realism



- Shadows
- Reflections (Mirrors)
- Transparency
- Interreflections
- Detail (Textures...)
- Complex Illumination
- Realistic Materials
- And many more

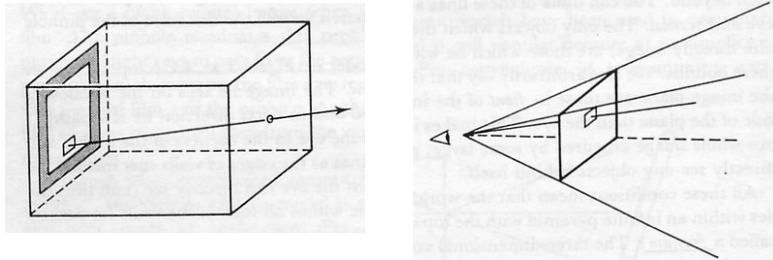


Other types of rendering

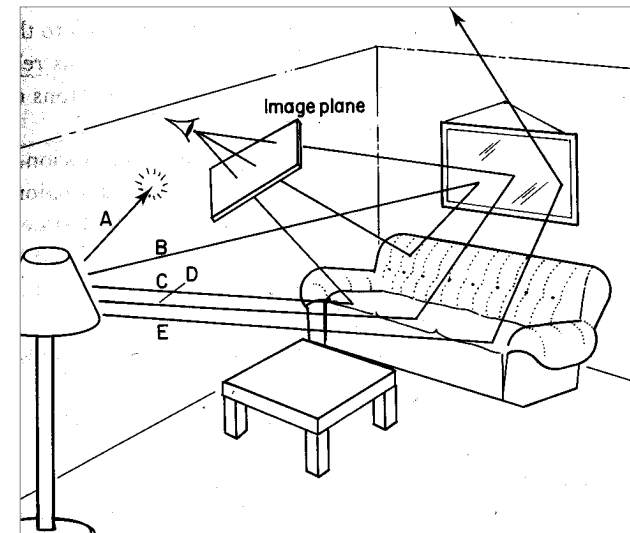


- Non-photorealistic rendering
- Image-based rendering
- Point-based rendering
- Volume rendering
- Perceptual-based rendering
- Artistic rendering

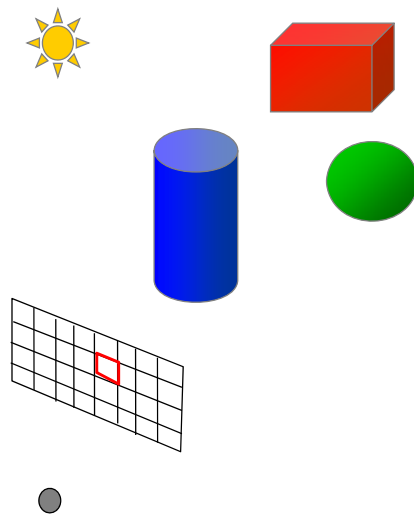
Pinhole camera



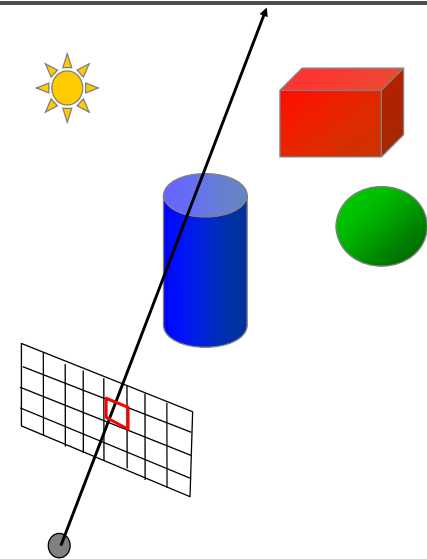
Introduction to ray tracing



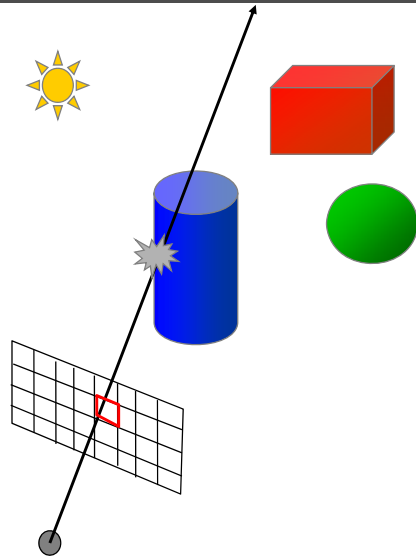
Ray Casting (Appel, 1968)



Ray Casting (Appel, 1968)



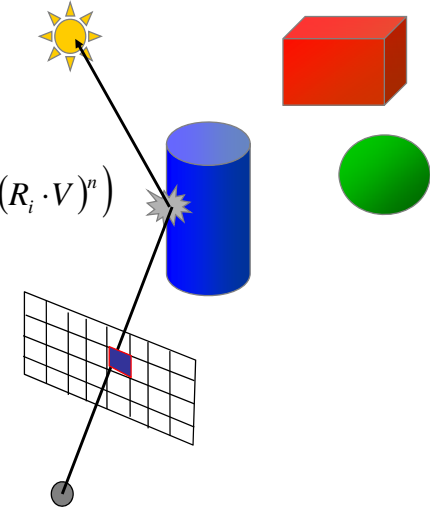
Ray Casting (Appel, 1968)



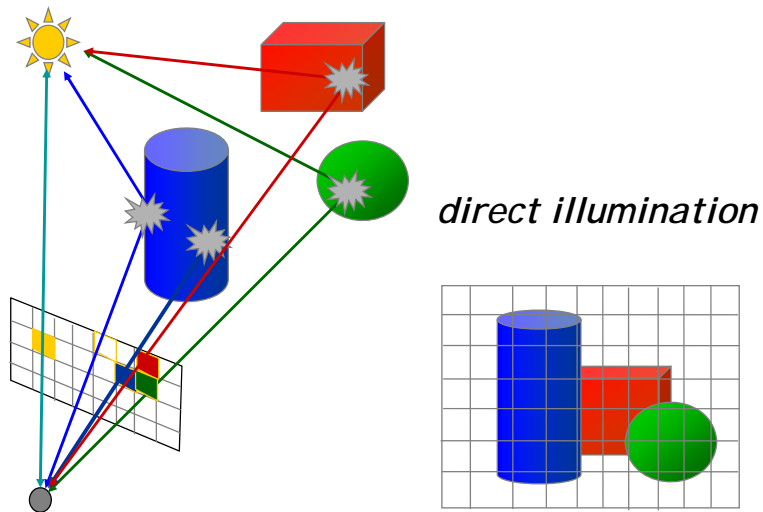
Ray Casting (Appel, 1968)



$$k_a I_a + \sum_{i=1}^{nls} I_i (k_d (L_i \cdot N) + k_s (R_i \cdot V)^n)$$



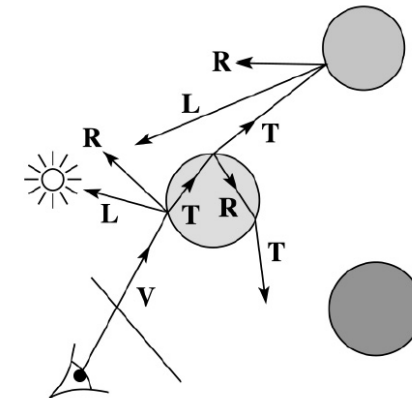
Ray Casting (Appel, 1968)



Whitted ray tracing algorithm



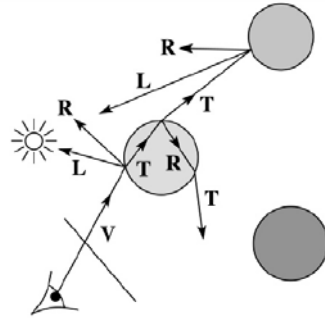
- ◆ Combines eye ray tracing + rays to light
- ◆ Recursively traces rays



Whitted ray tracing algorithm



- For each pixel, trace a **primary ray** in direction \mathbf{V} to the first visible surface.
- For each intersection, trace **secondary rays**:
 - ◆ **Shadow rays** in directions \mathbf{L}_i to light sources
 - ◆ **Reflected ray** in direction \mathbf{R} .
 - ◆ **Refracted ray** or **transmitted ray** in direction \mathbf{T} .



Shading



If $I(P_0, \mathbf{u})$ is the intensity seen from point P along direction \mathbf{u}

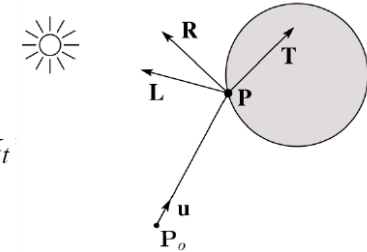
$$I(P_0, \mathbf{u}) = I_{direct} + I_{reflected} + I_{transmitted}$$

where

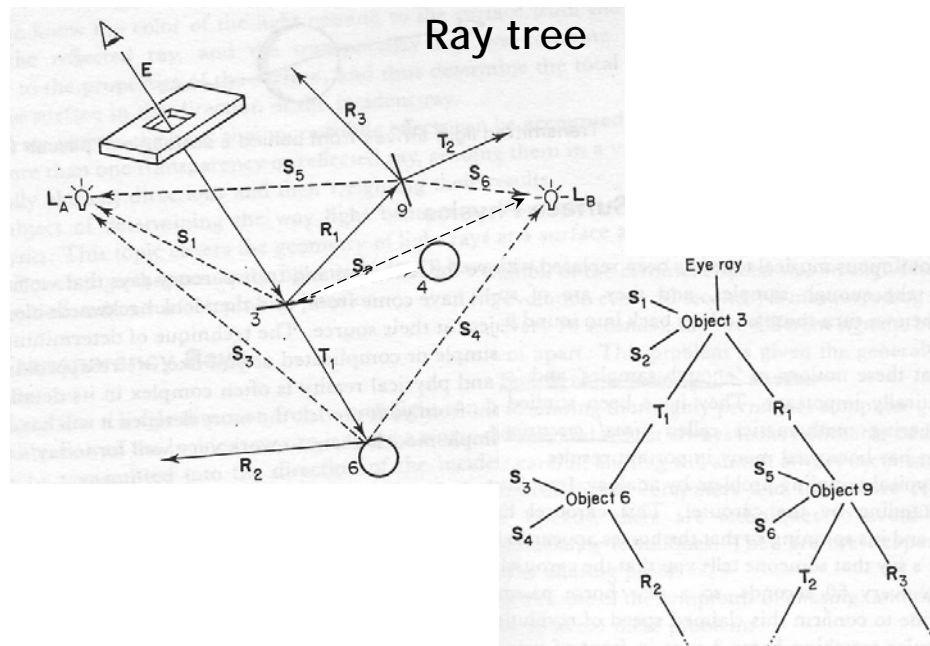
$$I_{direct} = \text{Shade}(\mathbf{N}, \mathbf{L}, \mathbf{u}, \mathbf{R}) \text{ (e.g. Phong shading model)}$$

$$I_{reflected} = k_r I(P, \mathbf{R})$$

$$I_{transmitted} = k_t I(P, \mathbf{T})$$



Typically, we set $k_r = k_s$ and k_t



Recursive ray tracing (Whitted, 1980)



Components of a ray tracer



- Cameras
- Films
- Lights
- Ray-object intersection
- Visibility
- Surface scattering
- Recursive ray tracing

Minimal ray tracer



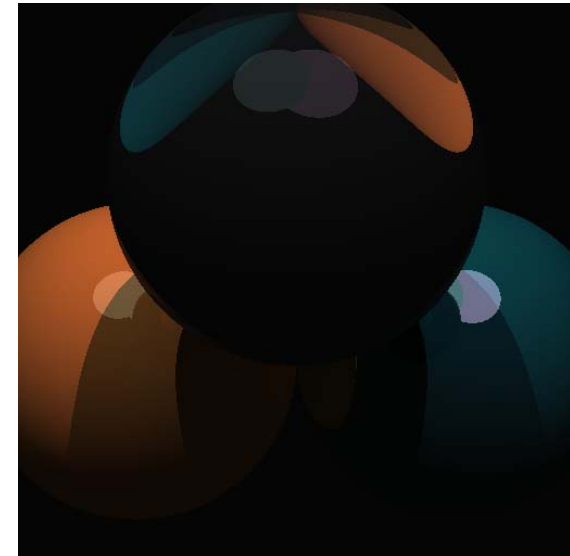
- Minimal ray tracer contest on *comp.graphics*, 1987
- Write the shortest Whitted-style ray tracer in C with the minimum number of tokens. The scene is consisted of spheres. (specular reflection and refraction, shadows)
- Winner: 916 tokens
- Cheater: 66 tokens (hide source in a string)
- Almost all entries have six modules: main, trace, intersect-sphere, vector-normalize, vector-add, dot-product.

Minimal ray tracer (Heckbert 1994)



```
typedef struct{double x,y,z;}vec;vec U,black,amb={.02,.02,.02};struct sphere{ vec cen,color;
double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,.05,.2,.85,0.,1.7,-1.,8.,-5,1.,.5,.2,1.,
.7,.3,0.,.05,1.2,1.,8.,-.5,1.,8.,8,1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,8,1.,7.,0.,0.,.6,1.5,-3.,-3.,12.,
.8,1.,1.,5.,0.,0.,.5,1.5.};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A ,B;{return A.x
*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*A.x;B.y+=a*A.y;B.z+=a*A.z;
return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(vdot(A,A)),A,black);}struct sphere*intersect
(P,D)vec P,D;{best=0;tmin=1e30;s= sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),
u=b*b-vdot(U,U)+s->rad*s ->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&
u<tmin?best=s,u: tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level-->return black;if(s=intersect(P,D));else return amb;color=amb;eta=
s->ir;d=-vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen)));if(d<0)N=vcomb(-1.,N,black),
eta=1/eta,d=-d;l=sph+5;while(l-->sph)if((e=l->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&
intersect(P,U)=l)color=vcomb(e ,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z
*=U.z;e=1-eta* eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-
sqrt(e),N,black)))&black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,color,vcomb
(s->kl,U,black)))));}main(){printf("%d %d\n",32,32);while(yx<32*32) U.x=yx%32-32/2,U.z=32/2-
yx+/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255., trace(3,black,vunit(U)),black).printf
("%%.0f %%.0f %%.0f\n",U);}/*minray!*/
```

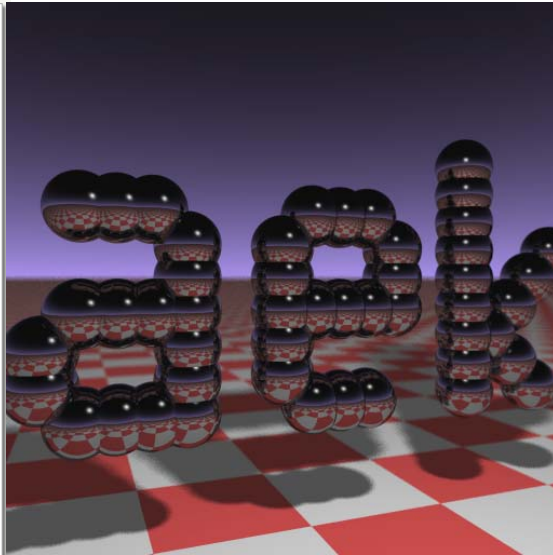
What it can do



Another business card raytracer



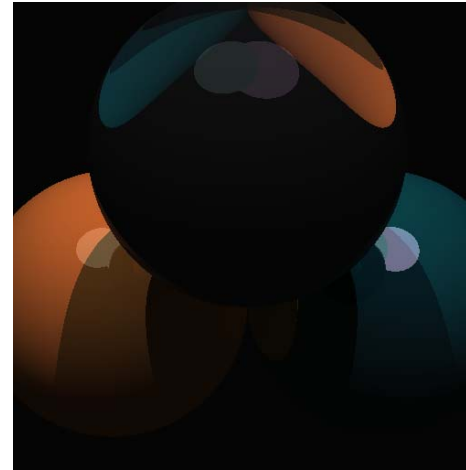
```
#include <stdlib.h> // card > aek.ppm
#include <stdio.h>
#include <math.h>
typedef int i;typedef float f;struct v{
f x,y,z;v operator+(v r){return v(x+r.x
,y+r.y,z+r.z);}v operator*(f r){return
v(x*r,y*r,z*r);}f operator%(v r){return
x*r.x+y*r.y+z*r.z;}v operator^(v r
){return v(y*r.z-z*r.y,z*r.x-x*r.z,x*r.r.
y-y*r.x);}v(f a,f b,f c){x=a;y=b;z=c;}v
operator!(){return*this*(1/sqrt(*this*
this));};i G[]={247570,200596,200600,
249748,18578,18577,231184,16,16};f R(){
return(f)rand()/RAND_MAX;};i T(v o,v d,f
&t,v&n){t=1e9;i m=0;f p=-o.z/d;f i(.01
<p)t=p,n=v(0,0,1),m=1;for(i k=19;k--;)
for(i j=9;j--;)if(G[j]&1<<k){v p=-(k
,0,-j-4);f b=p&n,c=p&n-1,q=b*b-c;if(q>0
){f s=-b-sqrt(q);if(s<t&&s>.01)t=s,n=!
(p+d*t),m=2;}}return m;};v S(v o,v d){f t
;v n; i m=T(o,d,t,n);if(!m)return v(.7
,.6,1)*pow(1-d,z,4);v h=o+d*t,l=(v(9+R(
),9+R(,16)+h*-1),r=d+n*(n&d*-2);f b=1&
n;if(b<0||T(h,l,t,n))b=0;f p=pow(1&n*(b
>0),99);if(m&1)(h=* 2;return(S)(ceil(
h.x)*ceil(h.y))&1?v(3,1,1):v(3,3,3))*(b
*.2+1);return v(o,p,p)+S(h,r)*.5;};
main(){printf("P6 512 512 255 ");v g=lv
(-6,-16,0),a=(v(0,0,1)*g)*.002,b=(g*a
)*.002,c=(a+b)*-.256+g;for(i y=512;y--;)
for(i x=512;x--;){v p(13,13,13);for(i r
=64;r--;){v t=a*(R()-.5)*99+b*(R()-.5)*
99;p=S(v(17,16,8)+t,(t*-1+(a*(R()-x)+b
*(y+R())+c)*16))*3.5+p;printf("%c%c%c"
,(i)p.x,(i)p.y,(i)p.z);}}
```



That's it?



- In this course, we will study how state-of-art ray tracers work.

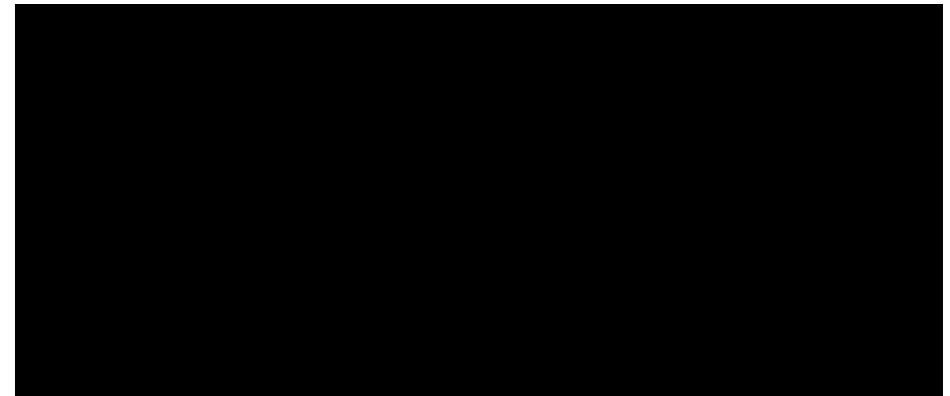


Issues



- Better Lighting + Forward Tracing
- Texture Mapping
- Sampling
- Modeling
- Materials
- Motion Blur, Depth of Field, Blurry Reflection/Refraction
 - *Distributed Ray-Tracing*
- Improving Image Quality
- Acceleration Techniques (better structure, faster convergence)

Disney's Practical Guide to Path Tracing



Really Practical Guide to Path Tracing



$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega_2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i$$

```
0000 **  
0001 #! /usr/bin/perl -I. -I./src -I./lib -I./include  
0002  
0003 #! Perl script for path tracing.  
0004  
0005 #! This file is part of the  
0006 #! Really Practical Guide to Path Tracing.  
0007 #! Copyright (c) 2014-2015, Matt Pharr and Greg Humphrey.  
0008 #! See http://www.pbs.org/~pharr/ for more details.  
0009 #! All rights reserved. This program and the accompanying materials  
0010 #! are distributed under the terms of the Creative Commons Attribution  
0011 #! License, version 4.0. See http://creativecommons.org/licenses/by/4.0/  
0012 #! for more details. Note that the text contents of the code  
0013 #! are not necessarily accurate. Some have been updated to use  
0014 #! the latest C++11 compiler and standard library. See the file  
0015 #! README for more details.  
0016  
0017 #! The original author's name and the name of the original project  
0018 #! are included in the program, if any, see http://www.pbs.org/~pharr/  
0019 #!  
0020  
0021 #! #include <string>  
0022 #! #include <vector>  
0023 #! #include <array>  
0024 #! #include <string_view>  
0025 #! #include <limits>  
0026 #! #include <cmath>  
0027 #! #include <random>  
0028 #! #include <random_numbers>  
0029 #! #include <math_constants>  
0030 #! #include <math_constants.h>  
0031 #! #include <math_constants.h>  
0032 #! #include <math_constants.h>  
0033 #! #include <math_constants.h>  
0034 #! #include <math_constants.h>  
0035 #! #include <math_constants.h>  
0036 #! #include <math_constants.h>  
0037 #! #include <math_constants.h>  
0038 #! #include <math_constants.h>  
0039 #! #include <math_constants.h>  
0040 #! #include <math_constants.h>  
0041 #! #include <math_constants.h>  
0042 #! #include <math_constants.h>  
0043 #! #include <math_constants.h>  
0044 #! #include <math_constants.h>  
0045 #! #include <math_constants.h>  
0046 #! #include <math_constants.h>  
0047 #! #include <math_constants.h>  
0048 #! #include <math_constants.h>  
0049 #! #include <math_constants.h>  
0050 #! #include <math_constants.h>  
0051 #! #include <math_constants.h>  
0052 #! #include <math_constants.h>  
0053 #! #include <math_constants.h>  
0054 #! #include <math_constants.h>  
0055 #! #include <math_constants.h>  
0056 #! #include <math_constants.h>  
0057 #! #include <math_constants.h>  
0058 #! #include <math_constants.h>  
0059 #! #include <math_constants.h>  
0060 #! #include <math_constants.h>  
0061 #! #include <math_constants.h>  
0062 #! #include <math_constants.h>  
0063 #! #include <math_constants.h>  
0064 #! #include <math_constants.h>  
0065 #! #include <math_constants.h>  
0066 #! #include <math_constants.h>  
0067 #! #include <math_constants.h>  
0068 #! #include <math_constants.h>  
0069 #! #include <math_constants.h>  
0070 #! #include <math_constants.h>  
0071 #! #include <math_constants.h>  
0072 #! #include <math_constants.h>  
0073 #! #include <math_constants.h>  
0074 #! #include <math_constants.h>  
0075 #! #include <math_constants.h>  
0076 #! #include <math_constants.h>  
0077 #! #include <math_constants.h>  
0078 #! #include <math_constants.h>  
0079 #! #include <math_constants.h>  
0080 #! #include <math_constants.h>  
0081 #! #include <math_constants.h>  
0082 #! #include <math_constants.h>  
0083 #! #include <math_constants.h>  
0084 #! #include <math_constants.h>  
0085 #! #include <math_constants.h>  
0086 #! #include <math_constants.h>  
0087 #! #include <math_constants.h>  
0088 #! #include <math_constants.h>  
0089 #! #include <math_constants.h>  
0090 #! #include <math_constants.h>  
0091 #! #include <math_constants.h>  
0092 #! #include <math_constants.h>  
0093 #! #include <math_constants.h>  
0094 #! #include <math_constants.h>  
0095 #! #include <math_constants.h>  
0096 #! #include <math_constants.h>  
0097 #! #include <math_constants.h>  
0098 #! #include <math_constants.h>  
0099 #! #include <math_constants.h>  
0100 #! #include <math_constants.h>  
0101 #! #include <math_constants.h>  
0102 #! #include <math_constants.h>  
0103 #! #include <math_constants.h>  
0104 #! #include <math_constants.h>  
0105 #! #include <math_constants.h>  
0106 #! #include <math_constants.h>  
0107 #! #include <math_constants.h>  
0108 #! #include <math_constants.h>  
0109 #! #include <math_constants.h>  
0110 #! #include <math_constants.h>  
0111 #! #include <math_constants.h>  
0112 #! #include <math_constants.h>  
0113 #! #include <math_constants.h>  
0114 #! #include <math_constants.h>  
0115 #! #include <math_constants.h>  
0116 #! #include <math_constants.h>  
0117 #! #include <math_constants.h>  
0118 #! #include <math_constants.h>  
0119 #! #include <math_constants.h>  
0120 #! #include <math_constants.h>  
0121 #! #include <math_constants.h>  
0122 #! #include <math_constants.h>  
0123 #! #include <math_constants.h>  
0124 #! #include <math_constants.h>  
0125 #! #include <math_constants.h>  
0126 #! #include <math_constants.h>  
0127 #! #include <math_constants.h>  
0128 #! #include <math_constants.h>  
0129 #! #include <math_constants.h>  
0130 #! #include <math_constants.h>  
0131 #! #include <math_constants.h>  
0132 #! #include <math_constants.h>  
0133 #! #include <math_constants.h>  
0134 #! #include <math_constants.h>  
0135 #! #include <math_constants.h>  
0136 #! #include <math_constants.h>  
0137 #! #include <math_constants.h>  
0138 #! #include <math_constants.h>  
0139 #! #include <math_constants.h>  
0140 #! #include <math_constants.h>  
0141 #! #include <math_constants.h>  
0142 #! #include <math_constants.h>  
0143 #! #include <math_constants.h>  
0144 #! #include <math_constants.h>  
0145 #! #include <math_constants.h>  
0146 #! #include <math_constants.h>  
0147 #! #include <math_constants.h>  
0148 #! #include <math_constants.h>  
0149 #! #include <math_constants.h>  
0150 #! #include <math_constants.h>  
0151 #! #include <math_constants.h>  
0152 #! #include <math_constants.h>  
0153 #! #include <math_constants.h>  
0154 #! #include <math_constants.h>  
0155 #! #include <math_constants.h>  
0156 #! #include <math_constants.h>  
0157 #! #include <math_constants.h>  
0158 #! #include <math_constants.h>  
0159 #! #include <math_constants.h>  
0160 #! #include <math_constants.h>  
0161 #! #include <math_constants.h>  
0162 #! #include <math_constants.h>  
0163 #! #include <math_constants.h>  
0164 #! #include <math_constants.h>  
0165 #! #include <math_constants.h>  
0166 #! #include <math_constants.h>  
0167 #! #include <math_constants.h>  
0168 #! #include <math_constants.h>  
0169 #! #include <math_constants.h>  
0170 #! #include <math_constants.h>  
0171 #! #include <math_constants.h>  
0172 #! #include <math_constants.h>  
0173 #! #include <math_constants.h>  
0174 #! #include <math_constants.h>  
0175 #! #include <math_constants.h>  
0176 #! #include <math_constants.h>  
0177 #! #include <math_constants.h>  
0178 #! #include <math_constants.h>  
0179 #! #include <math_constants.h>  
0180 #! #include <math_constants.h>  
0181 #! #include <math_constants.h>  
0182 #! #include <math_constants.h>  
0183 #! #include <math_constants.h>  
0184 #! #include <math_constants.h>  
0185 #! #include <math_constants.h>  
0186 #! #include <math_constants.h>  
0187 #! #include <math_constants.h>  
0188 #! #include <math_constants.h>  
0189 #! #include <math_constants.h>  
0190 #! #include <math_constants.h>  
0191 #! #include <math_constants.h>  
0192 #! #include <math_constants.h>  
0193 #! #include <math_constants.h>  
0194 #! #include <math_constants.h>  
0195 #! #include <math_constants.h>  
0196 #! #include <math_constants.h>  
0197 #! #include <math_constants.h>  
0198 #! #include <math_constants.h>  
0199 #! #include <math_constants.h>  
0200 #! #include <math_constants.h>
```

Complex lighting



Complex lighting



Refraction/dispersion



Caustics



Realistic materials



Translucent objects



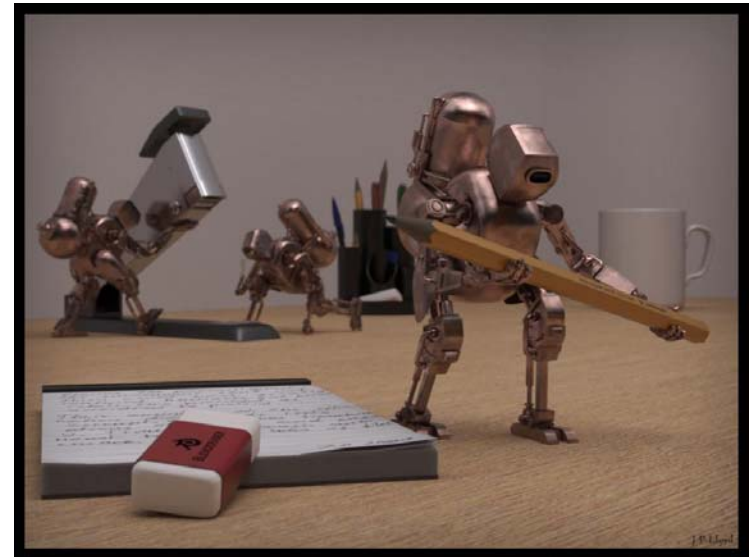
Texture and complex materials



Even more complex materials



Complex material (luxrender)



Depth of field (luxrender)



Motion blur (luxrender)



Refraction (Luxrender)



Applications



- Movies
- Interactive entertainment
- Industrial design
- Architecture
- Culture heritage



IKEA



- Today (2014), around 60-75% of all IKEA's product (single product) images are CG. About 35% of all IKEA Communication's non-product images are fully CG.



IKEA



- They use 3DStudio Max and V-Ray.



Animation production pipeline



story



text treatment



storyboard



voice

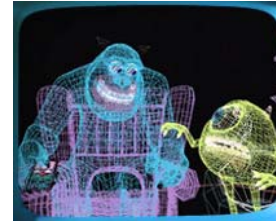


storyreal



look and feel

Animation production pipeline



modeling/articulation



layout



animation



shading/lighting



rendering



final touch

Monster University Progression



Pixar in a Box



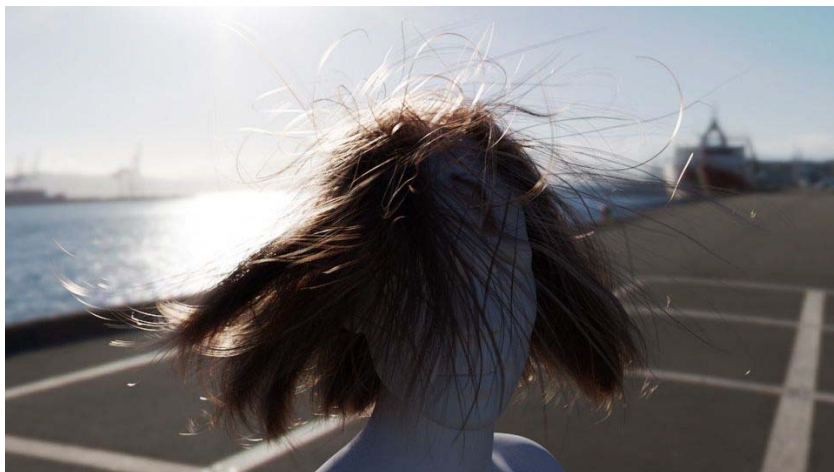
Ray tracing finally catches up



Arnold renderer



Manuka (Weta digital)

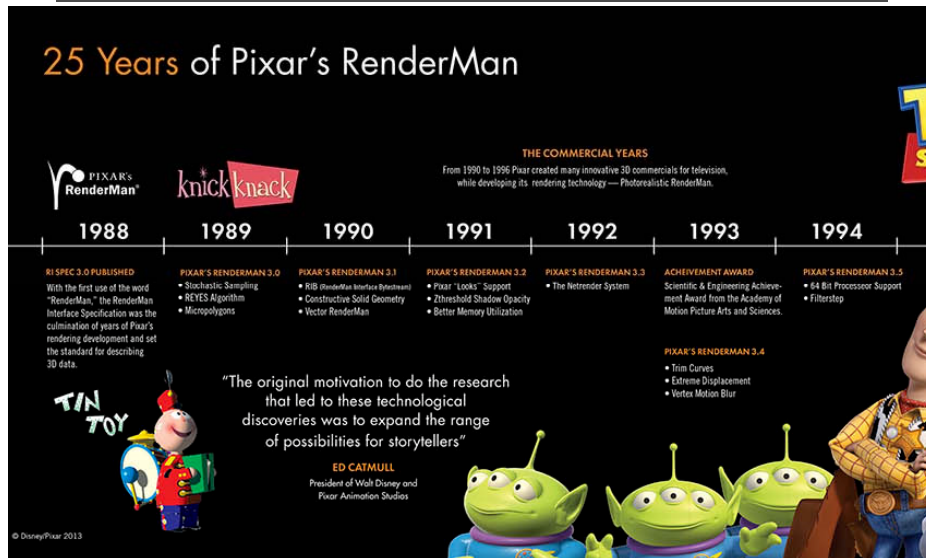


<http://www.fxguide.com/featured/manuka-weta-digital-s-new-renderer/>

Hyperion (Disney)



Pixar RenderMan Timeline



Monster University



The blue umbrella



Homework #0



- Download and install pbprt2.
- Run several examples
- Set it up in a debugger environment so that you can trace the code
- Optionally, create your own scene