Course overview

Digital Image Synthesis

Yung-Yu Chuang

9/17/2008

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

Logistics

- **Meeting time:** 2:20pm-5:20pm, Wednesday
- **Classroom:** CSIE Room 111
- **Instructor:** Yung-Yu Chuang (cyy@csie.ntu.edu.tw)
- **TA:** Ming-Fang Weng
- **Webpage**:
  http://www.csie.ntu.edu.tw/~cyy/rendering
  id/password
- **Forum:**
  http://www.cmlab.csie.ntu.edu.tw/~cyy/forum/viewforum.php?f=14
- **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 and numerical methods is a plus.
- Knowledge on compiler (bison/flex) might be useful.
- Though not required, it is recommended that you have background knowledge on computer graphics.

Requirements (subject to change)

- 3 programming assignments (55%)
- Class participation (5%)
- Final project (40%)

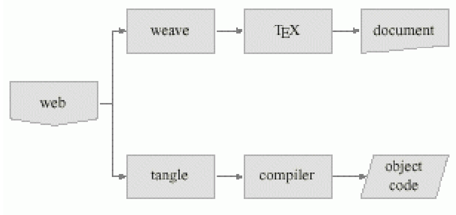
Textbook

[*Physically Based Rendering from Theory to Implementation*](), by Matt Pharr and Greg Humphreys

- Authors have a lot of experience on ray tracing
- Complete (educational) code, more concrete
- Plug-in architecture, easy for experiments and extensions
- Has been used in some courses and papers
- Downside: educational but not for high performance (unofficial fork: luxrenderer)

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 a 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 literate 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 it literate 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.

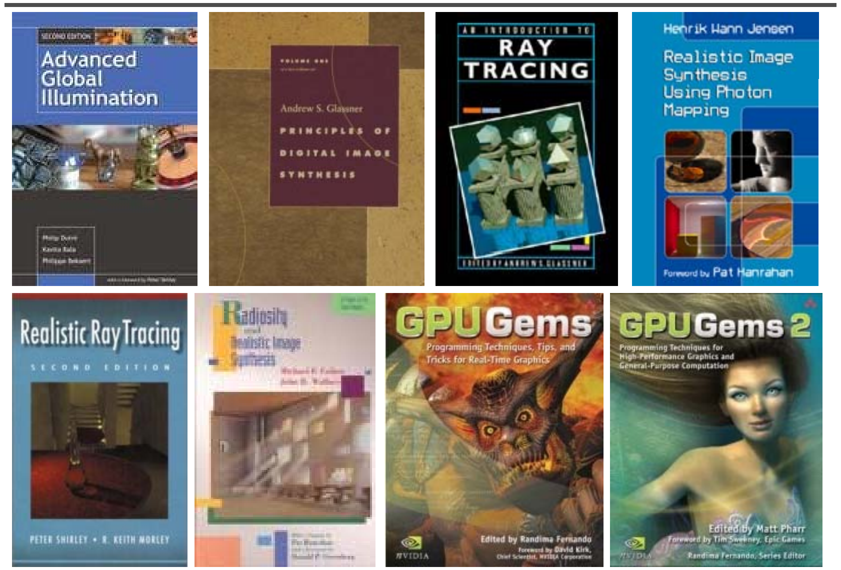
1a ⟨* 1a⟩≡

```
⟨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;
}
```

Reference books



References

- SIGGRAPH proceedings
- Proceedings of Eurographics Symposium on Rendering
- Eurographics proceedings

Image synthesis (Rendering)

- Create a 2D picture of a 3D world

Applications

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

Animation production pipeline

story

text treatment

storyboard

voice

storyreal

look and feel

Animation production pipeline


modeling/articulation


layout

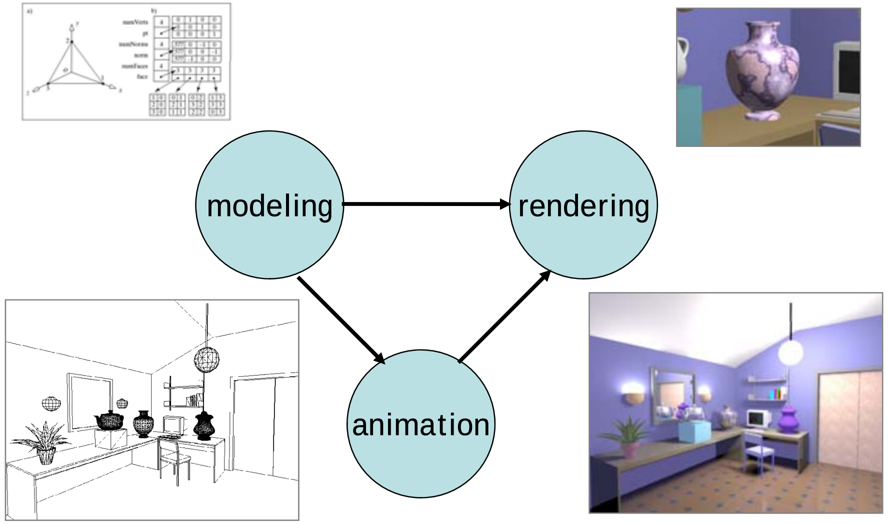

animation


shading/lighting


rendering


final touch

Computer graphics



The goal of this course

- Realistic rendering
- First 2/3: physically based rendering
- Remaining 1/3: real-time high-quality rendering

Physically-based rendering

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

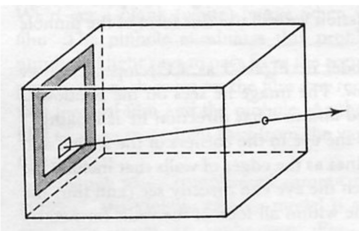
Realism

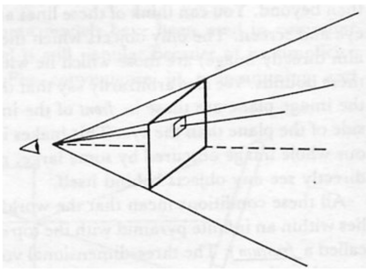
- Shadows
- Reflections (Mirrors)
- Transparency
- Interreflections
- Detail (Textures…)
- Complex Illumination
- Realistic Materials
- And many more

Other types of rendering

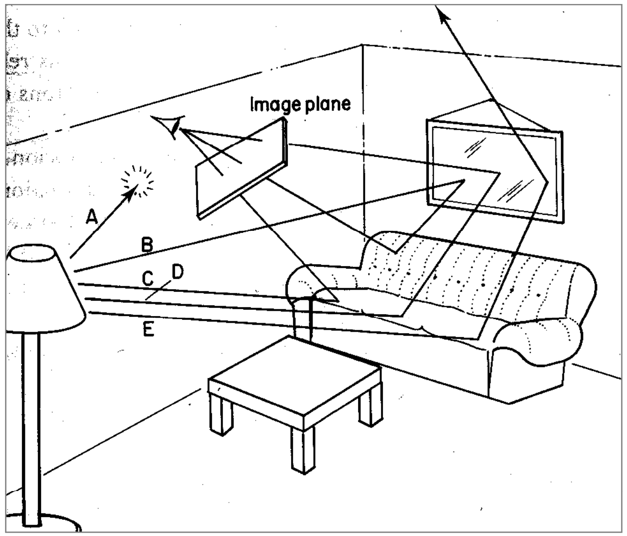
- Non-photorealistic rendering
- Image-based rendering
- Point-based rendering
- Volume rendering
- Perceptual-based rendering
- Artistic rendering

Pinhole camera

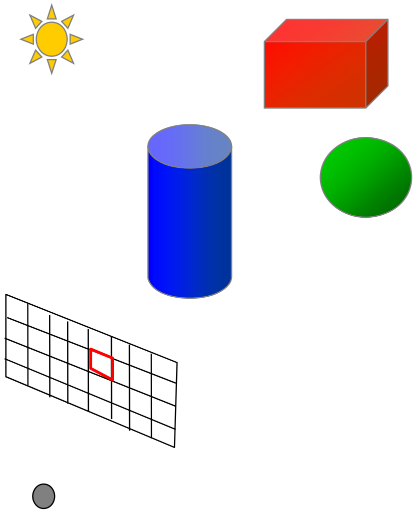




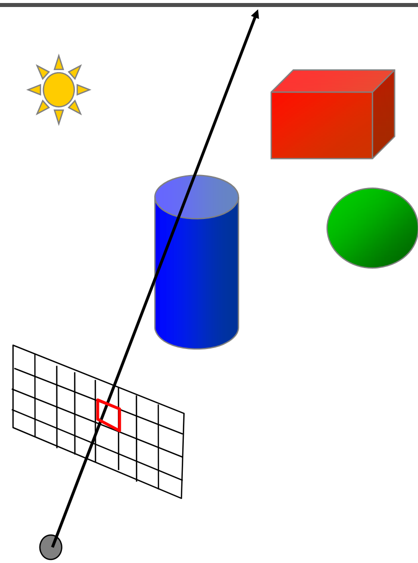
Introduction to ray tracing



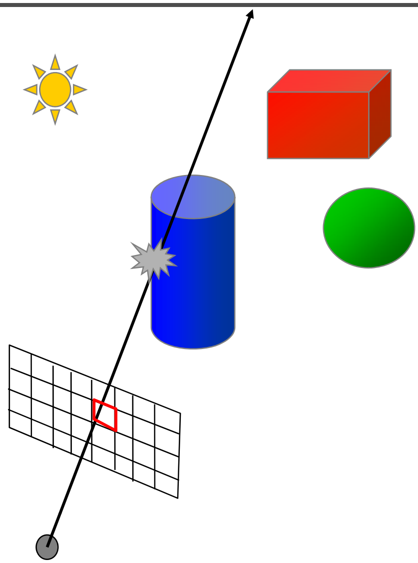
Ray Casting (Appel, 1968)



Ray Casting (Appel, 1968)

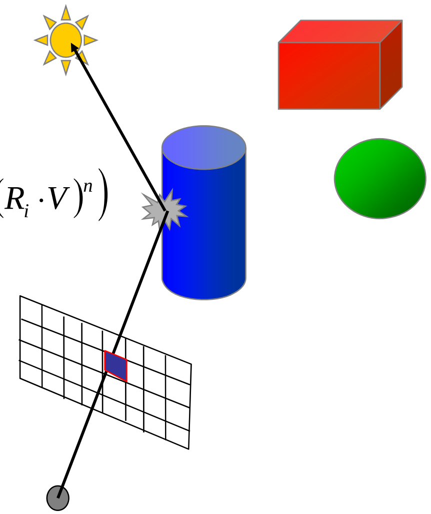


Ray Casting (Appel, 1968)

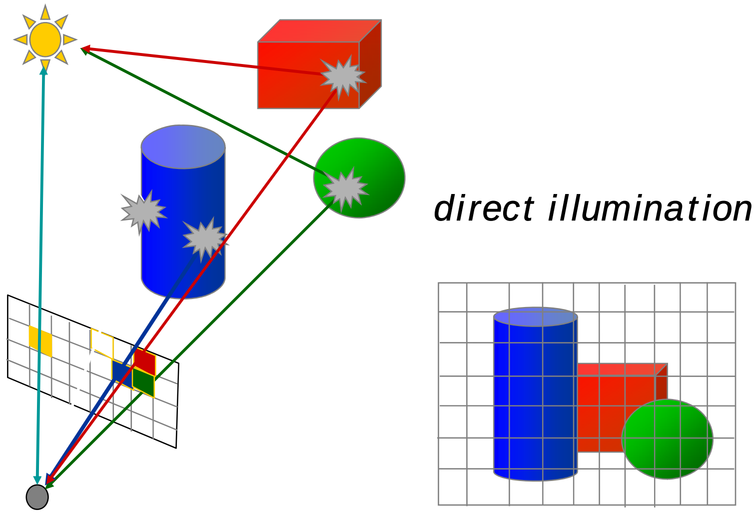


Ray Casting (Appel, 1968)

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

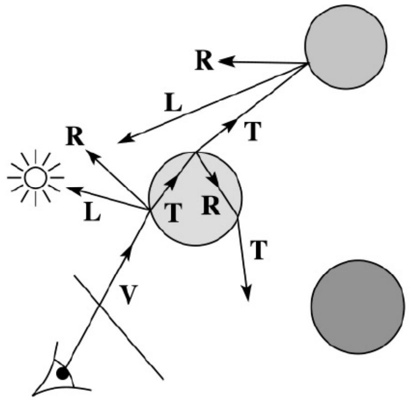


Ray Casting (Appel, 1968)



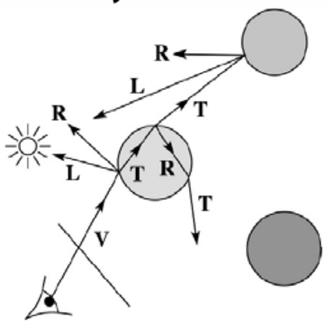
Whitted ray tracing algorithm

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



Whitted ray tracing algorithm

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



Shading

If I(P_0, **u**) is the intensity seen from point P along direction **u**

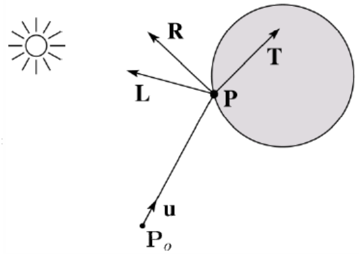
$$I(P_0, \mathbf{u}) = I_{direct} + I_{reflected} + I_{transmitted}$$

where

I_{direct} = Shade(**N**, **L**, **u**, **R**) (e.g. Phong shading model)

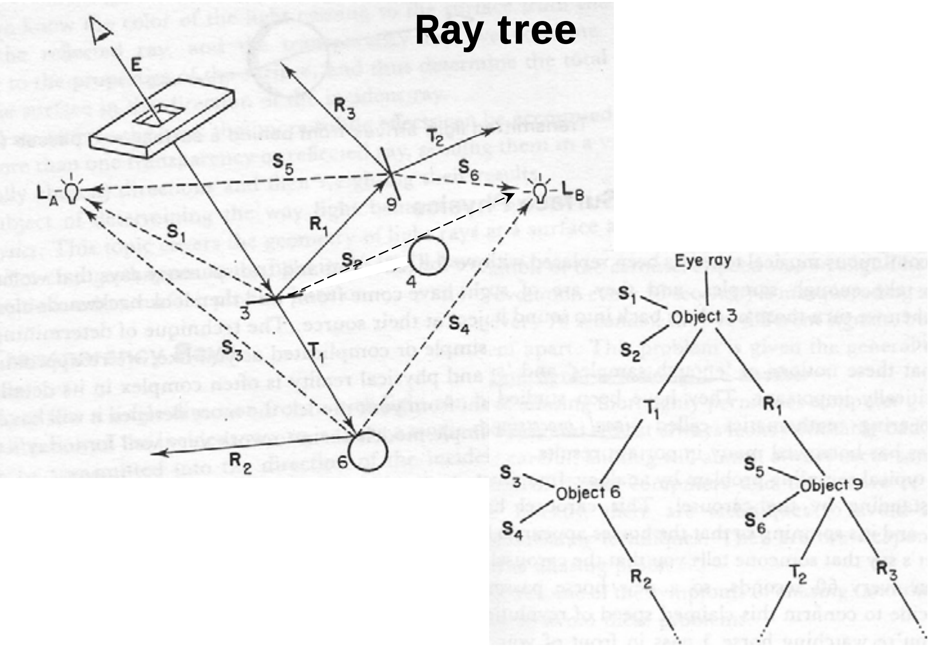
$$I_{reflected} = k_r I(P, \mathbf{R})$$

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

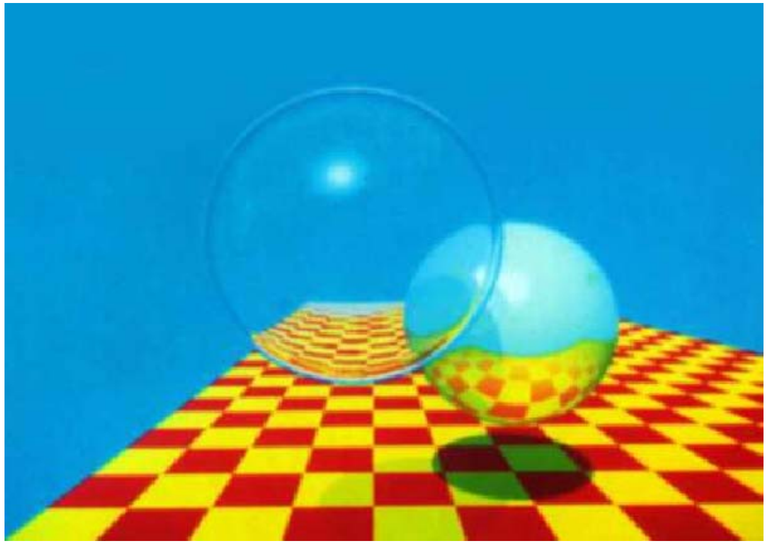


Typically, we set $k_r = k_s$ and k_t

Ray tree



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.,0.,.6,1.5,-3.,-3.,12.,
.8,1., 1.,5.,0.,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!*/
```

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)

Complex lighting

Complex lighting

Refraction/dispersion

Caustics

Realistic materials

Translucent objects

Texture and complex materials

Even more complex materials

Homework #0

- Download and install pbrt 1.03 (Linux version is recommended.)
- Run several examples
- Set it up in a debugger environment so that you can trace the code
- Optionally, create your own scene
- Pbrt1.03 source code tracing

Example scene

Location look at up vector

```
LookAt 0 10 100   0 -1 0 0 1 0
Camera "perspective" "float fov" [30]
PixelFilter "mitchell"
            "float xwidth" [2] "float ywidth" [2]
Sampler "bestcandidate"
Film "image" "string filename" ["test.exr"]
     "integer xresolution" [200]
     "integer yresolution" [200]
```

rendering options

```
# this is a meaningless comment
WorldBegin

AttributeBegin
  CoordSysTransform "camera"
  LightSource "distant"
              "point from" [0 0 0] "point to" [0 0 1]
              "color L"    [3 3 3]
AttributeEnd
```

id "type" param-list

"type name" [value]

Example scene

```
AttributeBegin
  Rotate 135 1 0 0

  Texture "checks" "color" "checkerboard"
          "float uscale" [8] "float vscale" [8]
          "color tex1" [1 0 0] "color tex2" [0 0 1]

  Material "matte"
           "texture Kd" "checks"
  Shape "sphere" "float radius" [20]
AttributeEnd
WorldEnd
```