Course overview

Digital Image Synthesis

Yung-Yu Chuang

with slides by Mario Costa Sousa, Pat Hanrahan and Revi 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

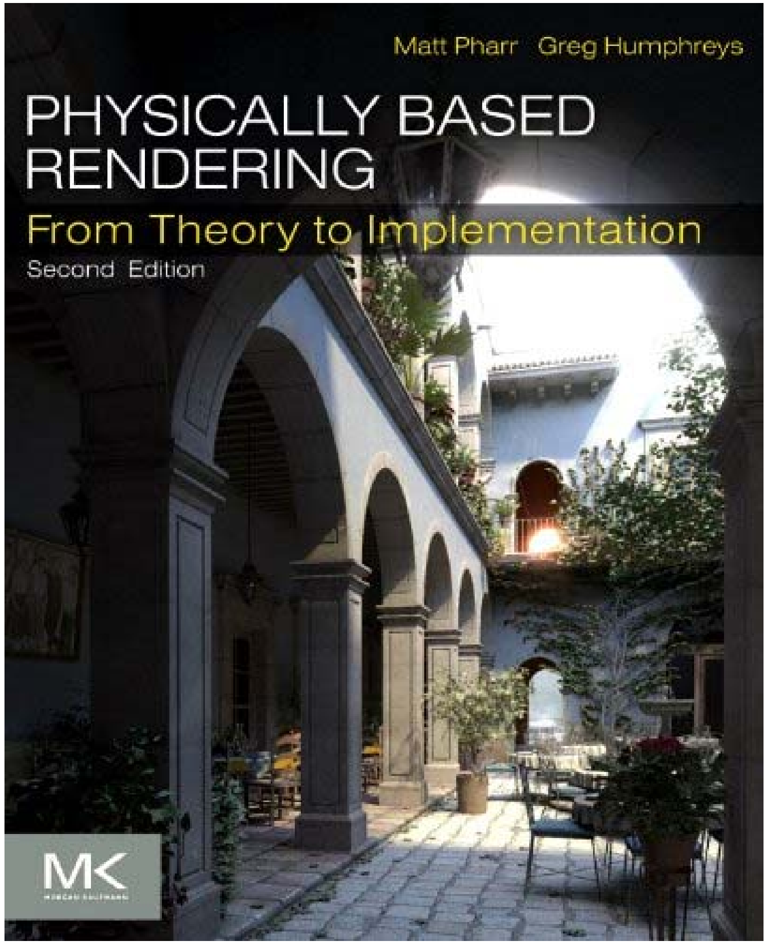
- **C++ programming experience is required**.
- Basic knowledge on algorithm and data structure is essential.
- Knowledge on linear algebra, probability, calculus and numerical methods is a plus.
- Though not required, it is recommended that you have background knowledge on computer graphics.

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!

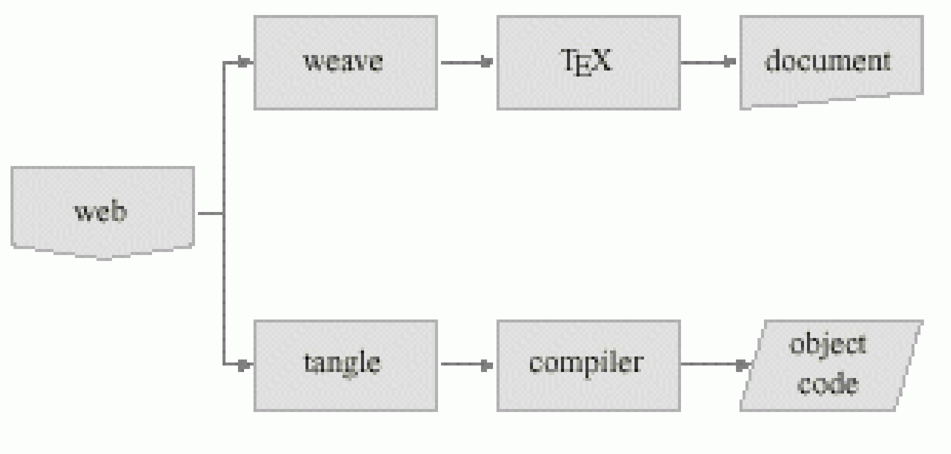
pbrt won Oscar 2014

- 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 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 continute in this way until the entire array is sorted. The following code implement the procedure for seletion 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⟩
        }
}
```

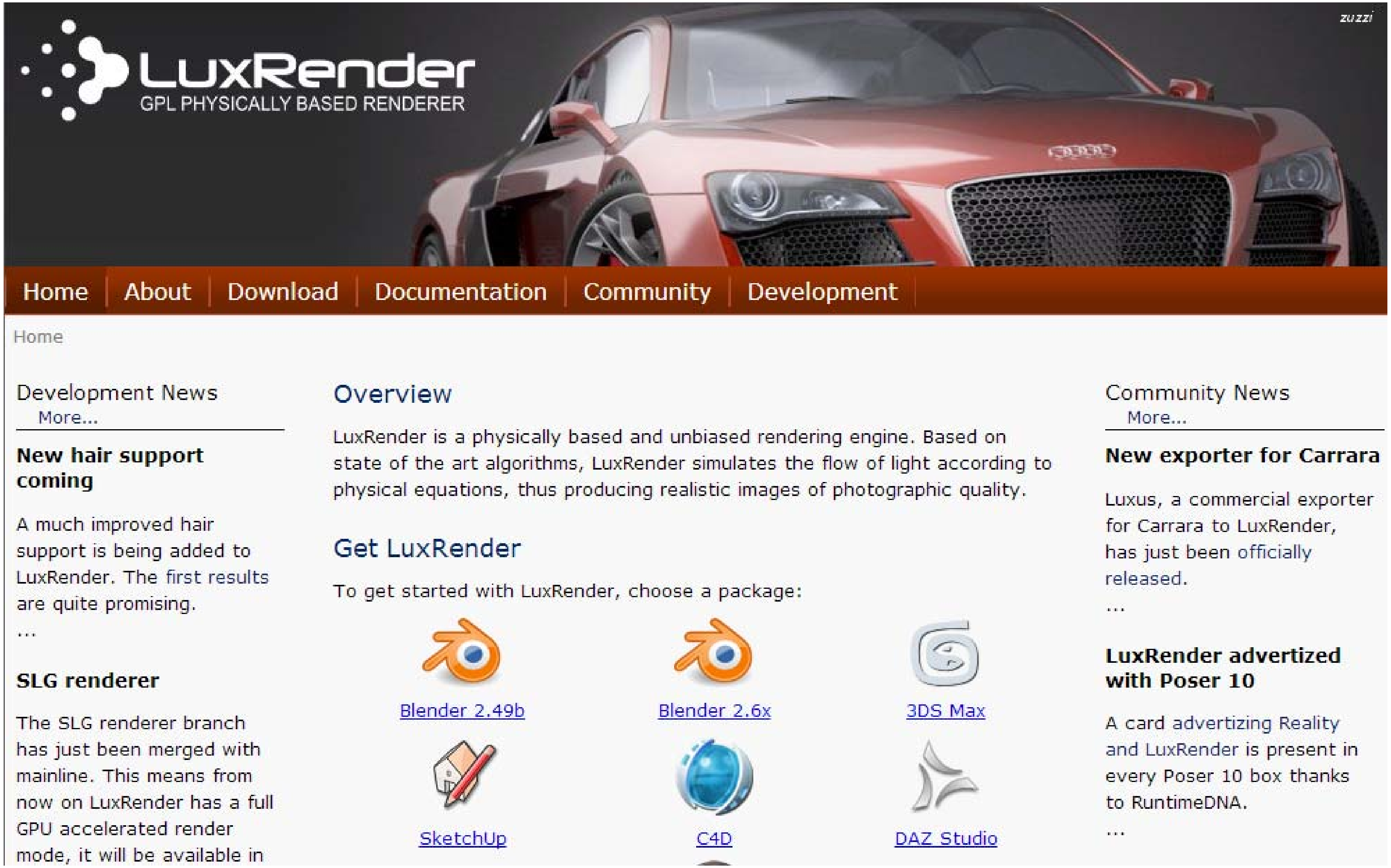
1b ⟨*find minimum after the ith element* 1b⟩≡ (1a)

```
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 could be more efficient)
- Source code browser

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

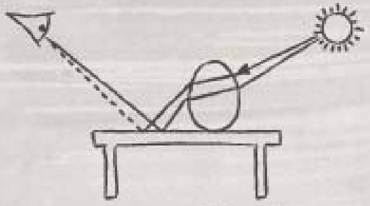


Mitsuba (http://www.mitsuba-renderer.org/)

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

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

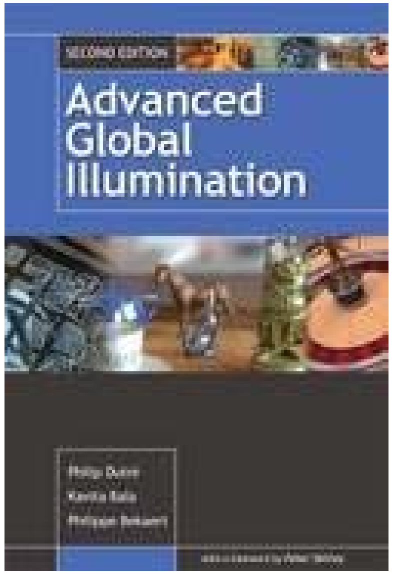
New features of pbrt2

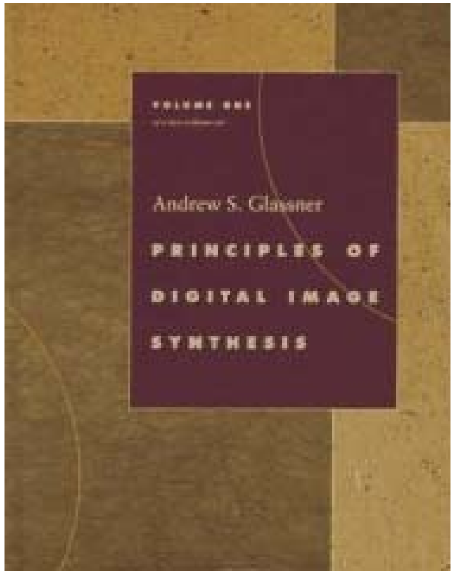
- Remove plug-in architecture, but still an *extensible* architecture
- Add multi-thread support (automatic or --ncores)
- OpenEXR is recommended, not required
- HBV is added and becomes default
- Can be full spectral, do it at compile time
- Animation is supported
- Instant global illumination, extended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

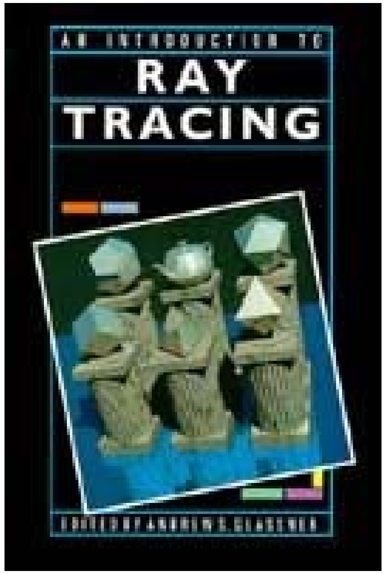
New features of pbrt2

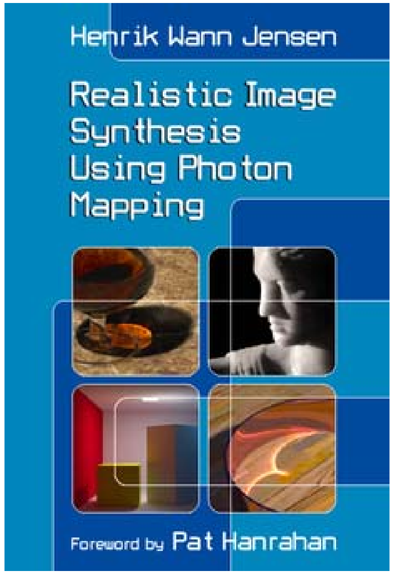
- BSSRDF is added
- Metropolis light transport
- Precomputed radiance transfer
- Support measured BRDF

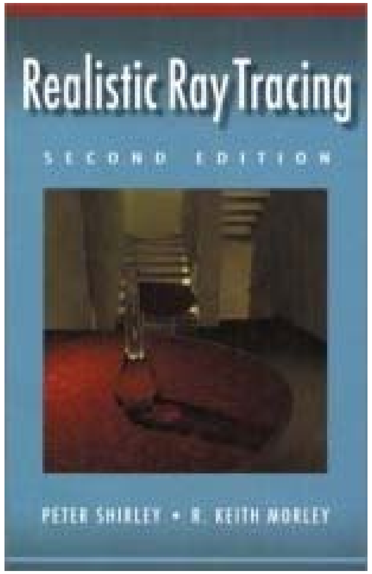
Reference books

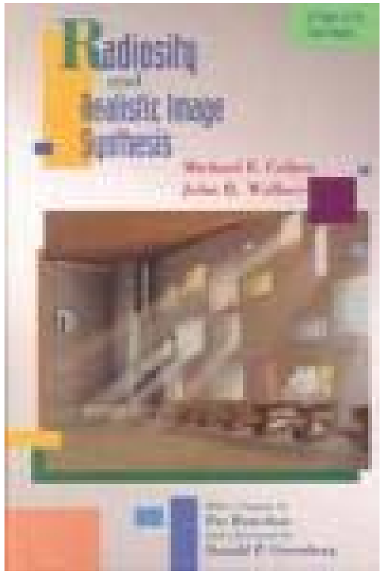


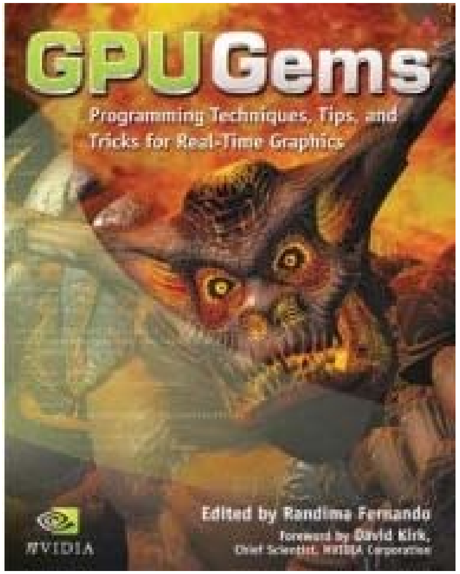


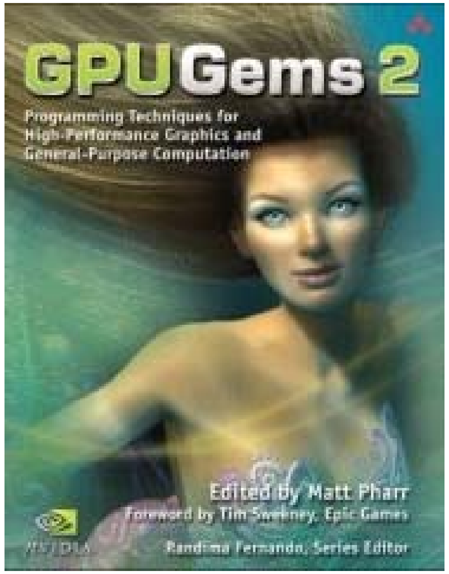












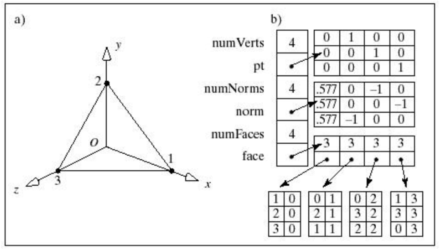
References

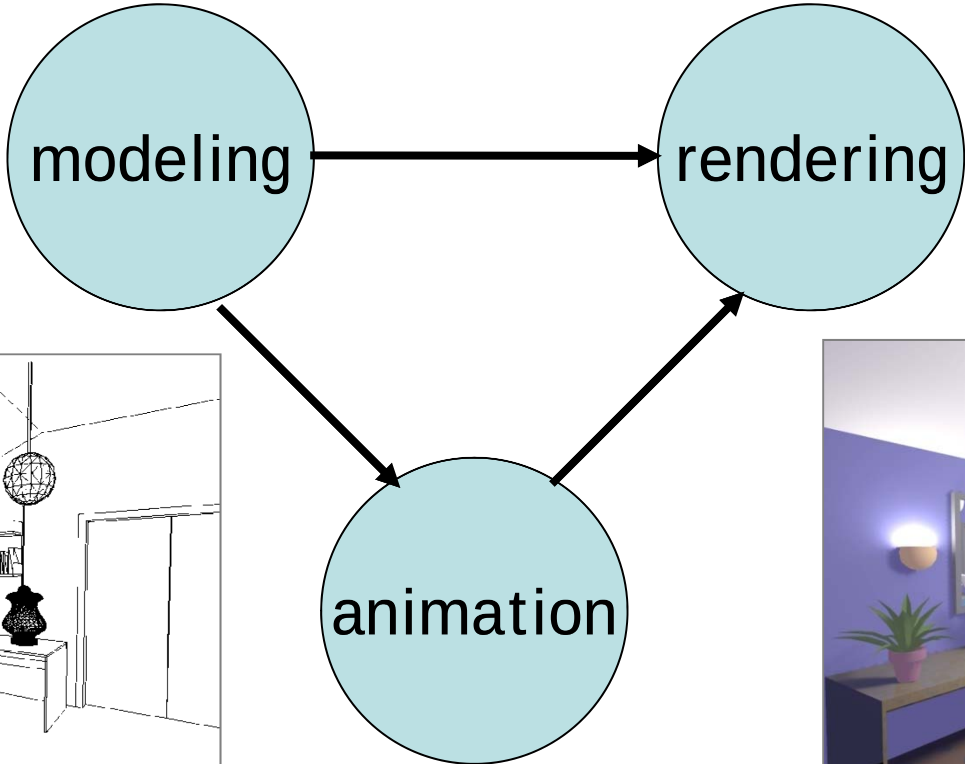
- SIGGRAPH proceedings
- SIGGRAPH Asia proceedings
- Proceedings of Eurographics Symposium on Rendering
- Eurographics proceedings
- Most can be found at this link.

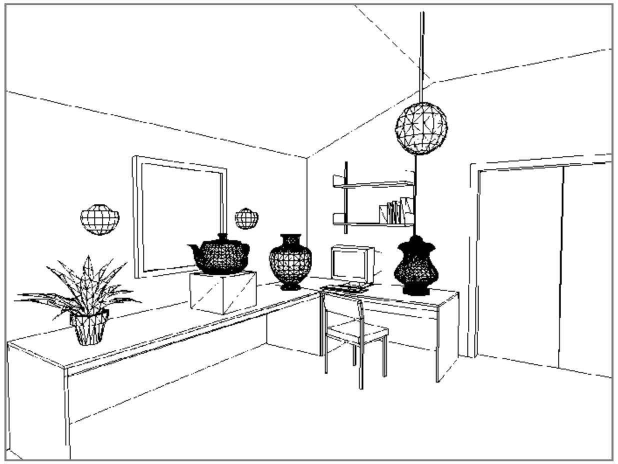
Image synthesis (Rendering)

- 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

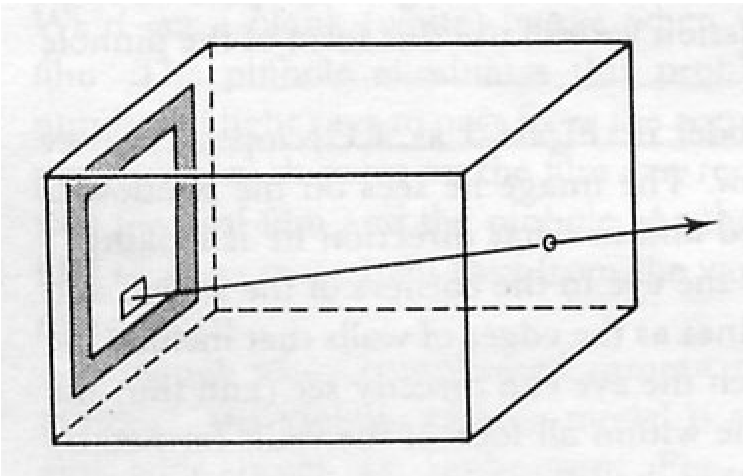
Realism

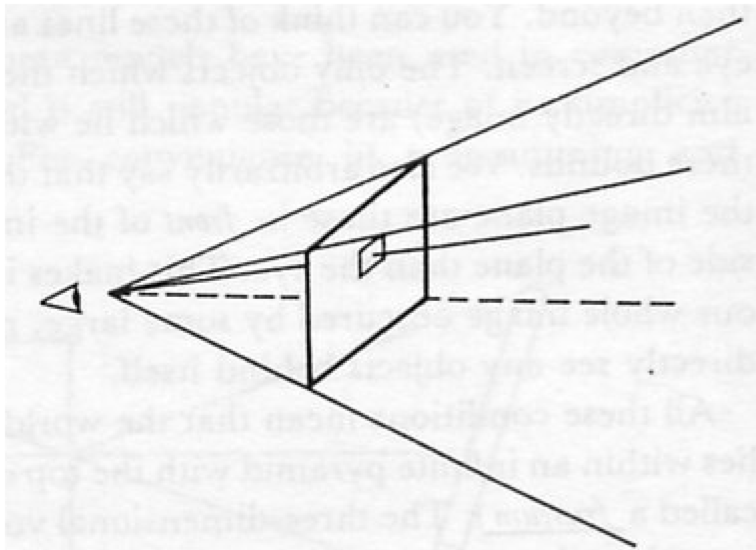
- Shadows
- Reflections (Mirrors)
- Transparency
- Interreflections
- Detail (Textures…)
- Complex Illumination
- Realistic Materials
- And many more

Other types of rendering

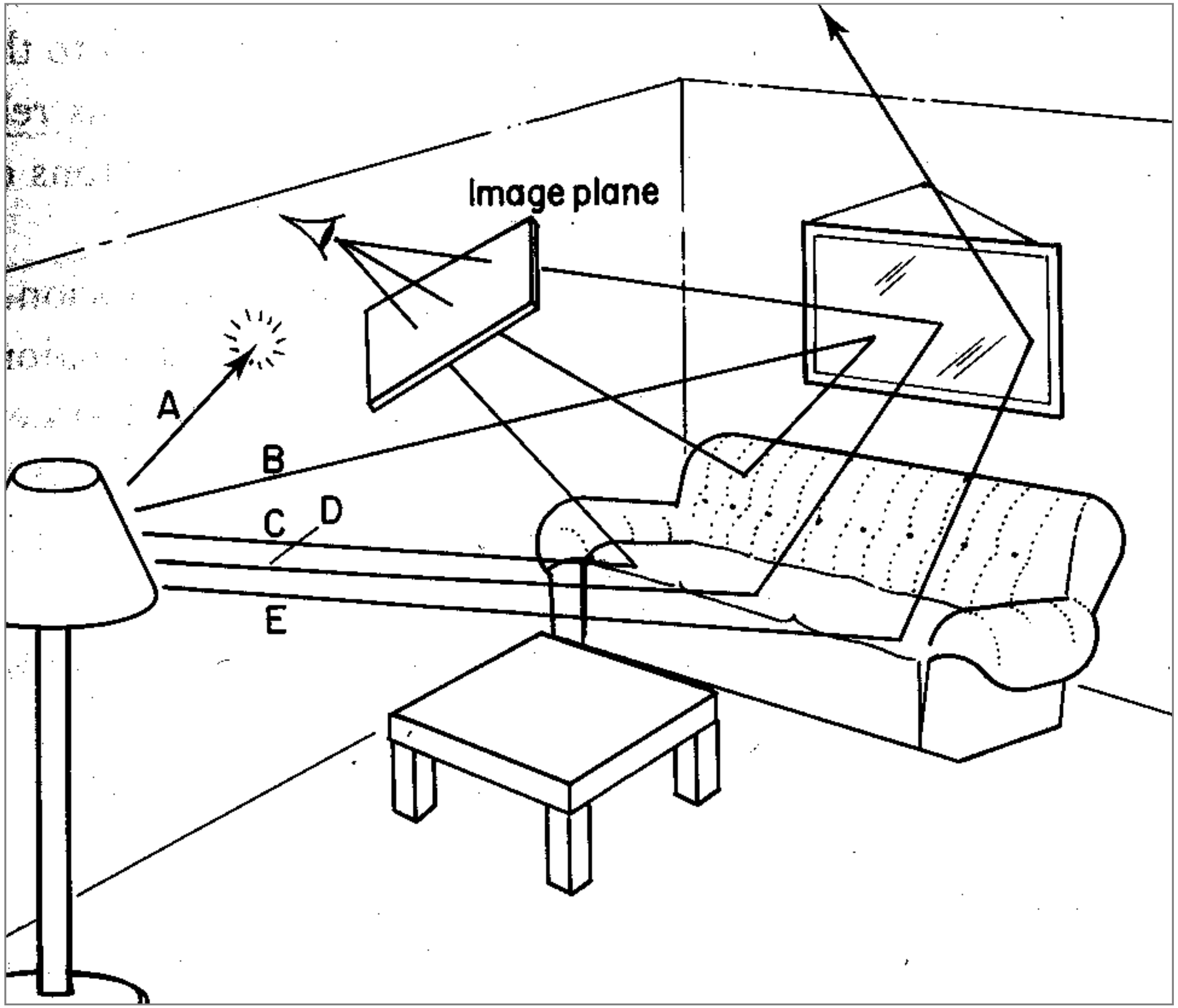
- Non-photorealistic rendering
- Image-based rendering
- Point-based rendering
- Volume rendering
- Perceptual-based rendering
- Artistic rendering

Pinhole camera

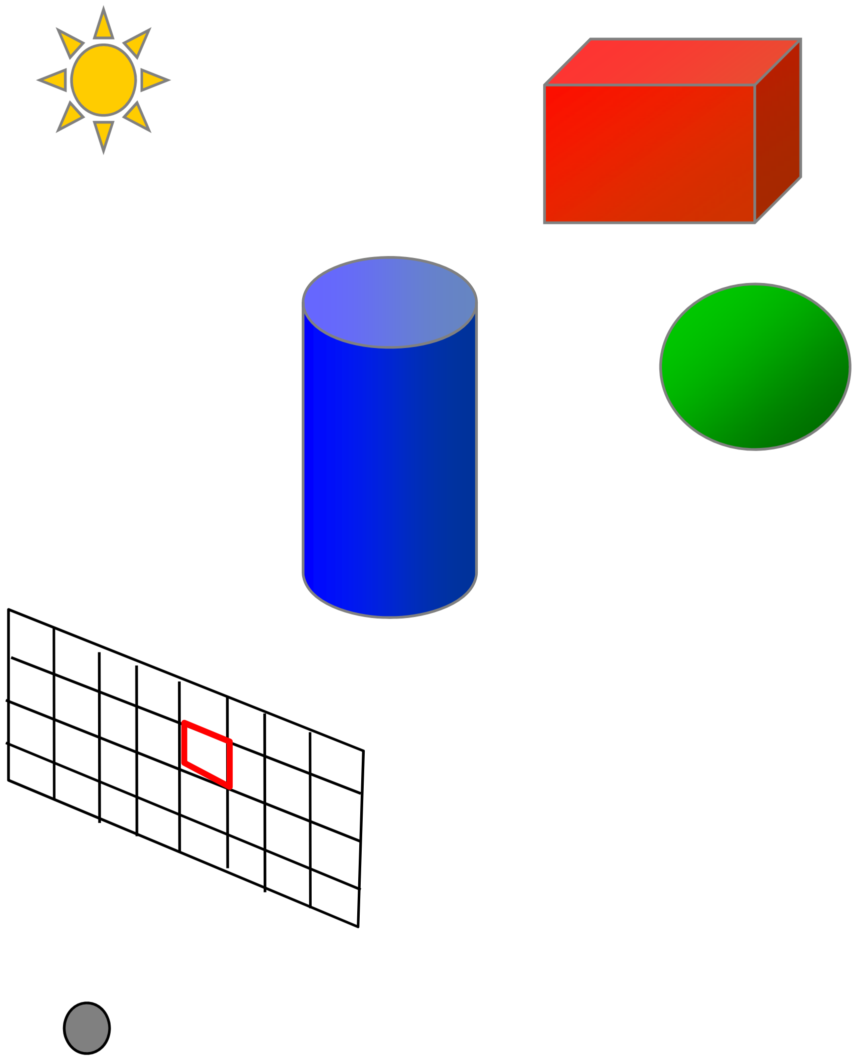




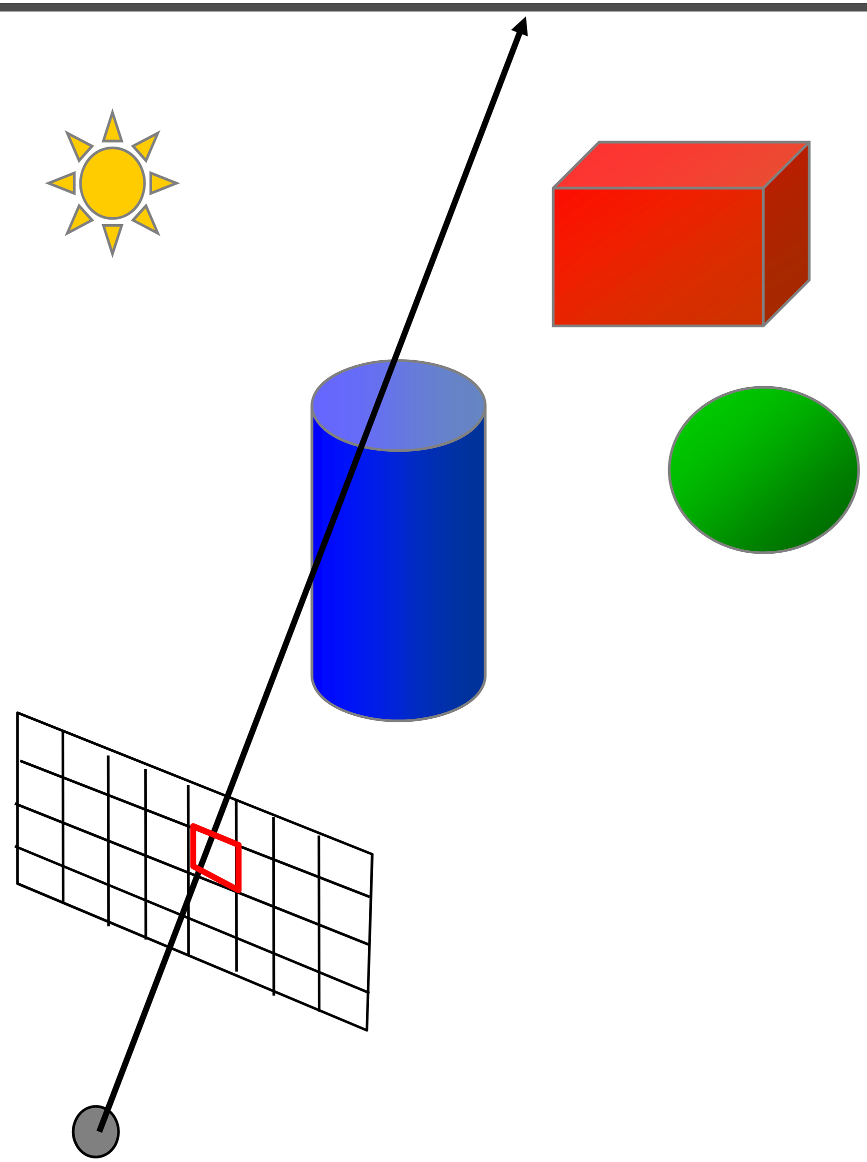
Introduction to ray tracing



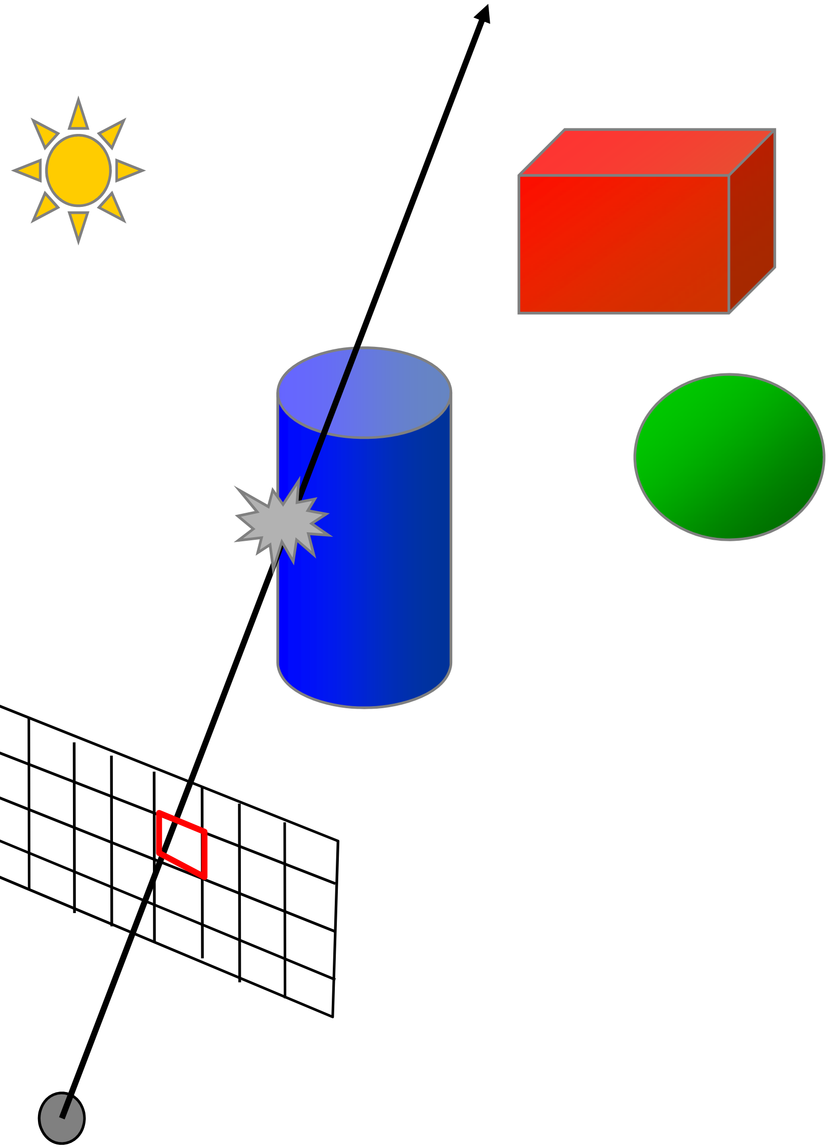
Ray Casting (Appel, 1968)



Ray Casting (Appel, 1968)

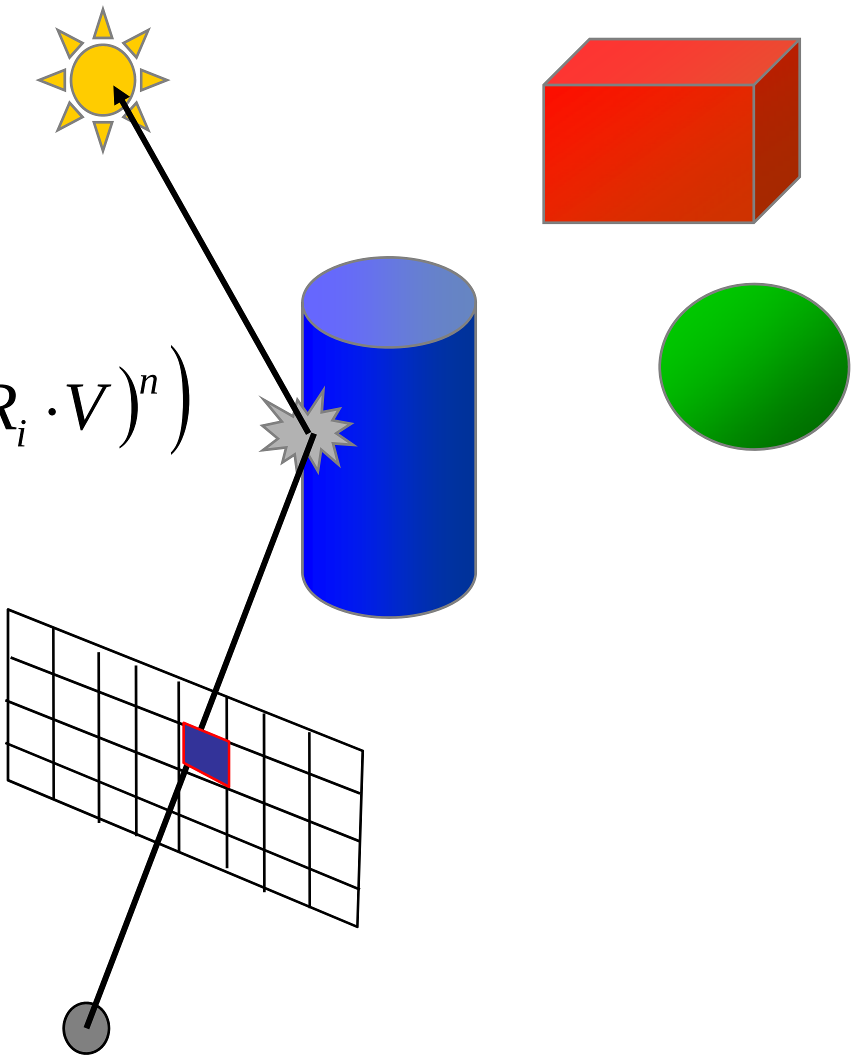


Ray Casting (Appel, 1968)

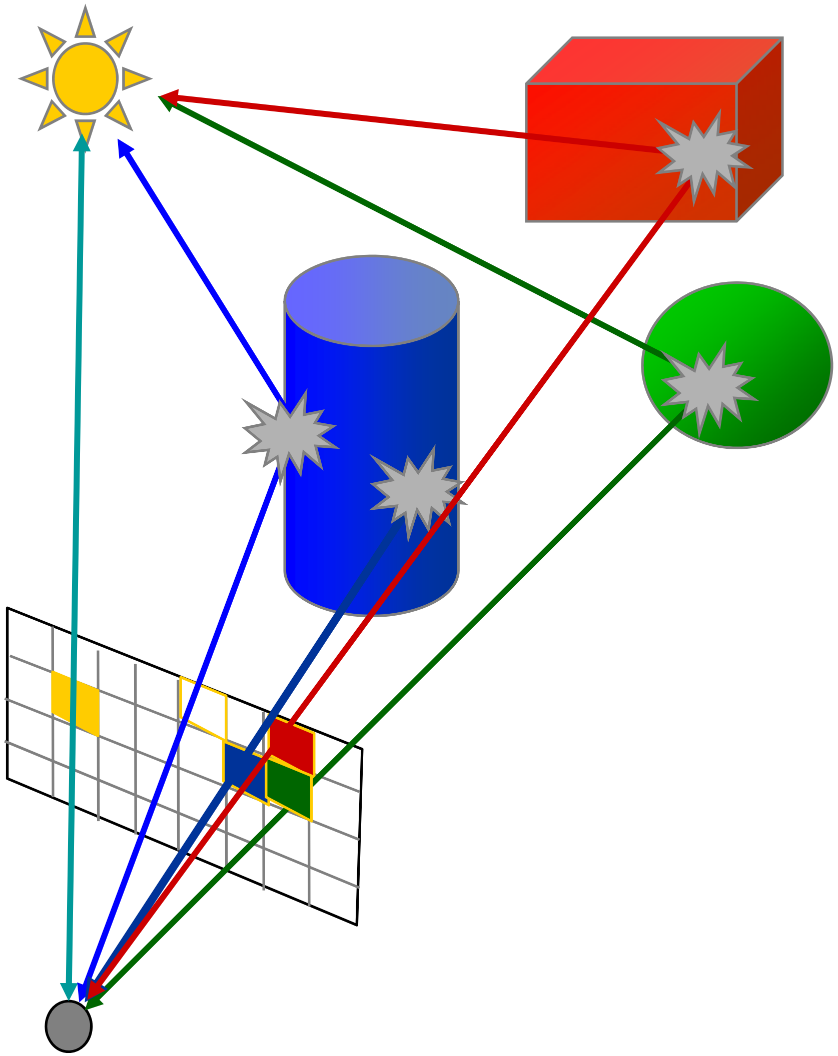


Ray Casting (Appel, 1968)

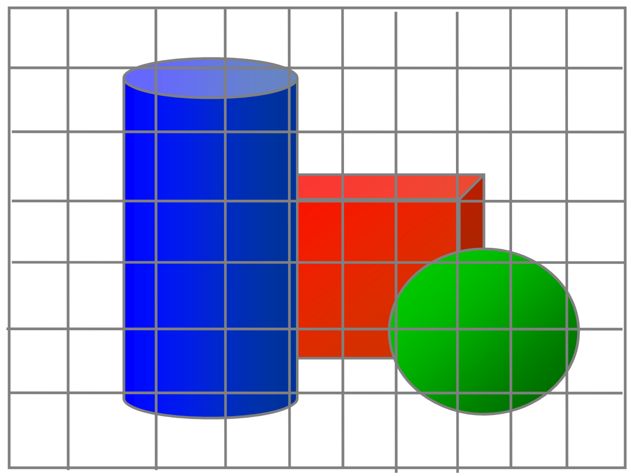
$$k_a I_a + \sum_{i=1}^{nls} I_i \left(k_d \left(L_i \cdot N \right) + k_s \left(R_i \cdot V \right)^n \right)$$



Ray Casting (Appel, 1968)

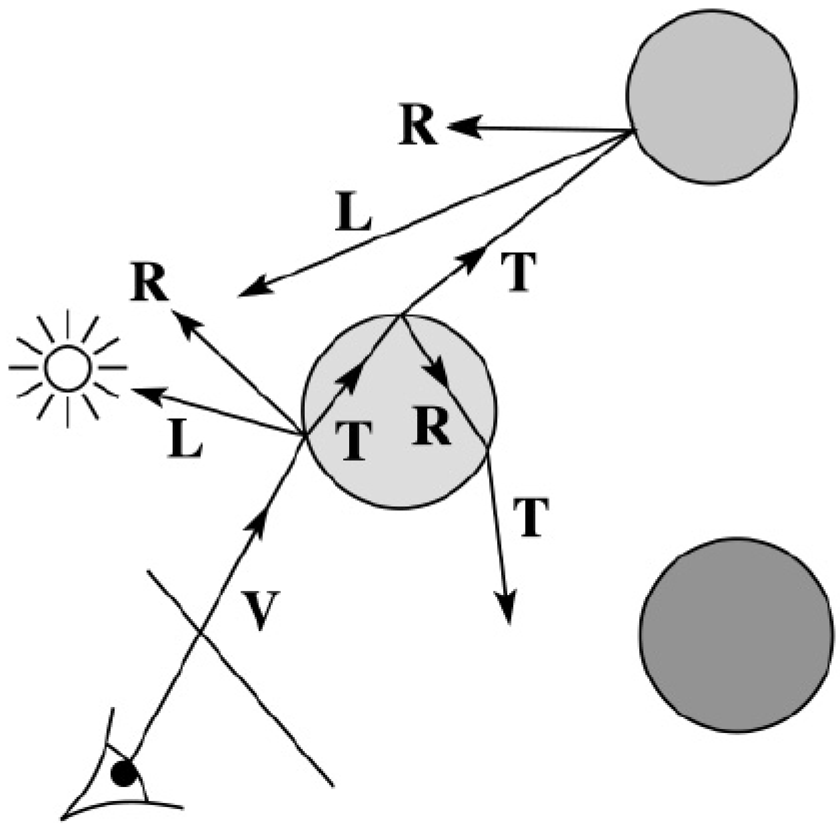


direct illumination



Whitted ray tracing algorithm

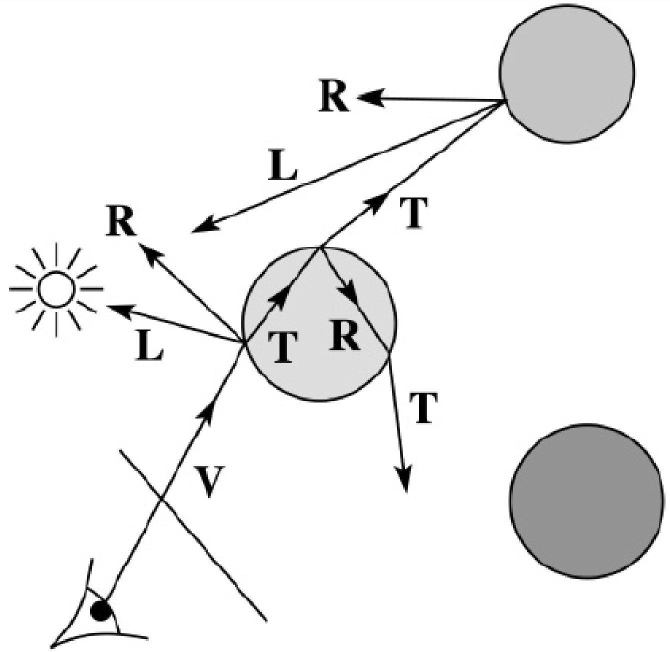
- Combines eye ray tracing + rays to light
- Recursively traces rays



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

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

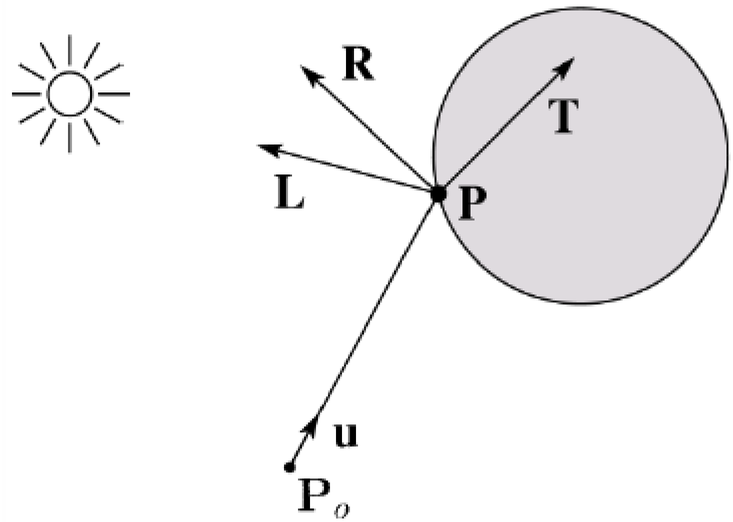
where

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

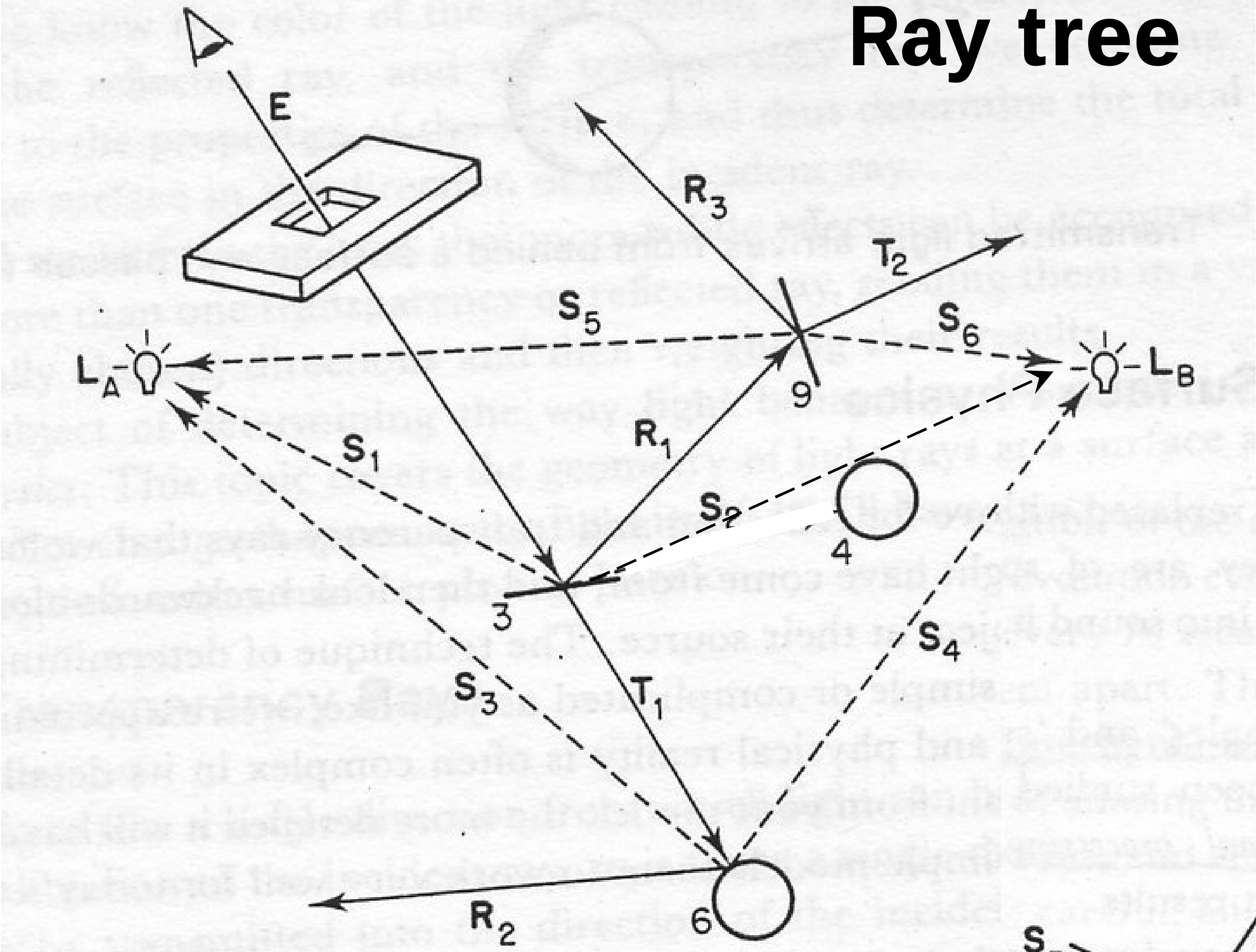
$$I_{reflected} = k_r I(P, \mathbf{R})$$

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

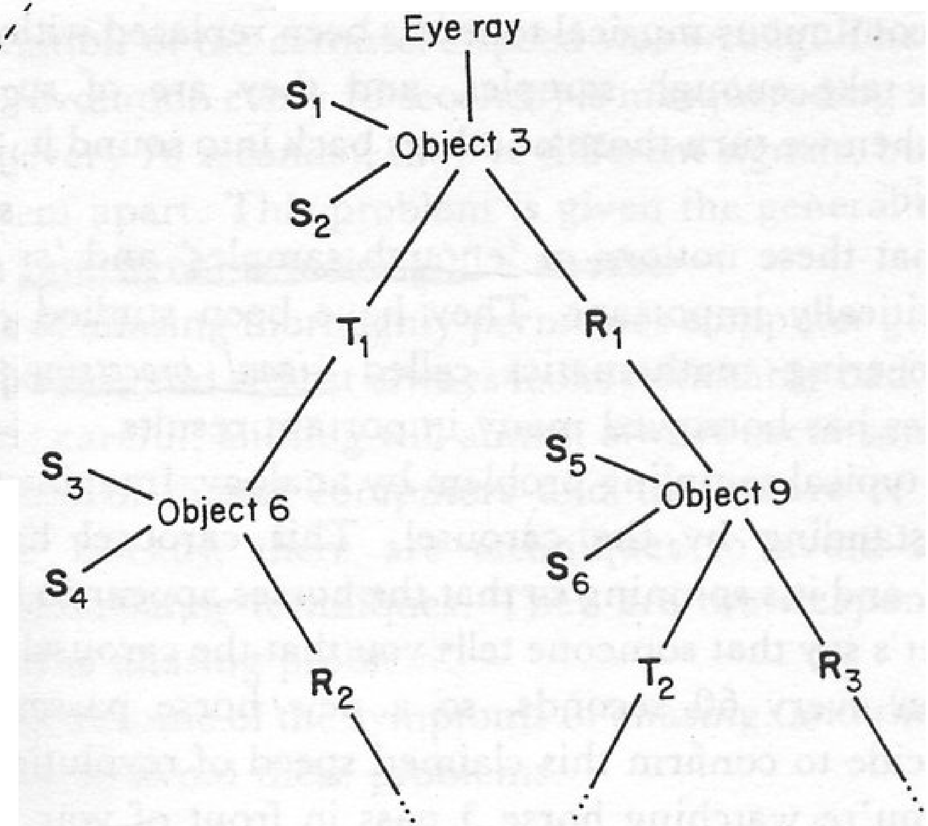
Typically, we set $k_r = k_s$ and k_t



Ray tree

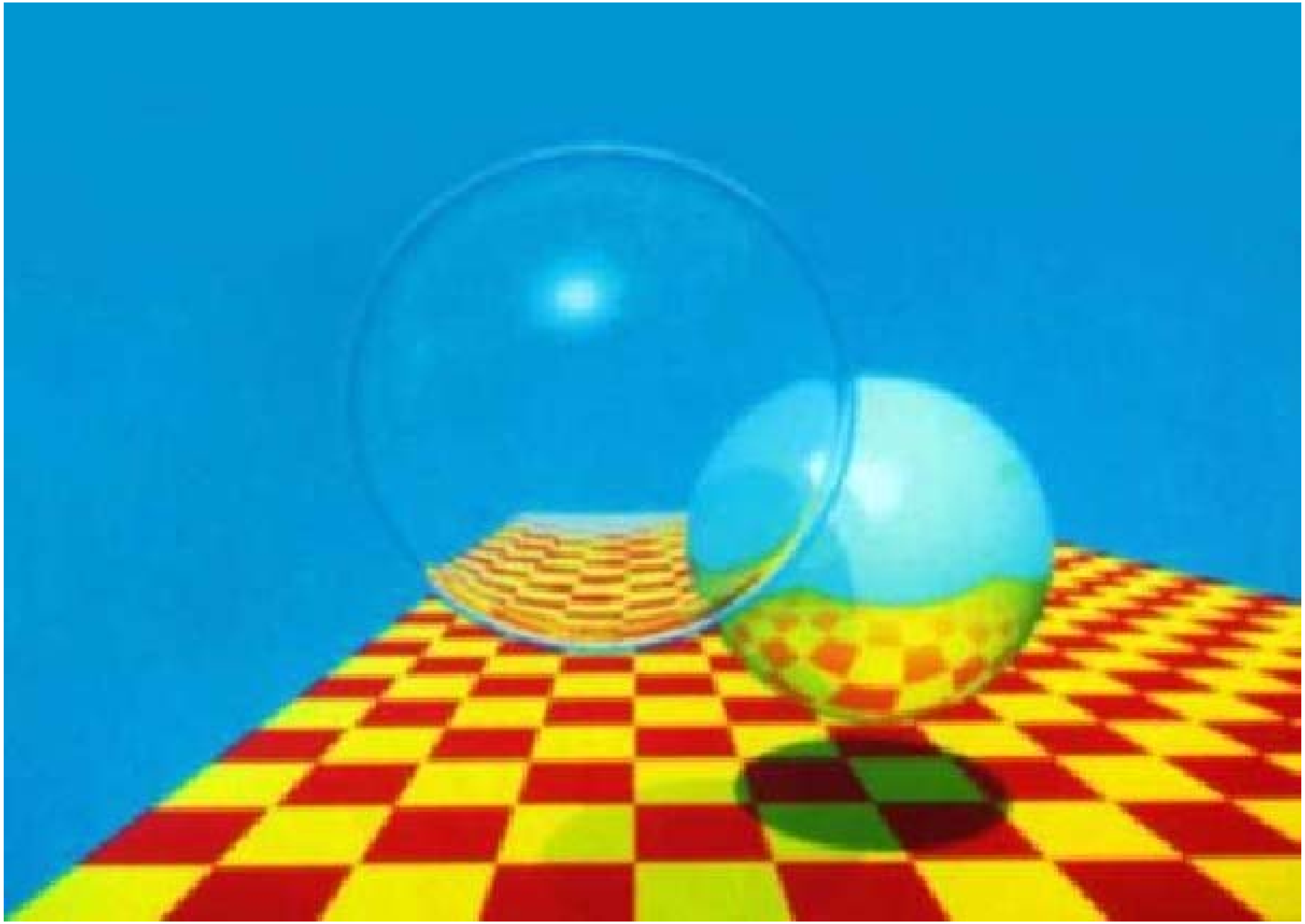


E
R_3
T_2
S_5
S_6
L_A
L_B
9
R_1
S_1
S_2
4
3
S_4
S_3
T_1
6
R_2



Eye ray
S_1
Object 3
S_2
T_1
R_1
S_3
Object 6
S_4
S_5
Object 9
S_6
R_2
T_2
R_3

Recursive ray tracing (Whitted, 1980)



Components of a ray tracer

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

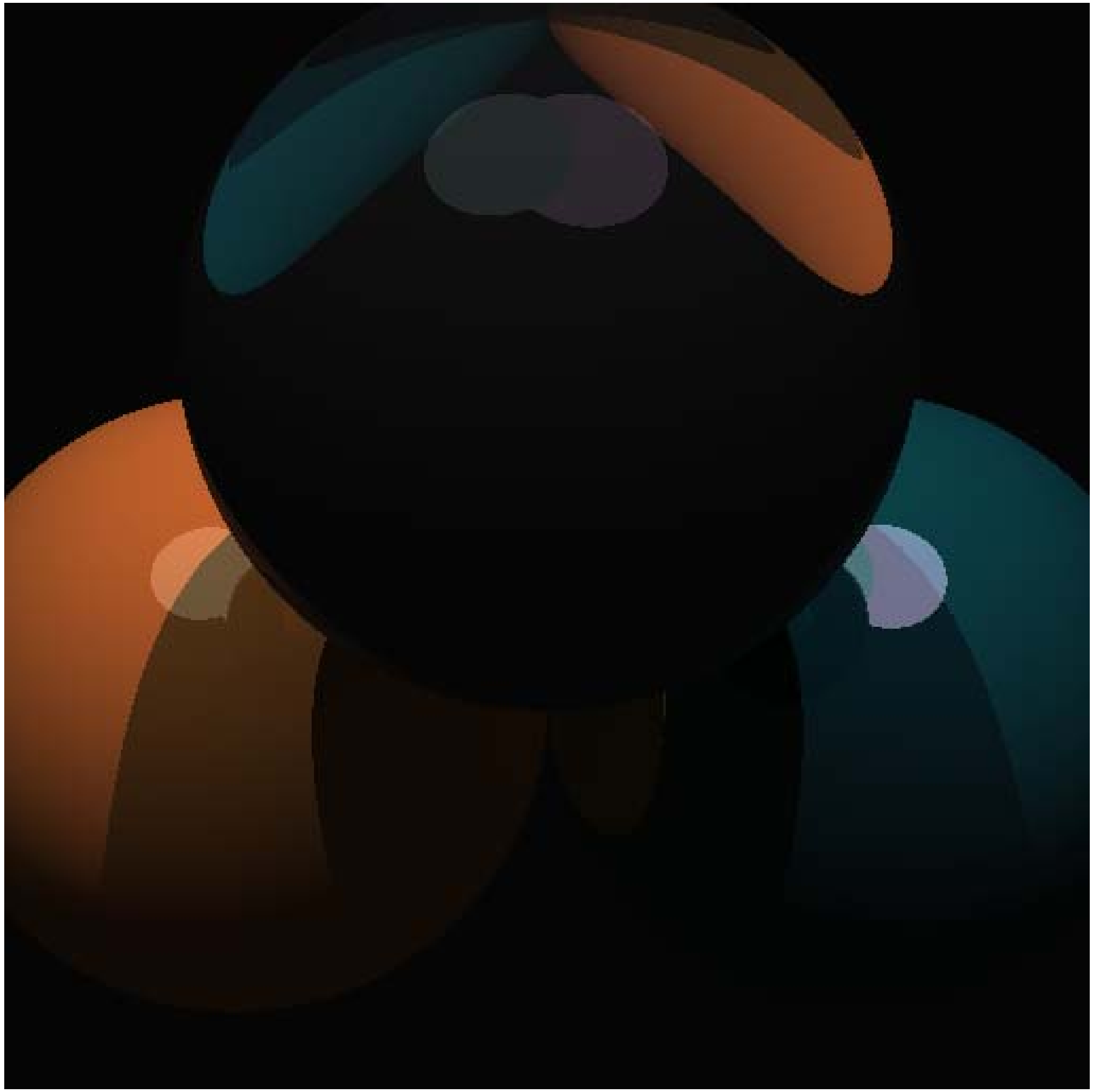
Minimal ray tracer

- 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)

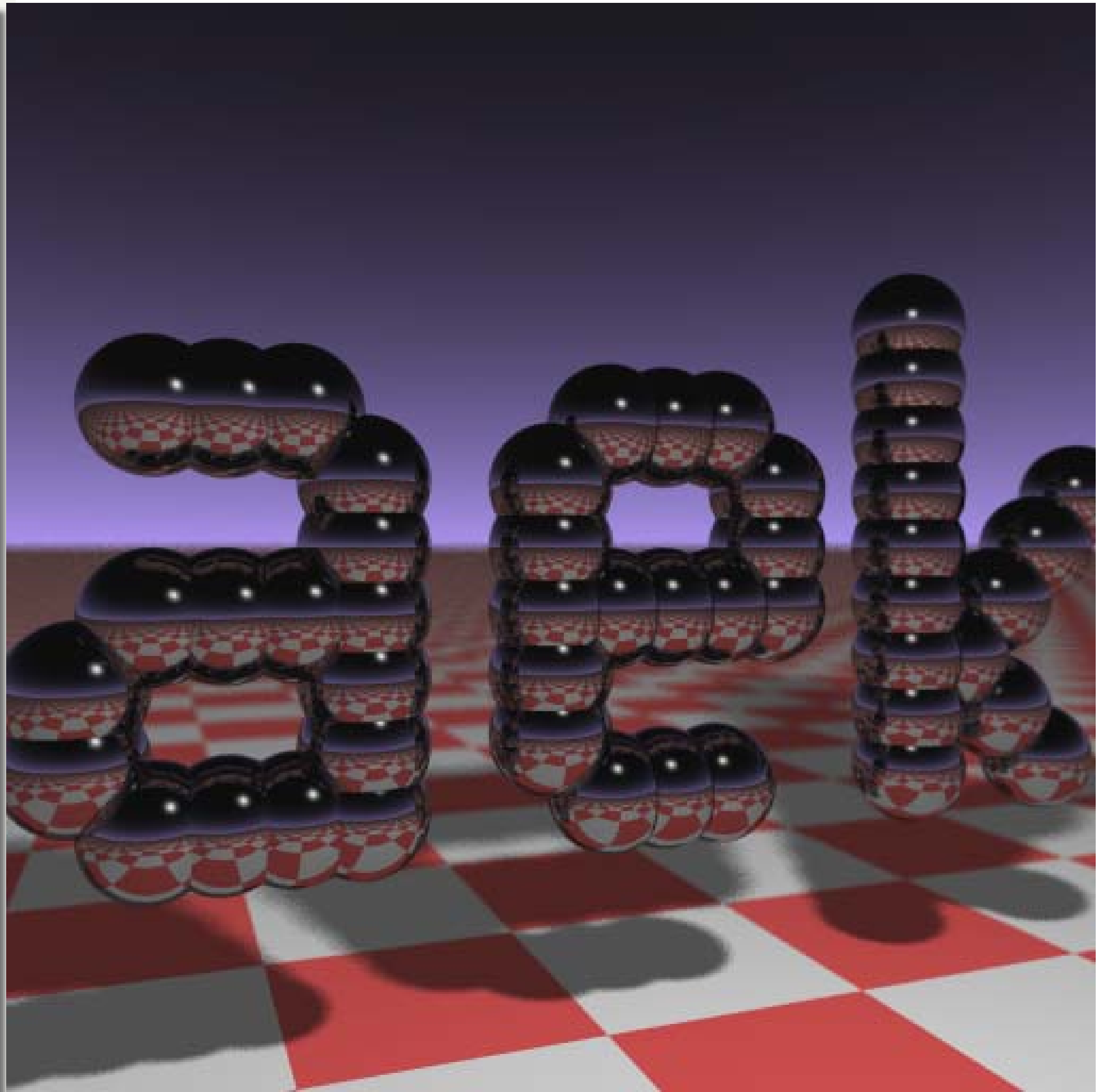
```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{ vec cen,color;
double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9, .05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,
.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8, 1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,
.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!*/
```

What it can do



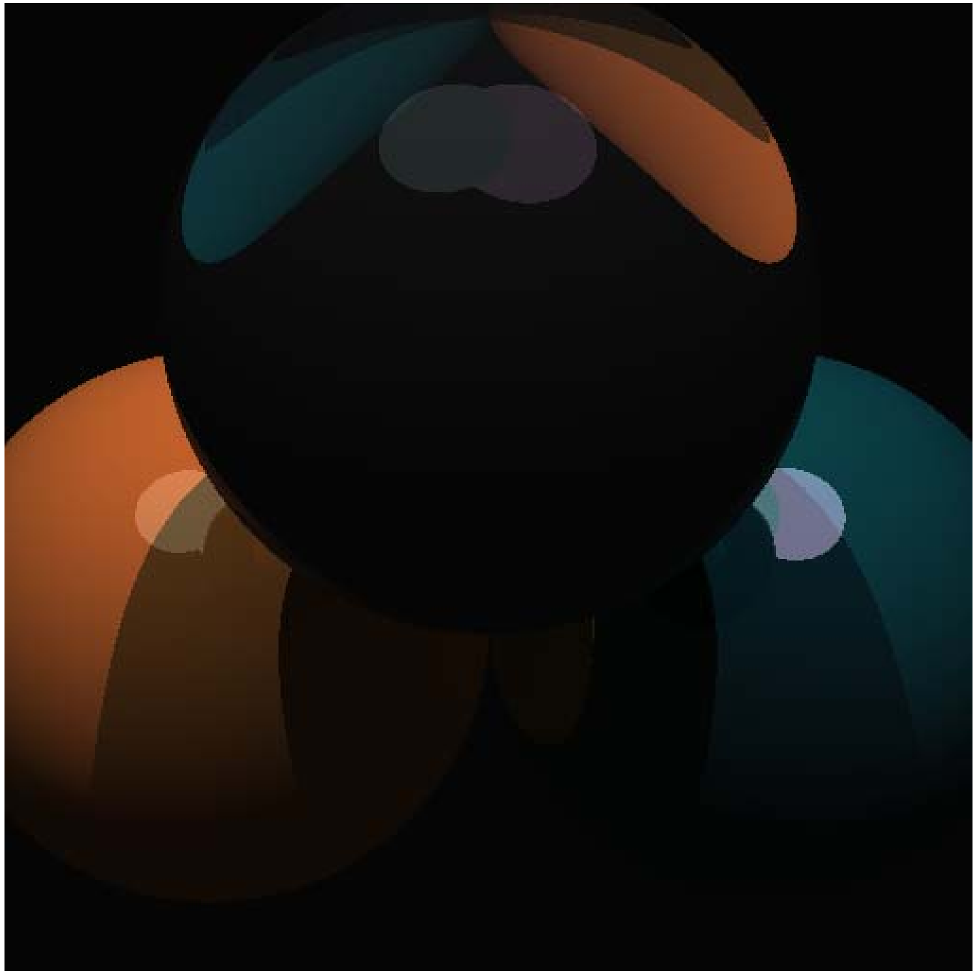
Another business card raytracer

```
#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(){}v operator^(v r
){return v(y*r.z-z*r.y,z*r.x-x*r.z,x*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[]={247570,280596,280600,
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.z;if(.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=o+v(-k
,0,-j-4);f b=p%d,c=p%p-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=l%
n;if(b<0||T(h,l,t,n))b=0;f p=pow(l%r*(b
>0),99);if(m&1){h=h*.2;return((i)(ceil(
h.x)+ceil(h.y))&1?v(3,1,1):v(3,3,3))*(b
*.2+.1);}return v(p,p,p)+S(h,r)*.5;}i
main(){printf("P6 512 512 255 ");v g=!v
(-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)

Really Practical Guide to Path Tracing

$$L_o(\mathrm{p},\omega_o) = L_e(\mathrm{p},\omega_o) + \int_{S^2} f(\mathrm{p},\omega_o,\omega_i) L_i(\mathrm{p},\omega_i) \left|\cos\theta_i\right| \, d\omega_i$$

```cpp

/*
    pbrt source code Copyright(c) 1998-2010 Matt Pharr and Greg Humphreys.

    This file is part of pbrt.

    pbrt is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.  Note that the text contents of
    the book "Physically Based Rendering" are *not* licensed under the
    GNU GPL.

    pbrt is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

 */


// accelerators/bvh.cpp*
#include "stdafx.h"
#include "accelerators/bvh.h"
#include "probes.h"
#include "paramset.h"

// BVHAccel Local Declarations
struct BVHPrimitiveInfo {
    BVHPrimitiveInfo() { }
    BVHPrimitiveInfo(int pn, const BBox &b)
        : primitiveNumber(pn), bounds(b) {
        centroid = .5f * b.pMin + .5f * b.pMax;
    }
    int primitiveNumber;
    Point centroid;
    BBox bounds;
};


struct BVHBuildNode {
    // BVHBuildNode Public Methods
    BVHBuildNode() { children[0] = children[1] = NULL; }
    void InitLeaf(uint32_t first, uint32_t n, const BBox &b) {
        firstPrimOffset = first;
        nPrimitives = n;
        bounds = b;
    }
    void InitInterior(uint32_t axis, BVHBuildNode *c0, BVHBuildNode *c1) {
        children[0] = c0;
        children[1] = c1;
        bounds = Union(c0->bounds, c1->bounds);
        splitAxis = axis;
        nPrimitives = 0;
    }
    BBox bounds;
    BVHBuildNode *children[2];
```

```cpp
    uint32_t splitAxis, firstPrimOffset, nPrimitives;
};


struct CompareToMid {
    CompareToMid(int d, float m) { dim = d; mid = m; }
    int dim;
    float mid;
    bool operator()(const BVHPrimitiveInfo &a) const {
        return a.centroid[dim] < mid;
    }
};


struct ComparePoints {
    ComparePoints(int d) { dim = d; }
    int dim;
    bool operator()(const BVHPrimitiveInfo &a,
                    const BVHPrimitiveInfo &b) const {
        return a.centroid[dim] < b.centroid[dim];
    }
};


struct CompareToBucket {
    CompareToBucket(int split, int num, int d, const BBox &b)
        : centroidBounds(b)
    { splitBucket = split; nBuckets = num; dim = d; }
    bool operator()(const BVHPrimitiveInfo &p) const;

    int splitBucket, nBuckets, dim;
    const BBox &centroidBounds;
};


bool CompareToBucket::operator()(const BVHPrimitiveInfo &p) const {
    int b = nBuckets * ((p.centroid[dim] - centroidBounds.pMin[dim]) /
            (centroidBounds.pMax[dim] - centroidBounds.pMin[dim]));
    if (b == nBuckets) b = nBuckets-1;
    Assert(b >= 0 && b < nBuckets);
    return b <= splitBucket;
}


struct LinearBVHNode {
    BBox bounds;
    union {
        uint32_t primitivesOffset;    // leaf
        uint32_t secondChildOffset;   // interior
    };

    uint8_t nPrimitives;  // 0 -> interior node
    uint8_t axis;         // interior node: xyz
    uint8_t pad[2];       // ensure 32 byte total size
};


static inline bool IntersectP(const BBox &bounds, const Ray &ray,
        const Vector &invDir, const uint32_t dirIsNeg[3]) {
    // Check for ray intersection against $x$ and $y$ slabs
```

```cpp
    float tmin =  (bounds[  dirIsNeg[0]].x - ray.o.x) * invDir.x;
    float tmax =  (bounds[1-dirIsNeg[0]].x - ray.o.x) * invDir.x;
    float tymin = (bounds[  dirIsNeg[1]].y - ray.o.y) * invDir.y;
    float tymax = (bounds[1-dirIsNeg[1]].y - ray.o.y) * invDir.y;
    if ((tmin > tymax) || (tymin > tmax))
        return false;
    if (tymin > tmin) tmin = tymin;
    if (tymax < tmax) tmax = tymax;

    // Check for ray intersection against $z$ slab
    float tzmin = (bounds[  dirIsNeg[2]].z - ray.o.z) * invDir.z;
    float tzmax = (bounds[1-dirIsNeg[2]].z - ray.o.z) * invDir.z;
    if ((tmin > tzmax) || (tzmin > tmax))
        return false;
    if (tzmin > tmin)
        tmin = tzmin;
    if (tzmax < tmax)
        tmax = tzmax;
    return (tmin < ray.maxt) && (tmax > ray.mint);
}



// BVHAccel Method Definitions
BVHAccel::BVHAccel(const vector<Reference<Primitive> > &p,
                   uint32_t mp, const string &sm) {
    maxPrimsInNode = min(255u, mp);
    for (uint32_t i = 0; i < p.size(); ++i)
        p[i]->FullyRefine(primitives);
    if (sm == "sah")         splitMethod = SPLIT_SAH;
    else if (sm == "middle") splitMethod = SPLIT_MIDDLE;
    else if (sm == "equal")  splitMethod = SPLIT_EQUAL_COUNTS;
    else {
        Warning("BVH split method \"%s\" unknown.  Using \"sah\".",
                sm.c_str());
        splitMethod = SPLIT_SAH;
    }

    if (primitives.size() == 0) {
        nodes = NULL;
        return;
    }
    // Build BVH from _primitives_
    PBRT_BVH_STARTED_CONSTRUCTION(this, primitives.size());

    // Initialize _buildData_ array for primitives
    vector<BVHPrimitiveInfo> buildData;
    buildData.reserve(primitives.size());
    for (uint32_t i = 0; i < primitives.size(); ++i) {
        BBox bbox = primitives[i]->WorldBound();
        buildData.push_back(BVHPrimitiveInfo(i, bbox));
    }

    // Recursively build BVH tree for primitives
    MemoryArena buildArena;
    uint32_t totalNodes = 0;
    vector<Reference<Primitive> > orderedPrims;
    orderedPrims.reserve(primitives.size());
    BVHBuildNode *root = recursiveBuild(buildArena, buildData, 0,
                                        primitives.size(), &totalNodes,
```

Complex lighting

Complex lighting

Refraction/dispersion

Caustics

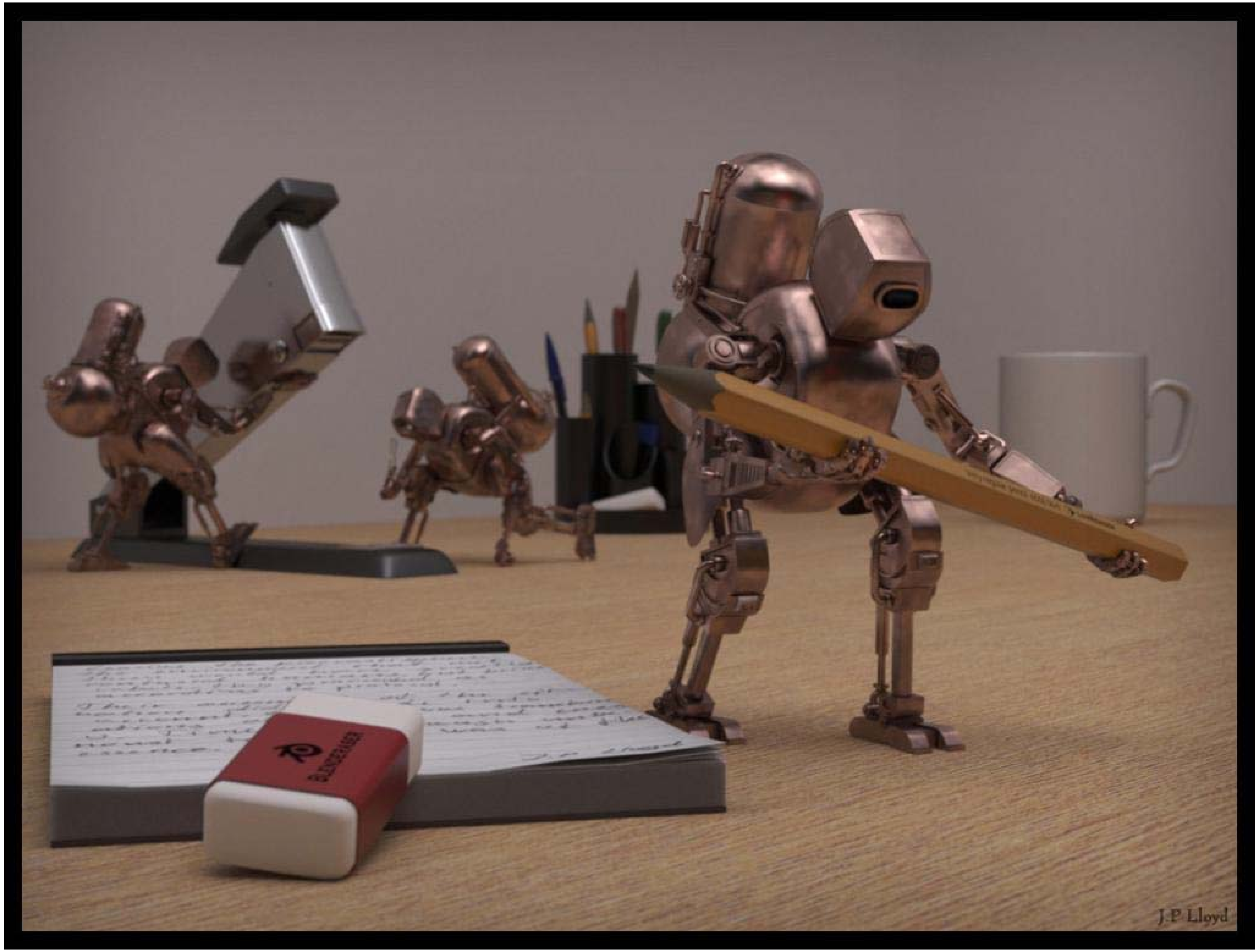
Realistic materials

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

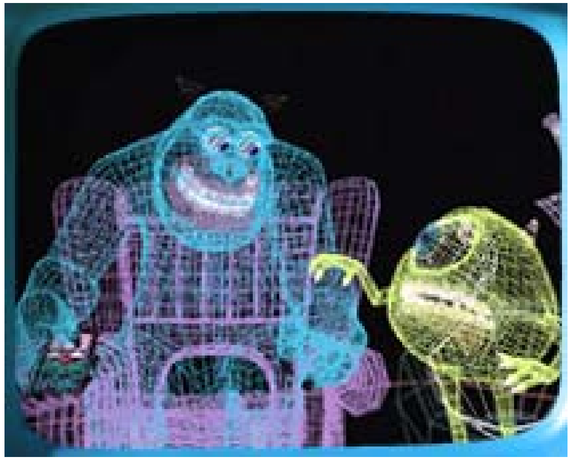
storyboard

voice

storyreal

look and feel

Animation production pipeline



modeling/articulation

layout

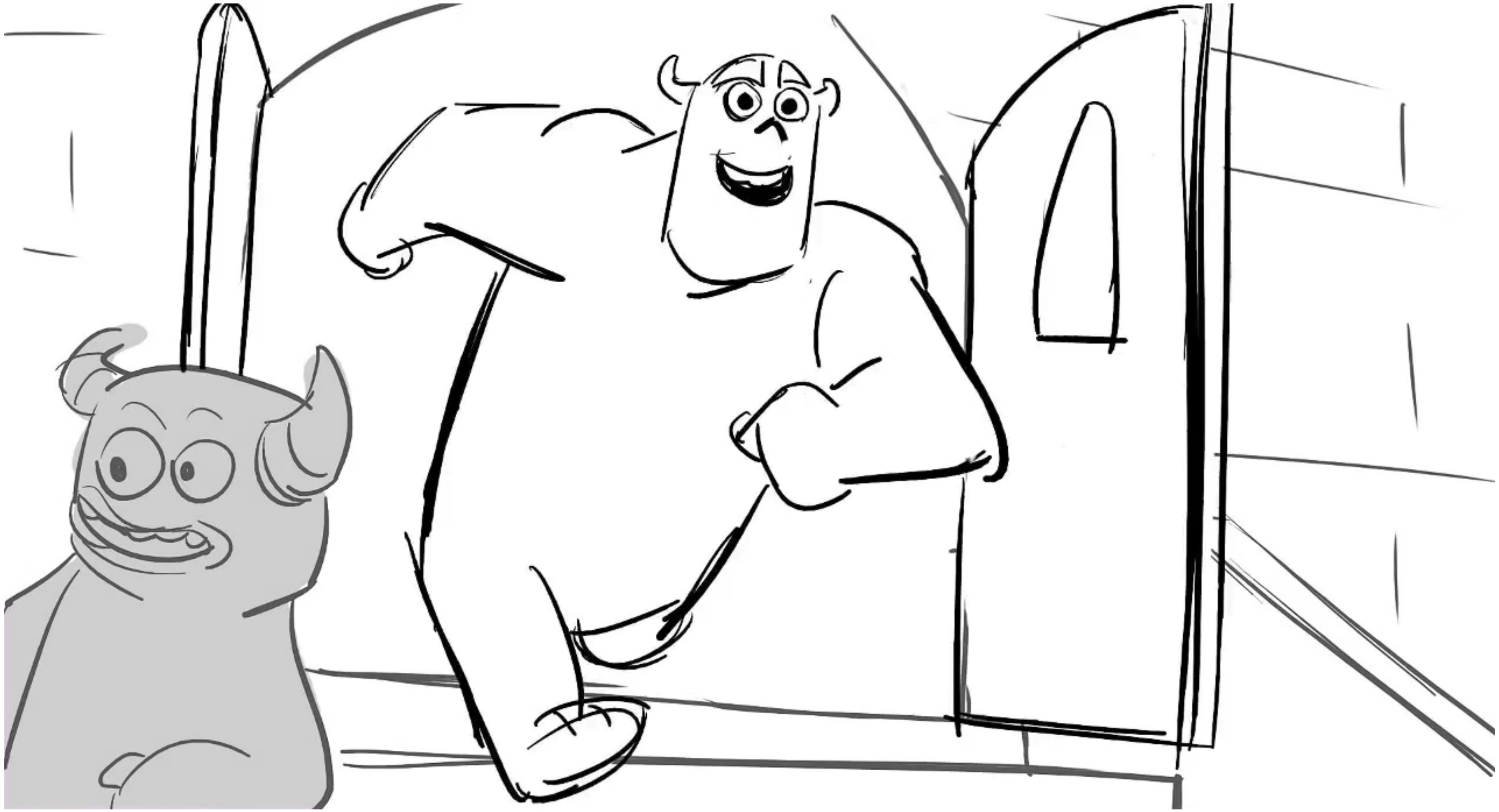
animation

shading/lighting

rendering

final touch

Monster University Progression



Pixar in a Box

Ray tracing finally catches up

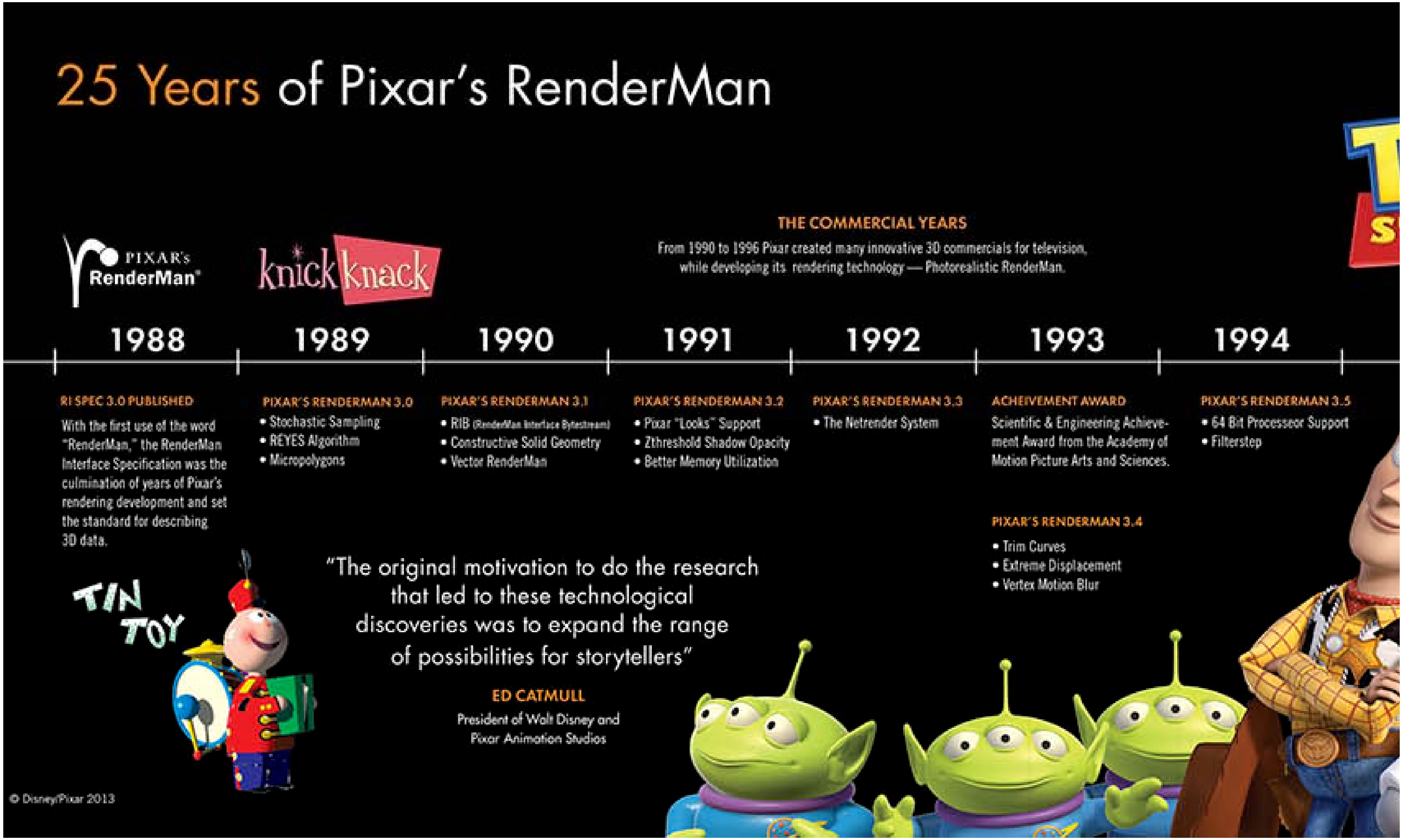
Arnold renderer

Manuka (Weta digital)

http://www.fxguide.com/featured/manuka-weta-digitals-new-renderer/

Hyperion (Disney)

Pixar Renderman Timeline



Monster University

The blue umbrella

Homework #0

- Download and install pbrt2.
- Run several examples
- Set it up in a debugger environment so that you can trace the code
- Optionally, create your own scene