CS6965: PARALLEL HARDWARE RAY TRACING

Fall 2011

What is Ray Tracing?

- A computer graphic rendering technique that simulates optics
  - Can generate very realistic-looking images
  - Can take a long time to create those images
- David Luebke (NVIDIA):
  - “Rasterization is fast, but needs cleverness to support complex visual effects. Ray tracing supports complex visual effects, but needs cleverness to be fast.”

What's the plan?

- Design and implement Ray Tracing variations on a (simulated) parallel machine
  - Start with everyone getting up to speed on a basic ray tracer
  - Then do projects on more advanced variations

What's the plan?

- Hardware Platform
  - TRaX – a many-core architecture designed for ray tracing
  - Quite different than a commercial GPU
  - Exists as a detailed simulator
- Software Platform
  - Compiler based on llvm
  - Generates x86 code (for running on your machine)
  - Also generates TRaX assembly (for the simulator)

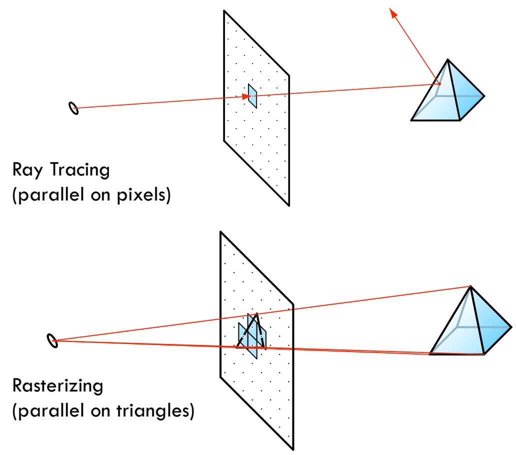
Projects

- Extend our understanding of HW support for advanced RT techniques…
  - Path tracing
  - Beam tracing
  - Ray bundles
  - Photon mapping
  - Motion blur
  - Ambient occlusion
  - Animated scenes
  - Participating media (fog, smoke, etc.)
  - Alternative acceleration structures (grid, KD tree, etc.)
  - Procedural texturing or geometry
  - power saving techniques

Projects

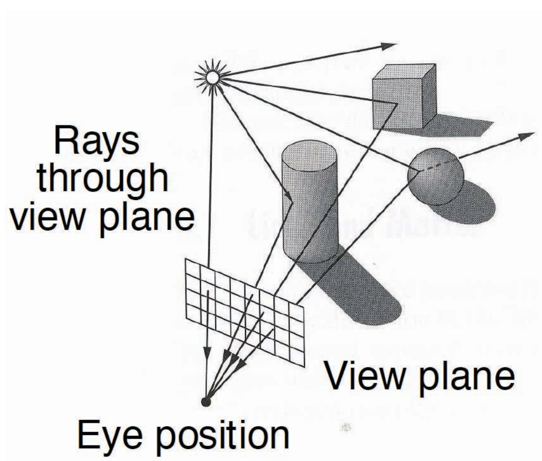
- Extend our architecture that supports advanced RT techniques… (i.e. enhance the simulator)
  - Memory system enhancements
  - Address generation units
  - New function units
  - Communication between thread processors
  - Function unit chaining – configurable macro instructions
  - Support for streaming data access

Ray Tracing vs. Rasterization



RAY TRACING

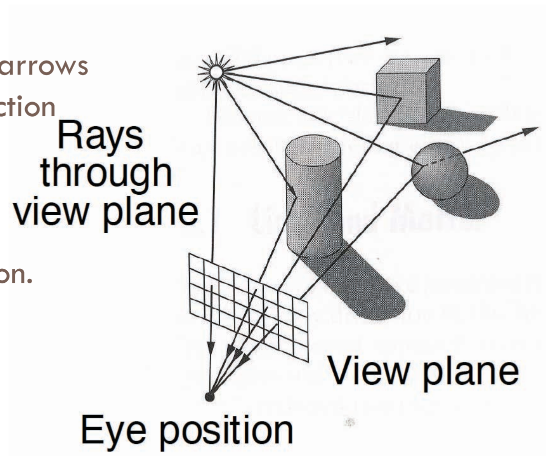
- Color each pixel based on the radiance from each visible surface



Tom Funkhouser, Princeton

RAY TRACING

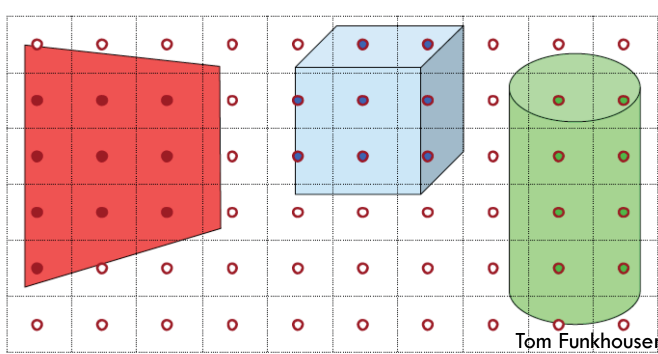
- Color each pixel based on the radiance from each visible surface
  - Note that these arrows point in the direction of the radiance. We normally trace rays in the other direction.



Tom Funkhouser, Princeton

Ray Tracing

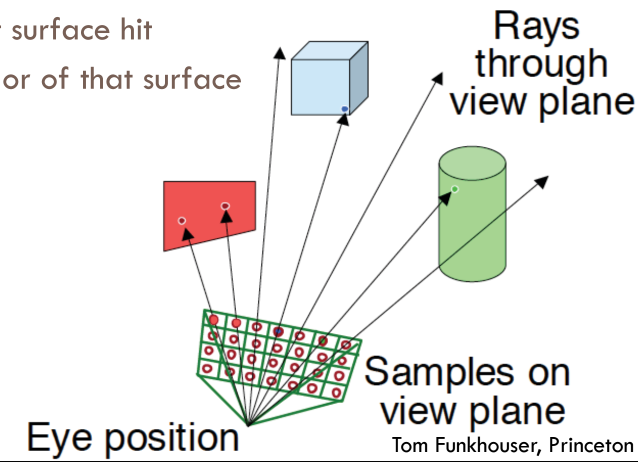
- For each sample
  - Construct a ray from the eye position through view plane
  - Find the first surface hit
  - Compute color of that surface



Tom Funkhouser, Princeton

Ray Tracing

- For each sample
  - Construct a ray from the eye position through view plane
  - Find the first surface hit
  - Compute color of that surface



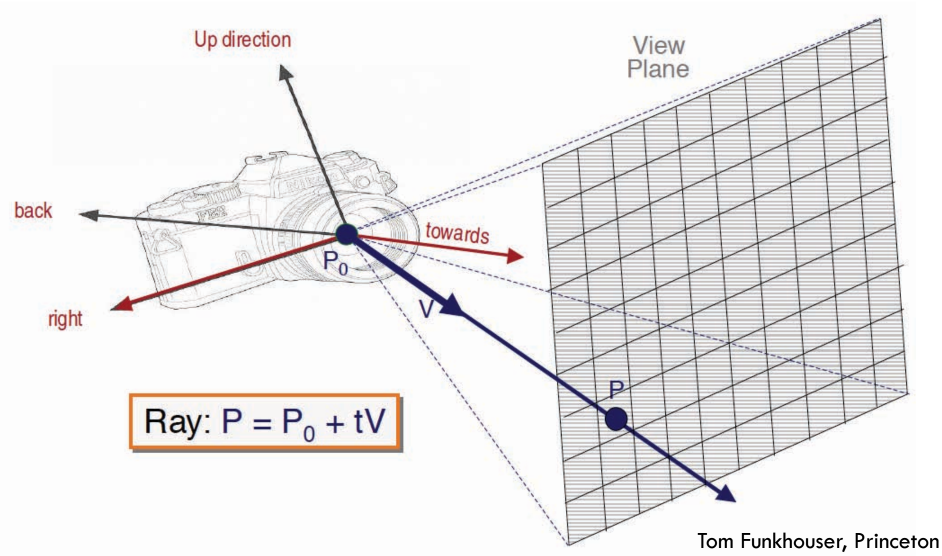
Tom Funkhouser, Princeton

Simple Ray Casting

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

Tom Funkhouser, Princeton

Construct a ray through a pixel



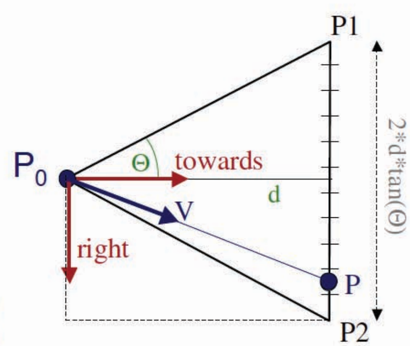
Tom Funkhouser, Princeton

Construct a ray through a pixel

- 2D Example

Θ = frustum half-angle
d = distance to view plane

right = towards x up



$P1 = P_0 + d*towards – d*tan(\Theta)*right$
$P2 = P_0 + d*towards + d*tan(\Theta)*right$

$P = P1 + ((i + 0.5) / width) * (P2 - P1)$
$V = (P - P_0) / ||P - P_0||$

Ray: $P = P_0 + tV$

Tom Funkhouser, Princeton

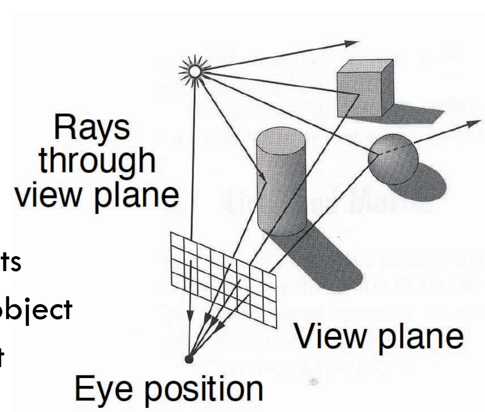
Recursive Ray Tracing

Create scene (objects, materials, lights, camera, background)
Preprocess scene
foreach frame
- foreach pixel
  - foreach sample
    - generate ray
    - intersect ray with objects
    - find normal of closest object
    - shade intersection point

Mutually recursive
Shading can generate new rays…

Recursive Ray Tracing

Create scene (objects, materials, lights, camera, background)
Preprocess scene
foreach frame
- foreach pixel
  - foreach sample
    - generate ray
    - intersect ray with objects
    - find normal of closest object
    - shade intersection point

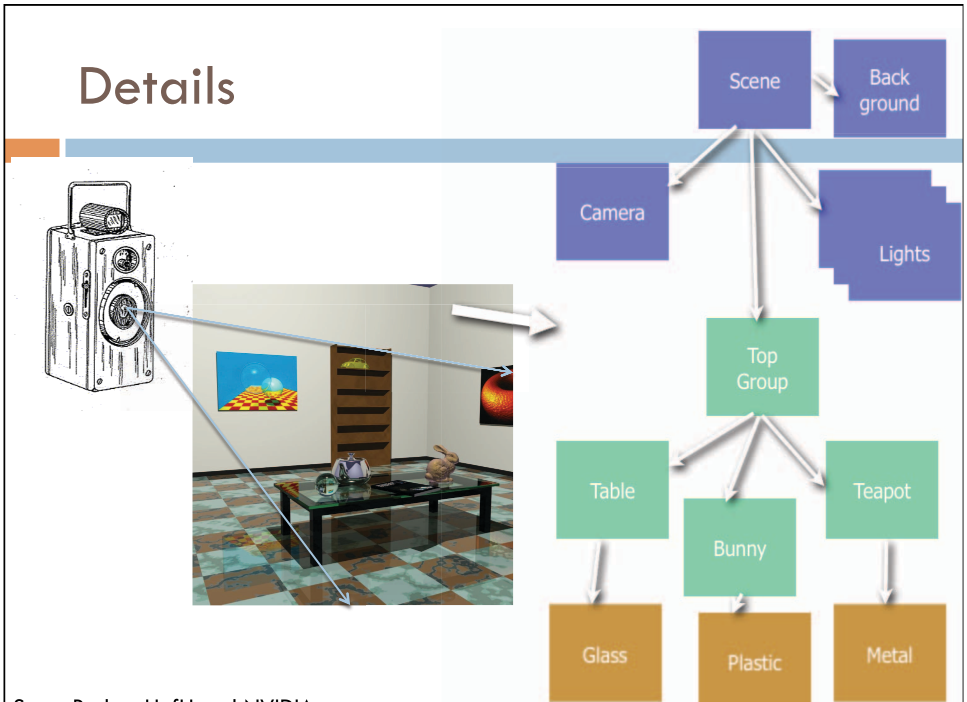


Major Components of a Ray Tracer

- Camera (Pixels to Rays)
- Objects (Rays to intersection info)
- Materials (Intersection info and light to color)
- Lights
- Background (Rays to Color)

- All together: a Scene

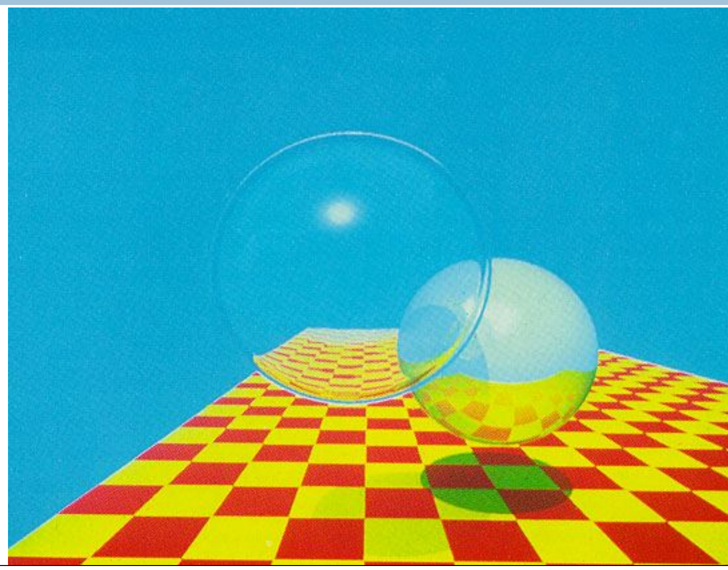
Details



Steve Parker, UofU and NVIDIA

Optical Effects

Turner Whitted
1980



Turner Whitted

An Improved Illumination Model for Shaded Display

Turner Whitted
Bell Laboratories
Holmdel, New Jersey

$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j) + k_s \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j')^n, \quad (1)$$

where

I = the reflected intensity,
I_a = reflection due to ambient light,
k_d = diffuse reflection constant,
$\bar{N}$ = unit surface normal,
$\bar{L}_j$ = the vector in the direction of the jth light source,
k_s = the specular reflection coefficient,
$\bar{L}_j'$ = the vector in the direction halfway between the viewer and the jth light source,
n = an exponent that depends on the glossiness of the surface.

Phong Model

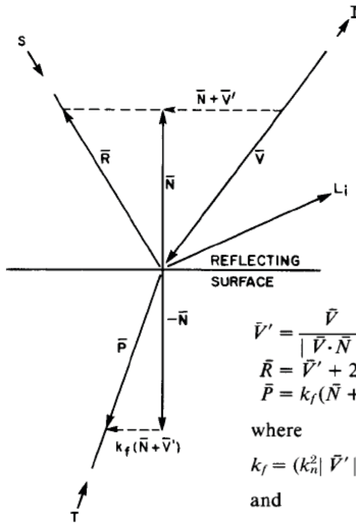
$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j) + k_s S + k_t T, \quad (2)$$

where

S = the intensity of light incident from the $\bar{R}$ direction,
k_t = the transmission coefficient,
T = the intensity of light from the $\bar{P}$ direction.

Improved Model

Turner Whitted



$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j) + k_s S + k_t T, \qquad (2)$$

where

S = the intensity of light incident from the $\bar{R}$ direction,
k_t = the transmission coefficient,
T = the intensity of light from the $\bar{P}$ direction.

Improved Model

$$\bar{V}' = \frac{\bar{V}}{|\bar{V} \cdot \bar{N}|},$$
$$\bar{R} = \bar{V}' + 2\bar{N},$$
$$\bar{P} = k_f(\bar{N} + \bar{V}') - \bar{N},$$

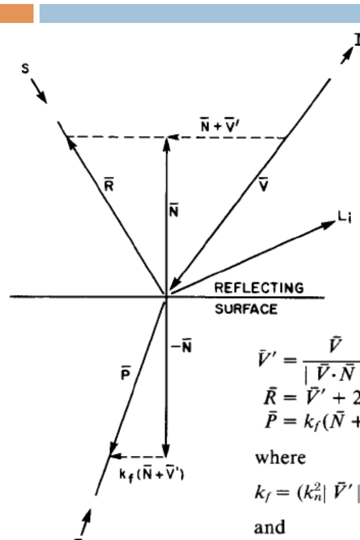
where

$$k_f = (k_n^2 |\bar{V}'|^2 - |\bar{V}' + \bar{N}|^2)^{-1/2},$$

and

k_n = the index of refraction.

Turner Whitted



$$\bar{V}' = \frac{\bar{V}}{|\bar{V} \cdot \bar{N}|},$$
$$\bar{R} = \bar{V}' + 2\bar{N},$$
$$\bar{P} = k_f(\bar{N} + \bar{V}') - \bar{N},$$

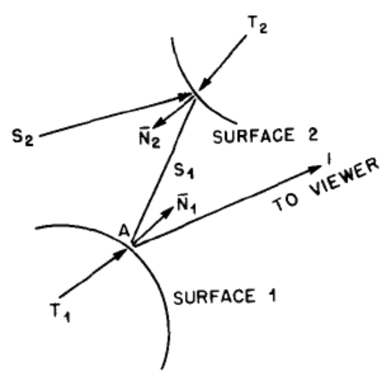
where

$$k_f = (k_n^2 |\bar{V}'|^2 - |\bar{V}' + \bar{N}|^2)^{-1/2},$$

and

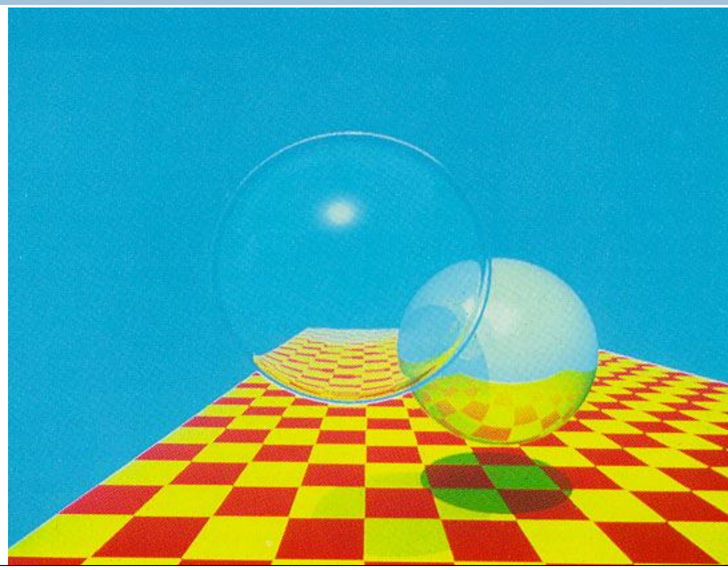
k_n = the index of refraction.

Fig. 2.



Optical Effects

Turner Whitted
1980



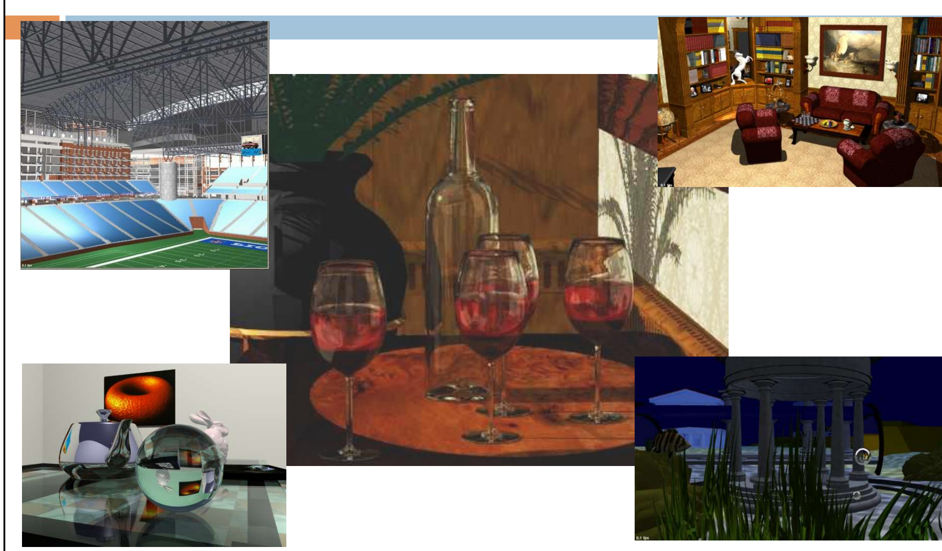
Optical Effects

Steve Parker, UofU and NVIDIA

Volume Rendering

Steve Parker, UofU and NVIDIA

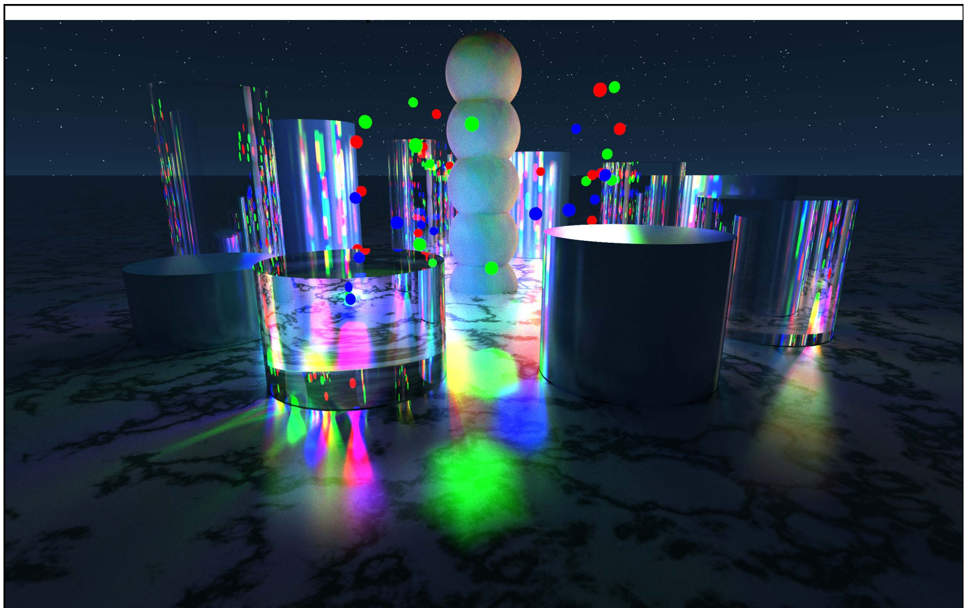
Surface model ray traced images



Steve Parker, UofU and NVIDIA

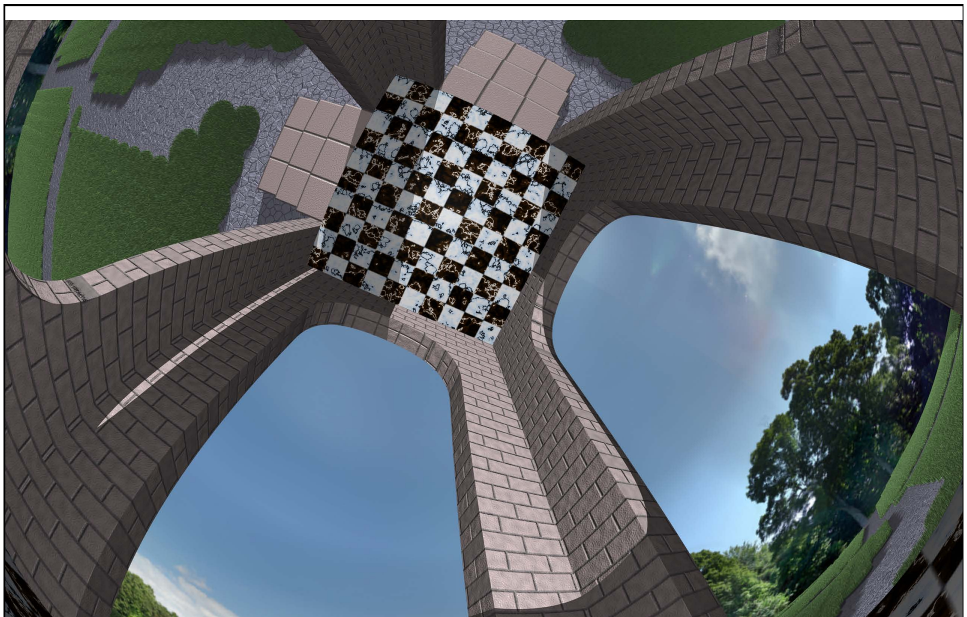
Student images from CS6620

Pegoraro

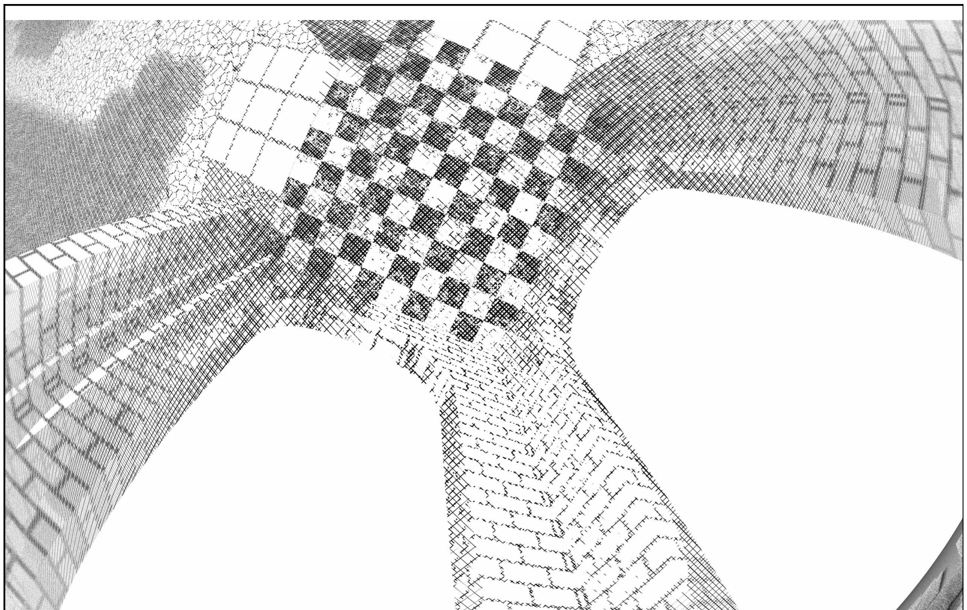


CS6620 Spring 08

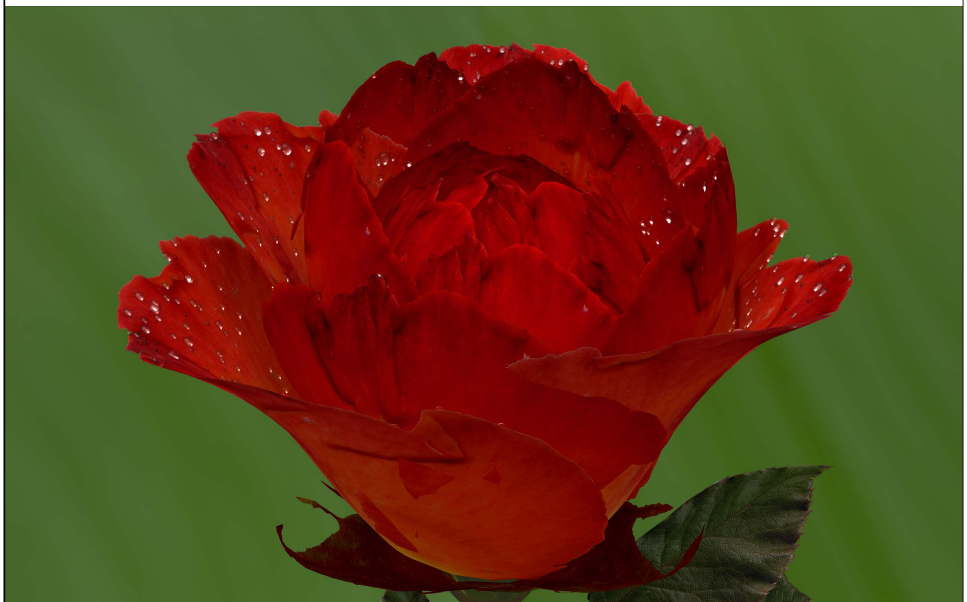
Gallup



Kensler

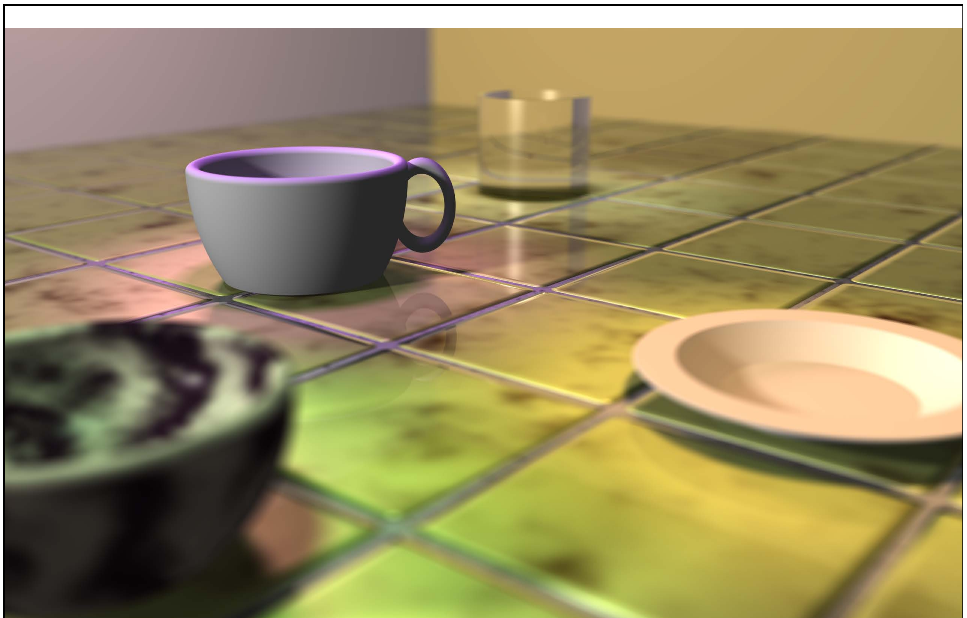


Kensler variant B



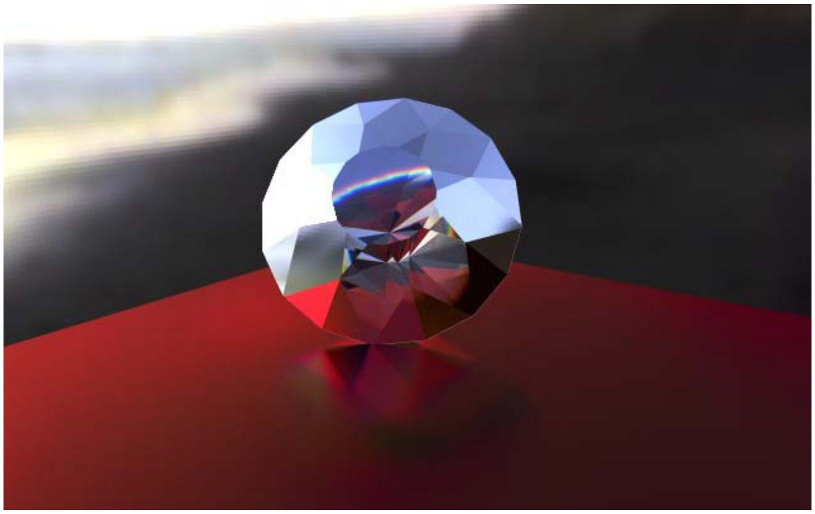
CS6620 Spring 08

Bavoil

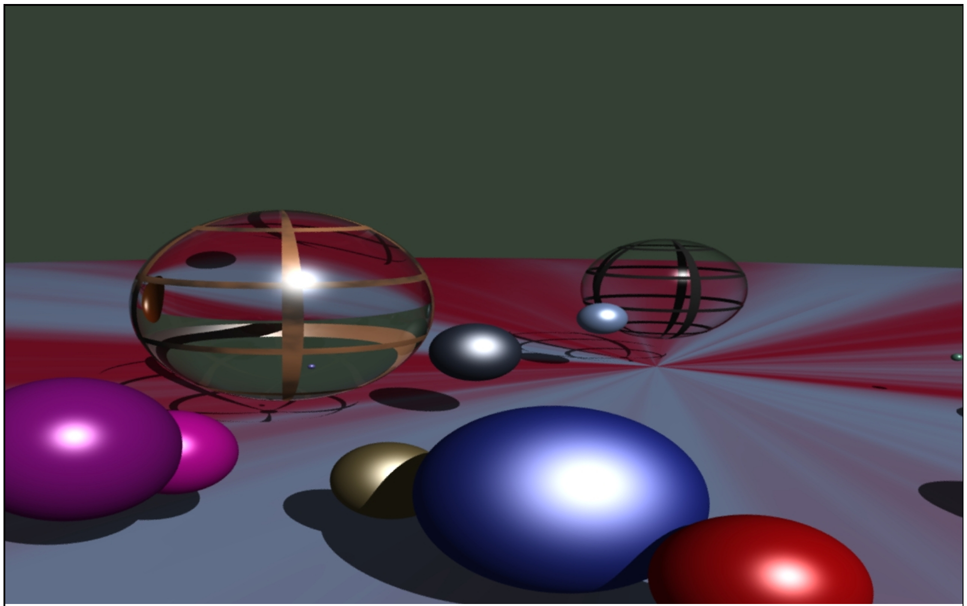


CS6620 Spring 08

Steffen



CS6620 Spring 08



CS6620 Spring 08

Archuletta

Luitjens

Martin

CS6620 Spring 08

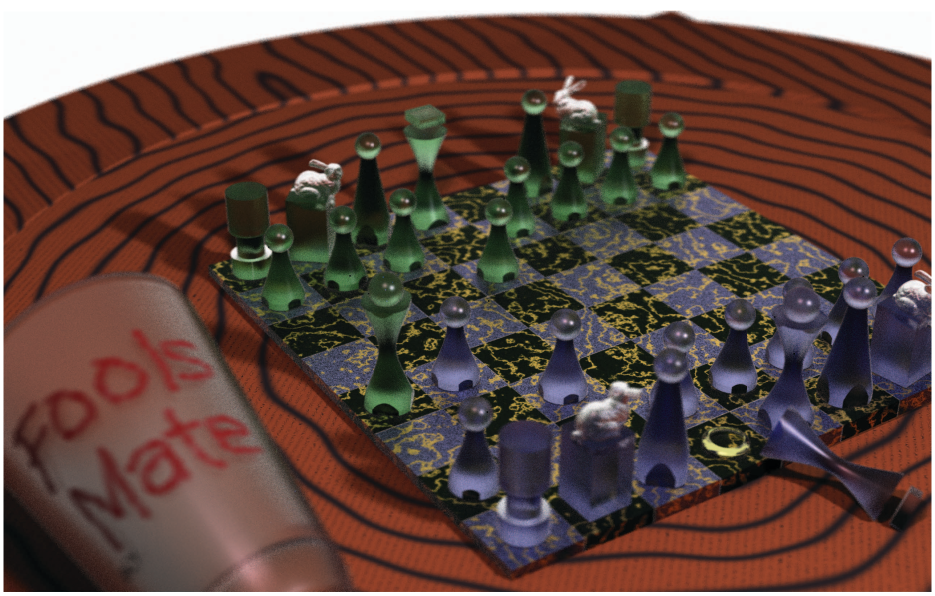
Ownby

CS6620 Spring 08

Schott

CS6620 Spring 08

Baird

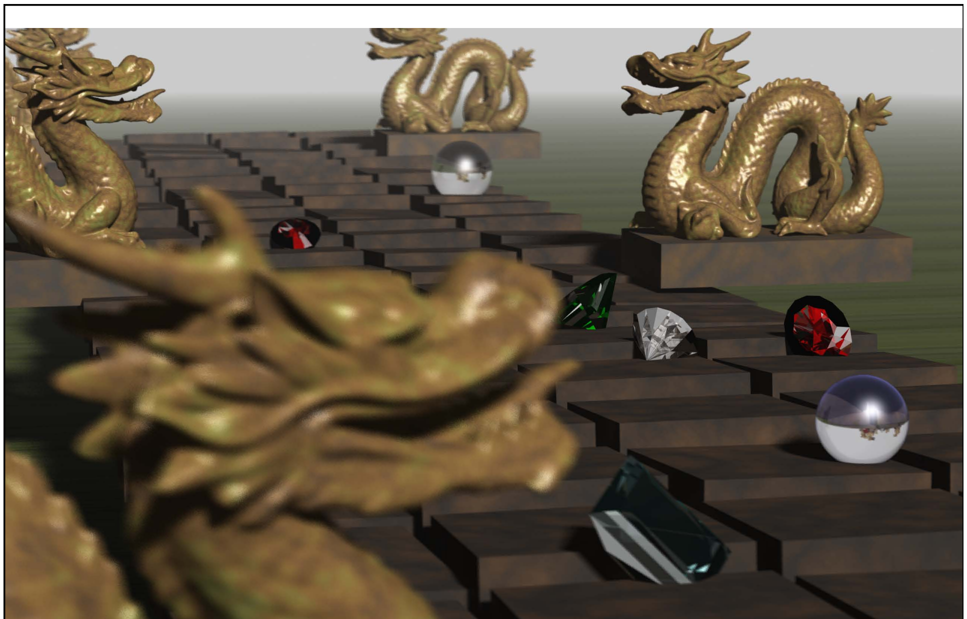


CS6620 Spring 08

Young

CS6620 Spring 08

Stratton



CS6620 Spring 08

Hawkins

Ray Tracing is complex?

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,colour;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,colour;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;colour=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)colour=vcomb(e
,l->colour,colour);U=s->colour;colour.x*=U.x;colour.y*=U.y;colour.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,
colour,vcomb(s->kl,U,black))));}main(){puts("P3\n32 32\n255");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!*/
```

Paul Heckbert's complete ray tracer on the back of his business card (c1989)

Does Whitted-style recursive ray tracing with reflections, refraction, two lights…

Ray Tracing is complex?

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,colour;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<t
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,I
amb;colour=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)=
,l->colour,colour);U=s->colour;colour.x*=U.x;colour.y*=U.y;co
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)
colour,vcomb(s->kl,U,black))));}main(){puts("P3\n32 32\n255")
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.591559026
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);},
```

Paul Heckbert's complete ray tracer on the back of his business card (c1989)

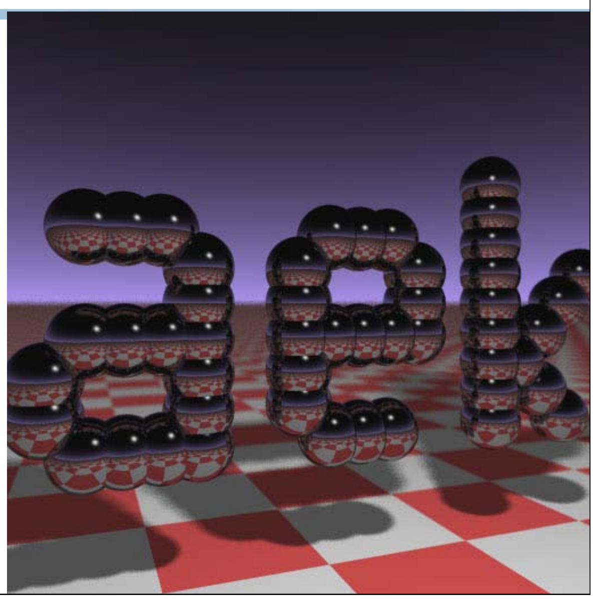
Does Whitted-style recursive ray tracing with reflections, refraction, two lights…

Andrew Kensler's business-card C++ RT

```
#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);}}
```

Andrew Kensler's business-card C++ RT

```
#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);}}
```

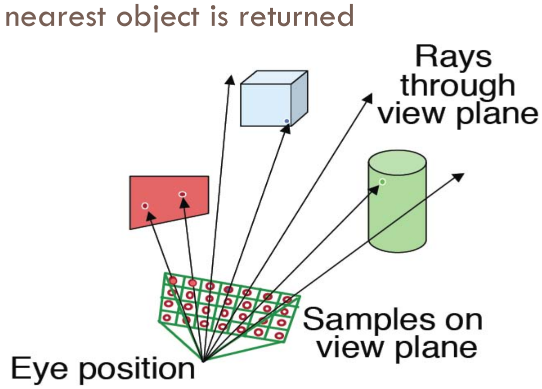



A Hierarchy of Ray Tracers

1. Ray casting
2. Ray casting with shadows
3. Whitted-style recursive ray tracing
4. Cook-style distribution ray tracing
5. Path tracing for indirect illumination (global illumination)
6. … even more advanced techniques…

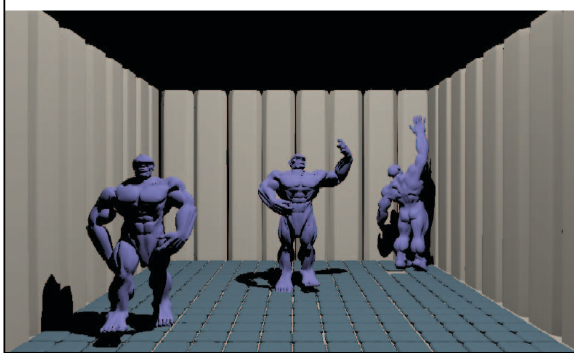
1: Ray Casting

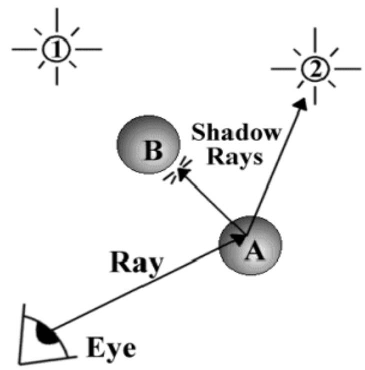
- A 3D line query to determine visibility
  - Rays are cast from the eye point through each pixel into the scene
  - Intersection point of nearest object is returned



2: Ray Casting with Shadows

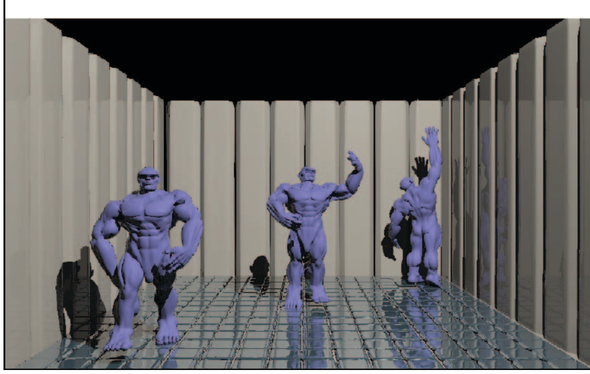
- At each intersection point, cast another ray in the direction of the light source
  - Checks whether the point is in shadow

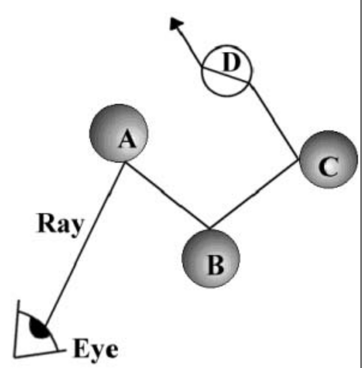




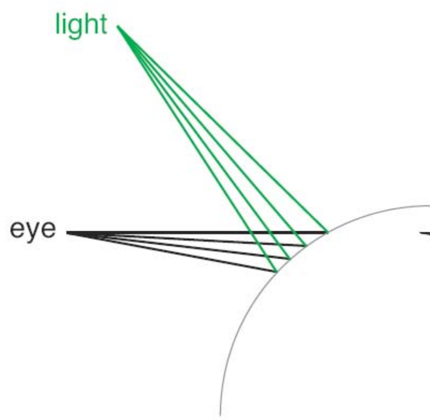
3: Whitted-Style Ray Tracing

- Recursively cast rays to account for reflections and refractions

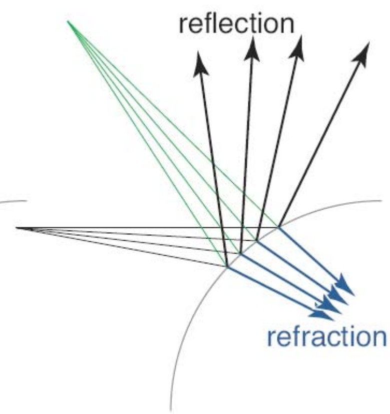




3: Whitted-Style Ray Tracing

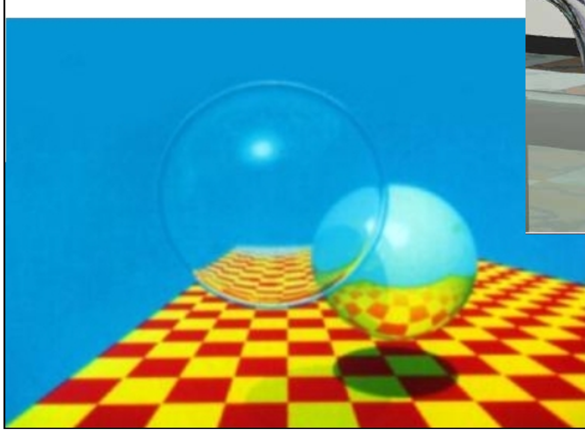


Ray casting with shadows



Whitted-style ray tracing

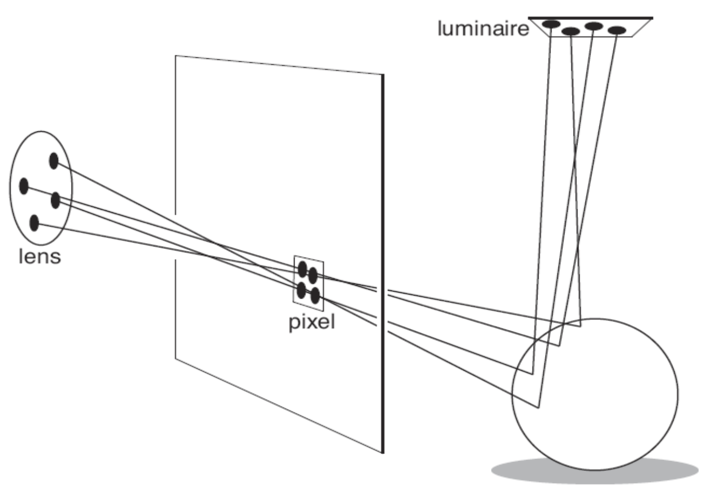
Classic Whitted Examples



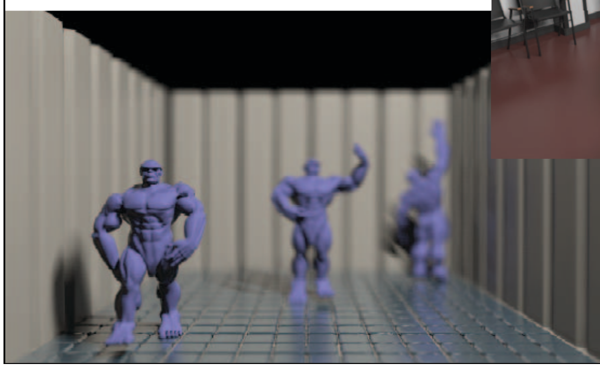
4: Distribution Ray Tracing

- AKA Cook-Style Ray Tracing
  - Rays can be cast through a lens with area (i.e. not just a pinhole)
    - Depth of field
  - secondary rays directions can be perturbed
    - Glossy reflections
  - Shadow rays can be aimed at area light sources
    - Soft shadows
  - Can also add time to the ray
    - Motion blur

4: Distribution Ray Tracing



4: Distribution Ray Tracing

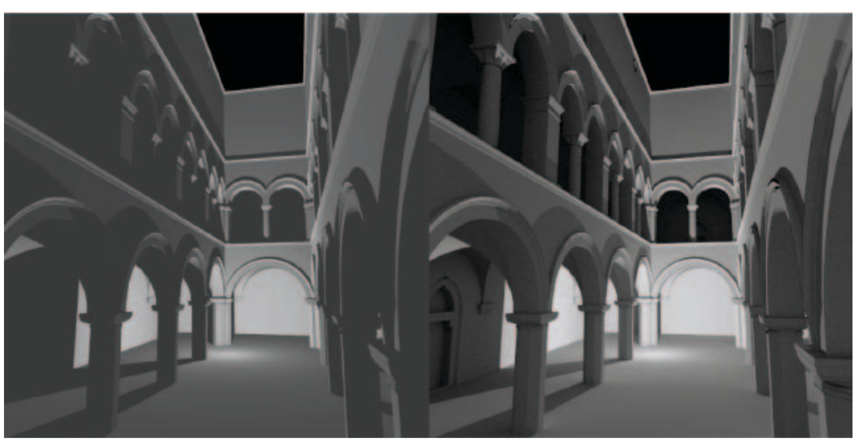


4: Distribution Ray Tracing

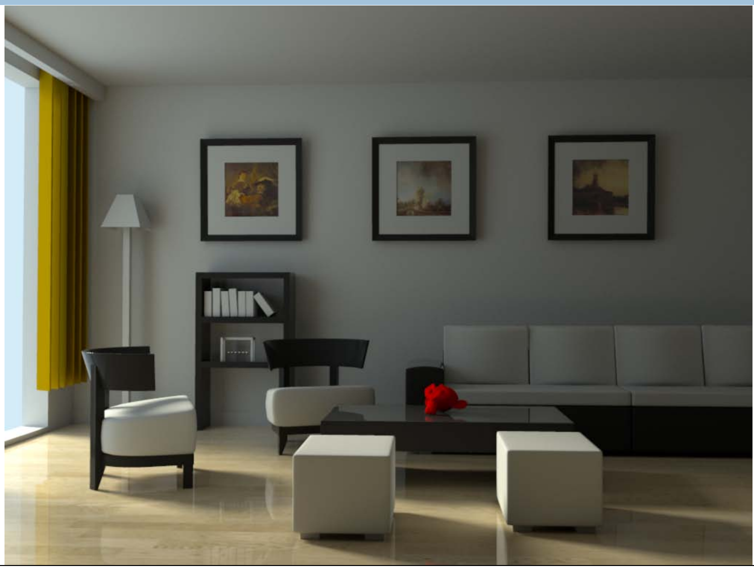
5: Path Tracing

- At each intersection point, cast a ray in a random direction to see if any light comes from there
  - With enough oversampling, this results in solving the "rendering equation"
  - Fills in the "ambient" shadowed spaces with indirect lighting

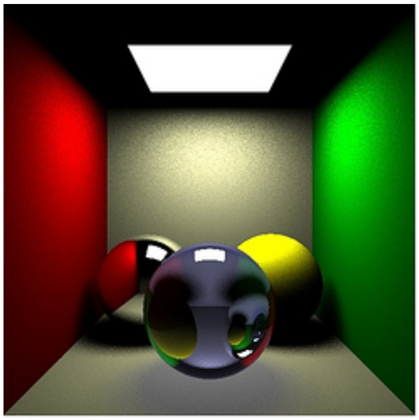
5: Path Tracing



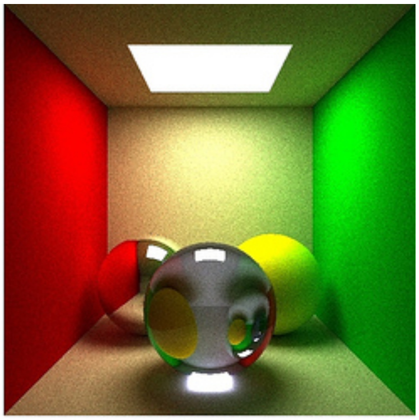
5: Path Tracing



5: Path Tracing



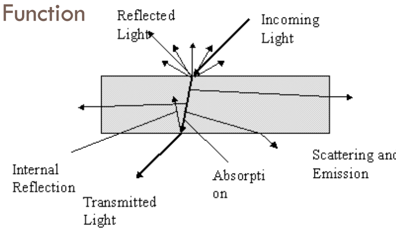
Whitted ray tracing



Path Tracing

Lots more to it…

- But this hierarchy helps me keep things straight
  - Ambient occlusion, ray bundles, beam tracing, photon mapping, metropolis light transport, etc. etc. etc.
- Material properties involve other huge set of issues that can impact realism
  - BRDF: Bidirectional Reflectance Distribution Function
  - BSDF: Bidirectional Scattering Distribution Function
  - BTDF: Bidirectional Transmission Distribution Function
  - BSSRDF: Bidirectional Scattering Surface Reflectance Distribution Function



TRaX

- If you could build a GPU that was customized for ray tracing, what would it look like?
  - Probably have lots of floating point units
  - NVIDA/ATI GPUs organize them as wide SIMD
    - For example, 32 threads in a "warp"
    - Great if all 32 threads truly do the exact same thing
    - Not so great if they branch…
  - TRaX takes a more MIMD/SPMD approach
    - Let the multiple threads each have their own PC
    - Letting the threads be out of sync has benefits…

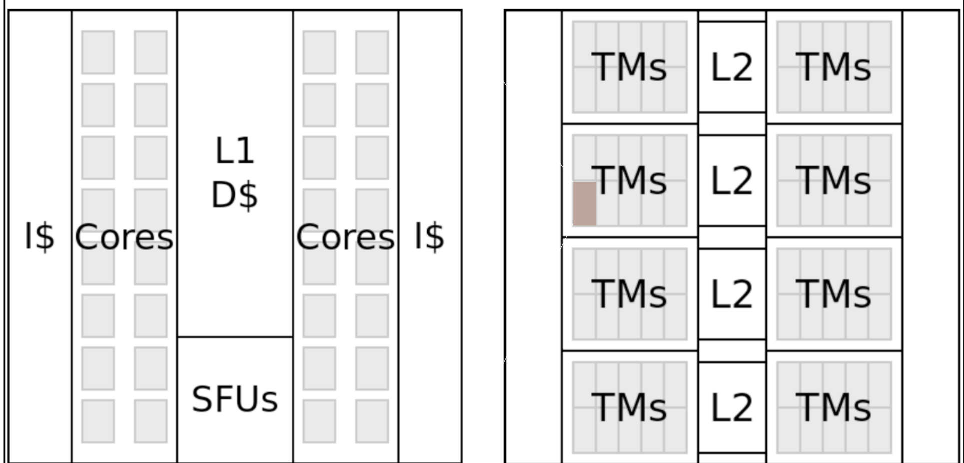
TRaX

- Ray tracing is by nature divergent in control flow
- Consider lots of light-weight MIMD threads capable of handling divergence
  - Designed from ground up specifically for ray tracing
- But what about area overhead for MIMD?

Ray Tracing Domain Features

- Ray tracing is “embarrassingly parallel”
  - Minimal communication and synchronization required
- All memory writes are to framebuffer only
  - We can enforce write-around policy to keep caches clean
  - Use local scratchpad memory for temporary variables
- Small program size makes for small fast icaches
- Threads all at different places in program (out of sync)
  - We can share various resources since all threads won’t be using them at the same time

TRaX Architecture



Basic tile
32 core thread multiprocessor (TM)

Chip

TRaX Software Model

- Write a single-threaded ray tracer
  - Copy this code to all thread processors
- Now make each thread use atomic increment to help it decide which rays are its responsibility
  - Let 'em loose on the scene

How well does it work?

L1 Size	L1 Banks	L2 Size	L2 Banks	L1 Hitrate	L2 Hitrate	Per Cache Bandwidth (GB/s) L1– >reg	L2– >L1	main– >L2	Thread Issue	Area (mm^2)	MRPS	MRPS/ mm^2
32KB	4	256KB	16	93%	75%	42	56	13	70%	147	322	2.2
32KB	4	512KB	16	93%	81%	43	57	10	71%	156	325	2.1
32KB	8	256KB	16	93%	75%	43	57	14	72%	159	330	2.1
32KB	8	512KB	16	93%	81%	43	57	10	72%	168	335	2.0
64KB	4	512KB	16	95%	79%	45	43	10	76%	175	341	1.9

- 2 int add, 8 FP mul and FP add, 1 invsqrt, and 2 16-banked instruction caches per TM
- 256KB, 16-bank L2 data cache x 4
  - Total off-chip bandwidth: 52 GB/sec
- 20 TMs per L2 (80 total)
- Total of 2560 cores

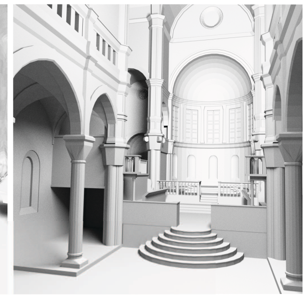
Comparison

- We compare our architecture against the best known wide-SIMD GPU ray tracer
  - Timo Aila (NVIDIA) et al, HPG09
  - Running on NVIDIA GTX 285
- Used same scenes and rendering techniques (shaders)
- Compare performance/area and overall performance

Benchmark Scenes


Conference Room
282K triangles


Fairy Forest
174K triangles


Sibenik Cathedral
80K triangles

•Primary rays only: ~1M rays per frame

•Shading (w/secondary rays): ~34M rays per frame

Results

		Conference (282k triangles)		Fairy (174k triangles)		Sibenik (80k triangles)	
MIMD	Ray Type	MIMD Issue Rate	MIMD MRPS	MIMD Issue Rate	MIMD MRPS	MIMD Issue Rate	MIMD MRPS
$175mm^2$	Primary	77%	387	73%	421	79%	285
	Secondary	67%	355	70%	402	46%	131
SIMD	Ray Type	GTX SIMD eff.	GTX MRPS	GTX SIMD eff.	GTX MRPS	GTX SIMD eff.	GTX MRPS
GTX285	Primary	74%	142	76%	75	77%	117
	Secondary	46%	61	46%	41	49%	47

MIMD/SPMD total area: 175mm^2

GTX285 total area: ~300mm^2

Both areas estimated at 65nm process

Resource Area

	GTX285 SM (8 cores)	MIMD TM (32 cores)
Registers	16384	4096
FPAdds	8	8
FPMuls	8	8
INTAdds	8	32
INTMuls	8	2
Spec op	2	1
Register Area (mm^2)	2.43	0.61
Compute Area (mm^2)	0.43	0.26

SM = streaming multiprocessor, NVIDIA's analogue to our TM

71

Analysis

- SPMD and resource sharing benefit from each other
  - threads get out of sync, resource requests become evenly staggered
  - which results in high performance, small area
- Small multi-banked icaches diminish area requirement of SPMD instruction fetch
  - Without constraint of synchronized threads

72

Comparison Conclusion

- Wide SIMD and general purpose multi-core CPUs seem over-provisioned for ray tracing
- lightweight SPMD architecture with shared resources out-performs a highly tuned ray tracer on highly evolved GPU hardware on realistic benchmark scenes

73

Back to: What's the Plan?

- Start with everyone writing a ray tracer
  - Using our llvm-based compiler
  - Get it running on your machine
  - Then on our TRaX simulator
- Then propose projects to write a more advanced RT
  - Benchmark on TRaX simulator
  - Could even result in papers?
- *Next time: Josef will start talking about RT infrastructure (mathematics, coding, etc.)*