Ray tracing

CS425: Computer Graphics I

Fabio Miranda

https://fmiranda.me

Overview

- Basic of ray tracing
- Object intersection
- Building a ray tracer
- Nvidia OptiX
- RT cores

Ray tracing

- How to render realistic images?
  - Color
  - Materials
  - Rendering equation
- Alternative to rasterization: much more realistic images (at a cost!)

Ray tracing

- Reverse of reality:
  - Shoot rays through image plane.
  - See what ray hits.
  - Secondary rays:
    - Reflections
    - Shadows

Ray tracing

Camera

Geometry

Materials

Lights

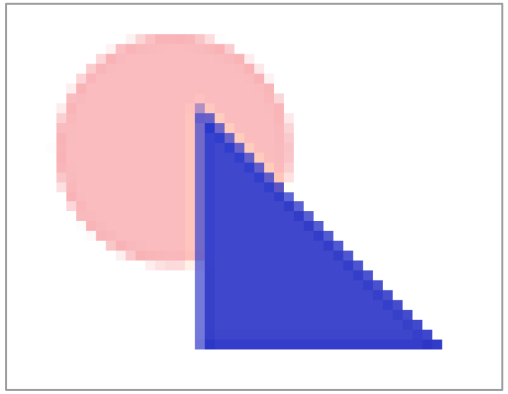
Scene

Ray tracer

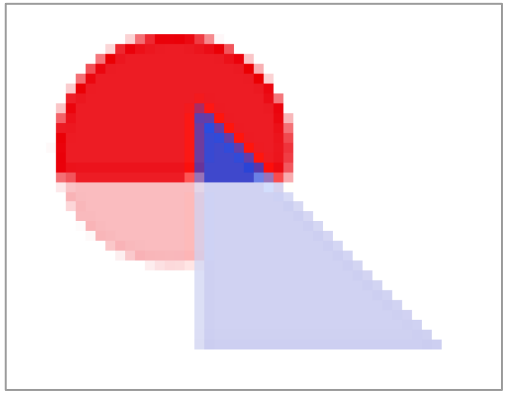
Ray tracing vs. rasterization

- Ray tracing and rasterization generate images in different ways.
- <u>Basic</u> difference: order in which samples are processed.

Rasterization



```
for each primitive:
    for each pixel:
        1. determine coverage
        2. evaluate color
```



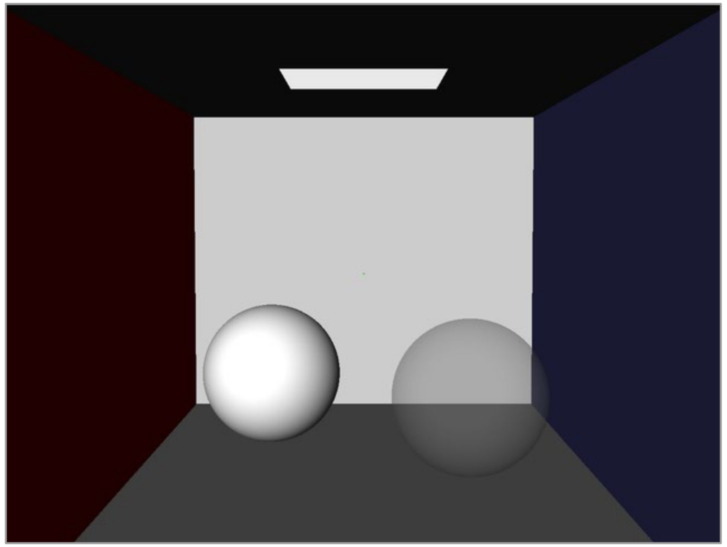
Ray tracing

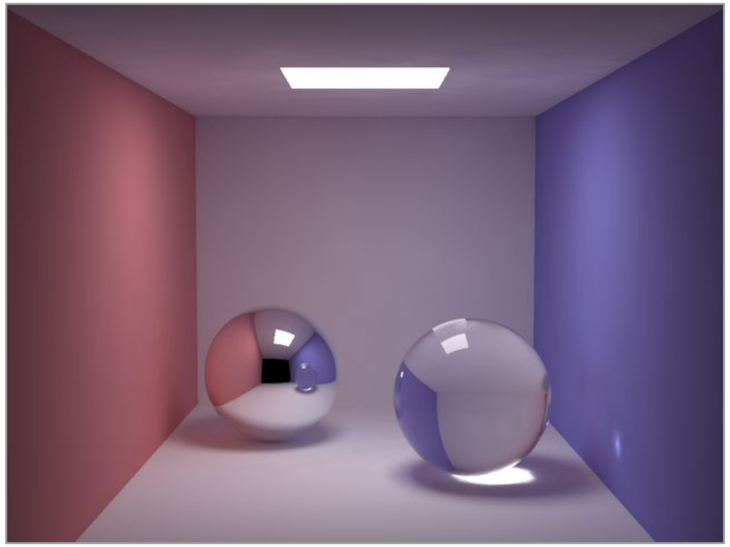
```
for each pixel:
    for each primitive:
        1. determine coverage
        2. evaluate color
```

Ray tracing vs. rasterization

- Illumination model:
  - Rasterization: local illumination – one primitive at a time.
  - Ray tracing: global illumination – one ray at a time (“easier” to determine global illumination effects).

Rasterization





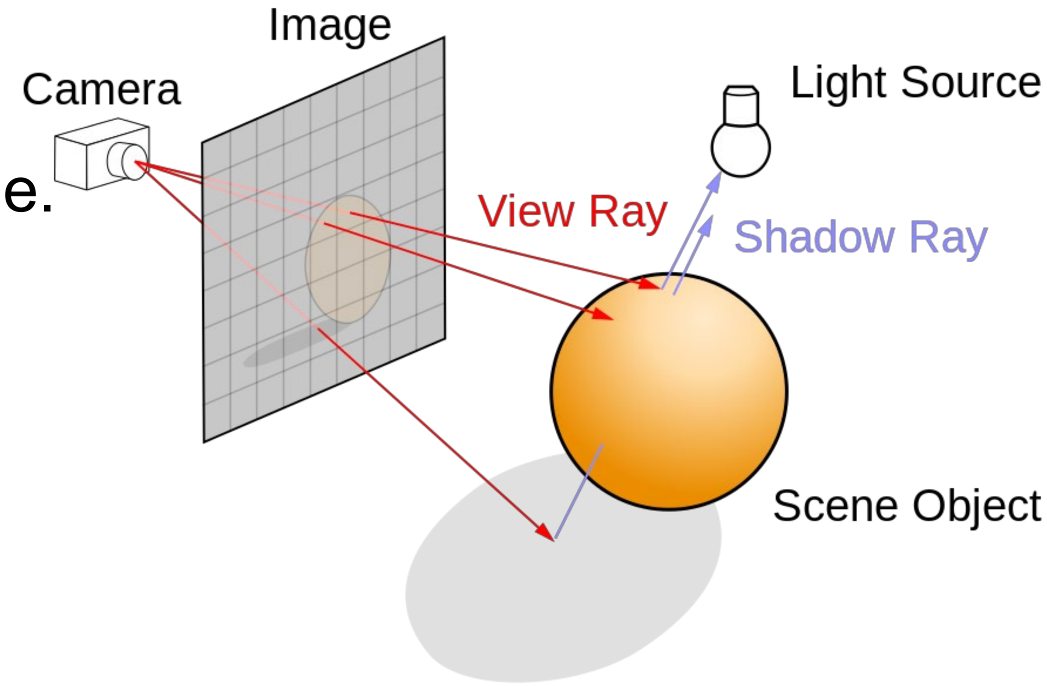
Ray tracing

Basic ray tracing

- Given a point source, follow rays of light as they traverse the scene.
  - Rays shot from the eye through the pixel grid into the scene.
  - Each ray: closest object is found.
    - Shoot ray from intersection point to each light and find if it intersects with any object (if so, point in shadow).
- Account for reflection and transmission.

Basic ray tracing

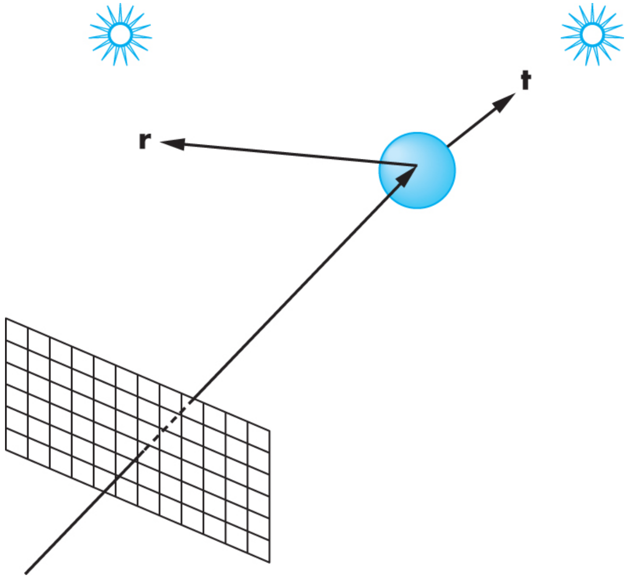
1. Create ray (one per pixel).
2. Intersect ray with objects in the scene.
3. Shade (compute color of the pixel)

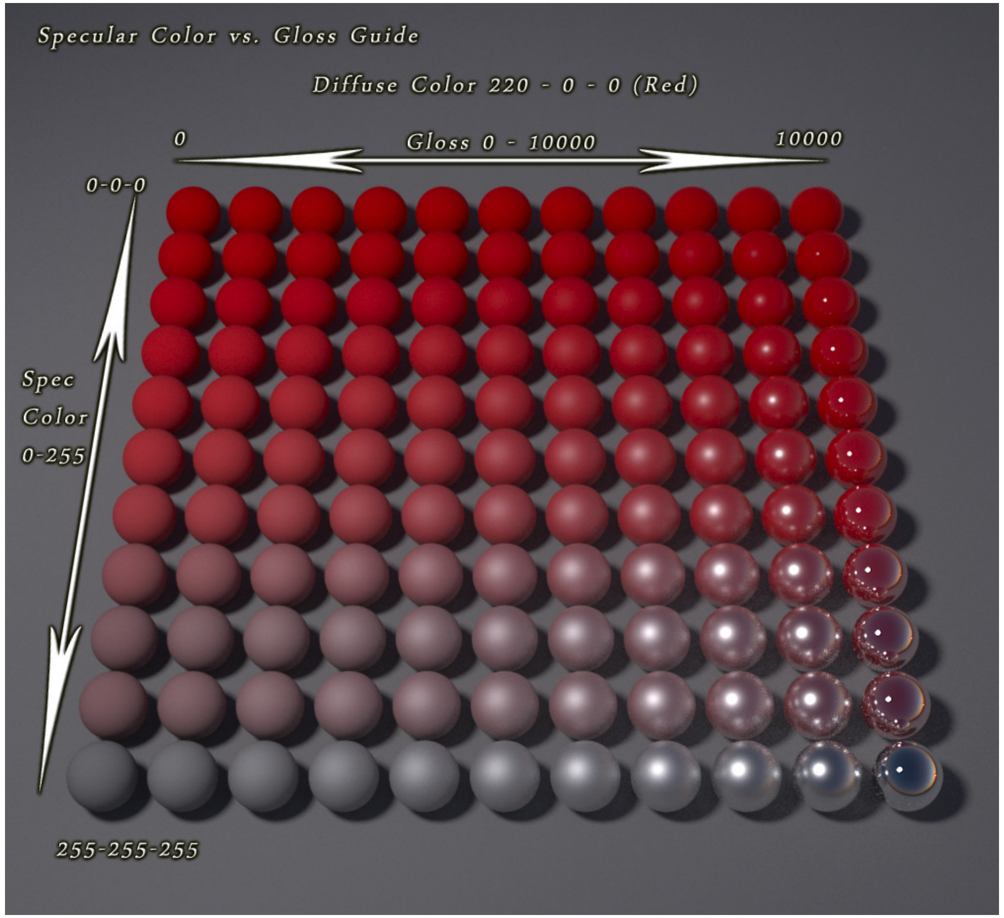


By Henrik - Own work, GFDL,
https://commons.wikimedia.org/w/index.php?curid=3869326

Basic ray tracing

- Intersection ray-object will determine *secondary* rays, according to object material.



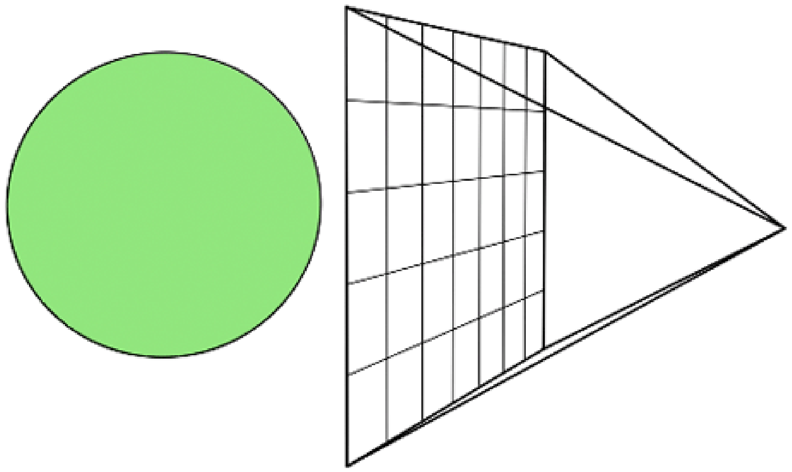


Constructing a ray

- Ray equation:

$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$

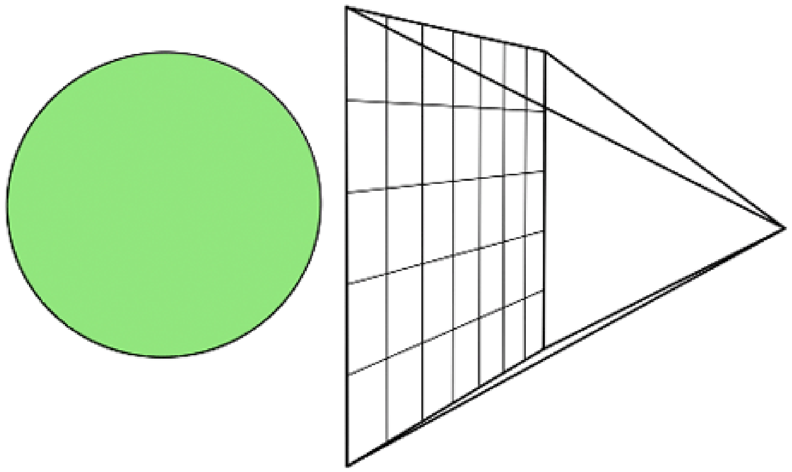
- How to construct it?
- We have:
  - Camera position
  - Camera direction
  - Up vector
  - Viewport (width, height)
- We want:
  - Ray described by camera position and pixel center.



scratchapixel.com

Constructing a ray

- Ray equation:
$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$
- How to construct it?
- We have:
  - Camera position
  - Camera direction
  - Up vector
  - Viewport (width, height)
- We want:
  - Ray described by camera position and pixel center.



scratchapixelc.com

Constructing a ray

- Ray equation:

$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$

- How to construct it?

1. Transform pixel position to camera space:

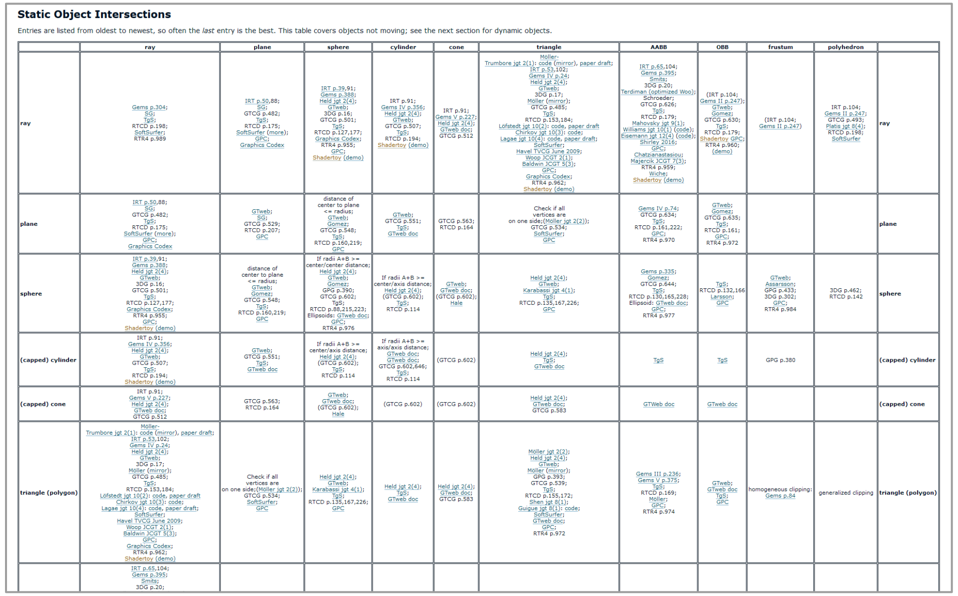
$$\mathbf{e} = ((2pixelScreen_x - 1) * AspectRatio * \tan\left(\frac{fov}{2}\right), (1 - 2pixelScreen_y) * \tan\left(\frac{fov}{2}\right), -1)$$

2. Ray direction:

$$\mathbf{d} = ||\mathbf{e} - \mathbf{eye}||$$

Intersections

- Intersections are expensive operations.
- Great resource for operations:
  - http://www.realtimerendering.com/intersections.html
  - More than 100 types of intersections.
- We will focus on two:
  - Ray-sphere intersection.
  - Ray-triangle intersection (we can approximate complex surfaces with triangles).



UIC COMPUTER SCIENCE

Ray-sphere intersection

- Ray:

$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$

- Sphere (radius R and center c):

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2$$
$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0$$

- Intersection:

$$f(\mathbf{p}(t)) = 0$$

Ray-sphere intersection

- Plugging ray equation into sphere equation:

$$(\mathbf{e} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{e} + t\mathbf{d} - \mathbf{c}) - R^2 = 0$$

$$t\mathbf{d} \cdot t\mathbf{d} + 2t\mathbf{d}(\mathbf{e} - \mathbf{c}) + (\mathbf{e} - \mathbf{c}) \cdot (\boldsymbol{e} - \boldsymbol{c}) - R^2 = 0$$

$$(\mathbf{d} \cdot \mathbf{d})t^2 + 2\mathbf{d} \cdot (\mathbf{e} - \mathbf{c})t + (\mathbf{e} - \mathbf{c}) \cdot (\mathbf{e} - \mathbf{c}) - R^2 = 0$$

$$At^2 + Bt + C = 0$$

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \begin{cases} B^2 - 4AC < 0: no\ intersection \\ B^2 - 4AC = 0: one\ intersection \\ B^2 - 4AC > 0: two\ intersections \end{cases}$$

Ray-triangle intersection

- Ray:

$$\mathbf{p}(t) = \mathbf{e} + t\mathbf{d}$$

- Triangle (vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$):

$$\mathbf{f}(u, v) = \mathbf{a} + u(\mathbf{b} - \mathbf{a}) + v(\mathbf{c} - \mathbf{a})$$

- Intersection:

$$\mathbf{f}(u, v) = \mathbf{p}(t)$$
$$t > 0$$
$$0 \leq u, v$$
$$u + v \leq 1$$

Ray-triangle intersection

- Plugging ray equation into triangle:

$$\mathbf{a} + u(\mathbf{b} - \mathbf{a}) + v(\mathbf{c} - \mathbf{a}) = \mathbf{e} + t\mathbf{d}$$

$$\mathrm{u}(\mathbf{b} - \mathbf{a}) + v(\mathbf{c} - \mathbf{a}) - t\mathbf{d} = \mathbf{e} - \mathbf{a}$$

$$\mathbf{Ax} = \mathbf{b}$$

$$\begin{bmatrix} (\mathbf{b} - \mathbf{a})_{\mathrm{x}} & (\mathbf{c} - \mathbf{a})_{\mathrm{x}} & -d_{\mathrm{x}} \\ (\mathbf{b} - \mathbf{a})_{\mathrm{y}} & (\mathbf{c} - \mathbf{a})_{\mathrm{y}} & -d_{\mathrm{y}} \\ (\mathbf{b} - \mathbf{a})_{\mathrm{z}} & (\mathbf{c} - \mathbf{a})_{\mathrm{z}} & -d_{\mathrm{z}} \end{bmatrix} \begin{bmatrix} u \\ v \\ t \end{bmatrix} = \begin{bmatrix} (\mathbf{e} - \mathbf{a})_{\mathrm{x}} \\ (\mathbf{e} - \mathbf{a})_{\mathrm{x}} \\ (\mathbf{e} - \mathbf{a})_{\mathrm{x}} \end{bmatrix}$$

- How to solve the system?
  - Cramer's rule:

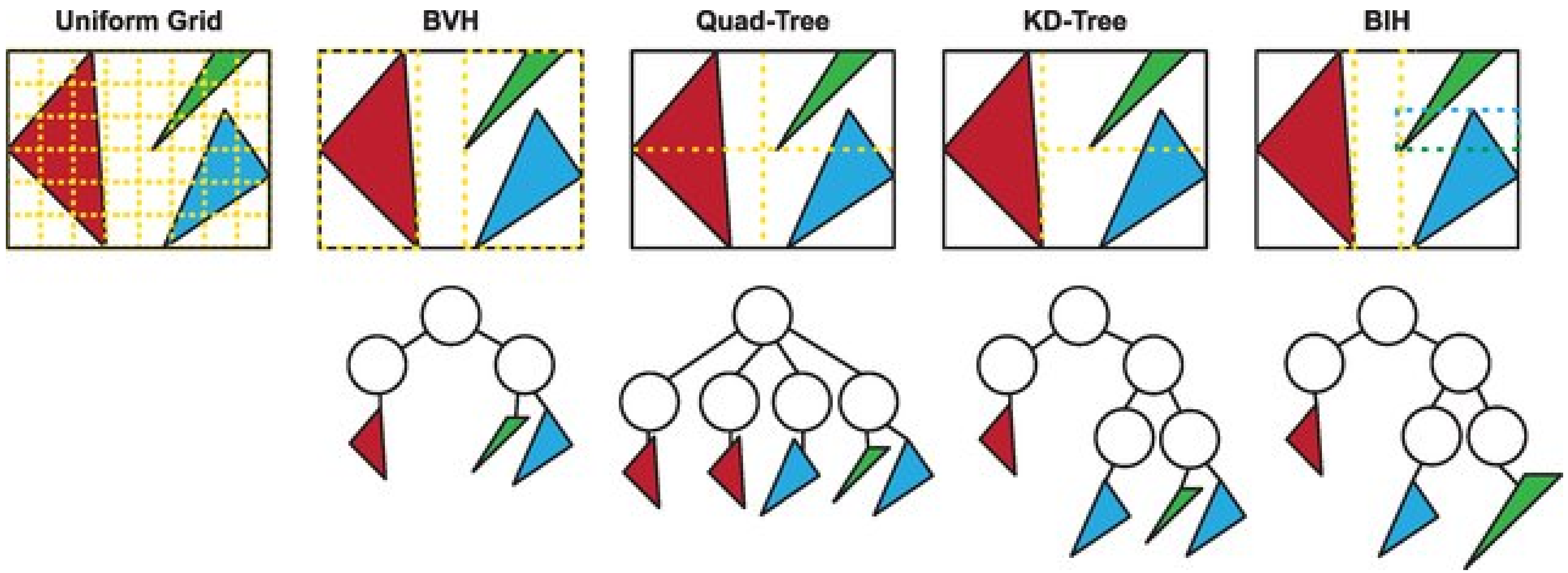
    $x_i = \frac{\det(A_i)}{\det(A)}$, where $\mathbf{A}_i$is the matrix $\mathbf{A}$ replacing the i-th column with vector $\mathbf{b}$

Multiple objects

- We must intersect each ray with **all** objects in the scene.
  - Complexity: O(n), where n is the number of primitives.
- To speed up computation, we can make use of spatial data structures to prune the number of collisions that we need to check.

Space-partitioning data structures

- Uniform grid
- Bounding volume hierarchy (BHV)
- Quadtree / Octree
- KD-Tree

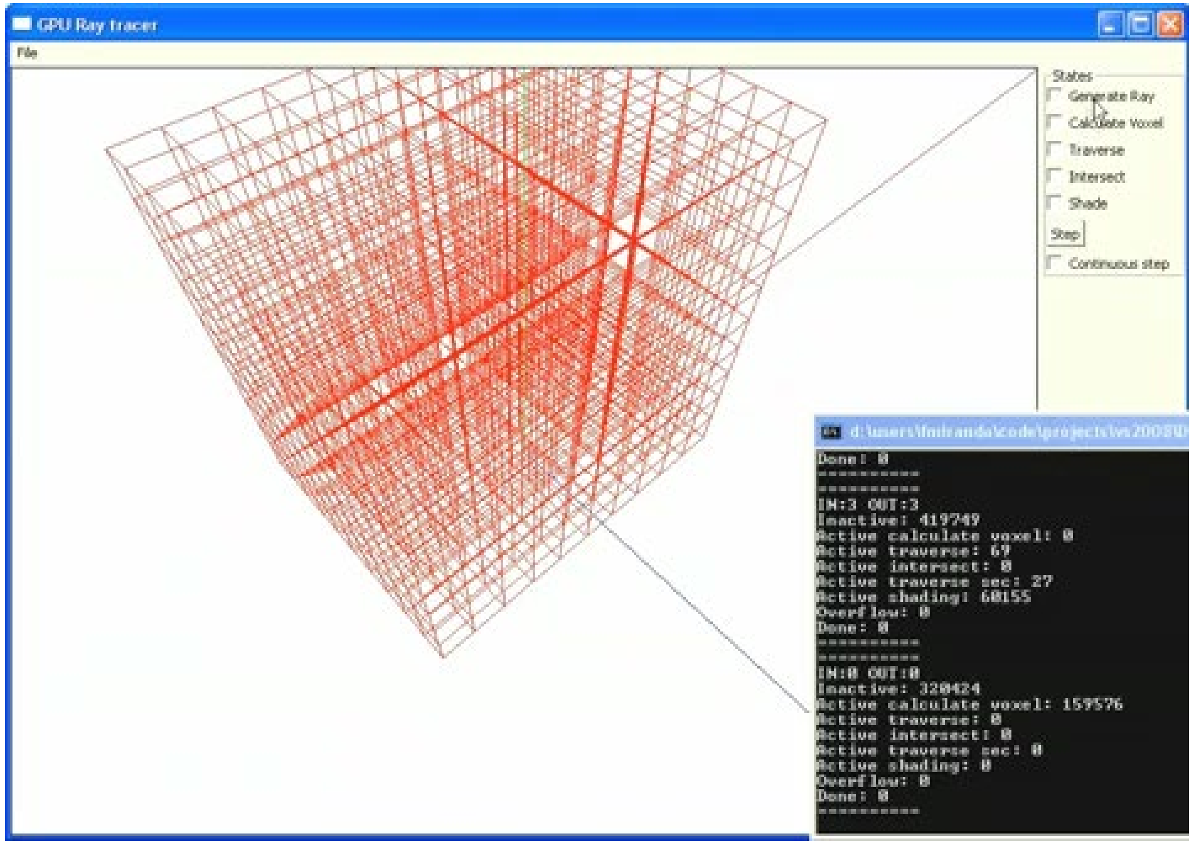


Review and Comparative Study of Ray Traversal Algorithms on a Modern GPU Architecture, Santos et al.

UIC COMPUTER SCIENCE

Space-partitioning data structures

- Uniform grid
- Bounding volume hierarchy (BHV)
- Quadtree / Octree
- KD-Tree



UIC COMPUTER SCIENCE

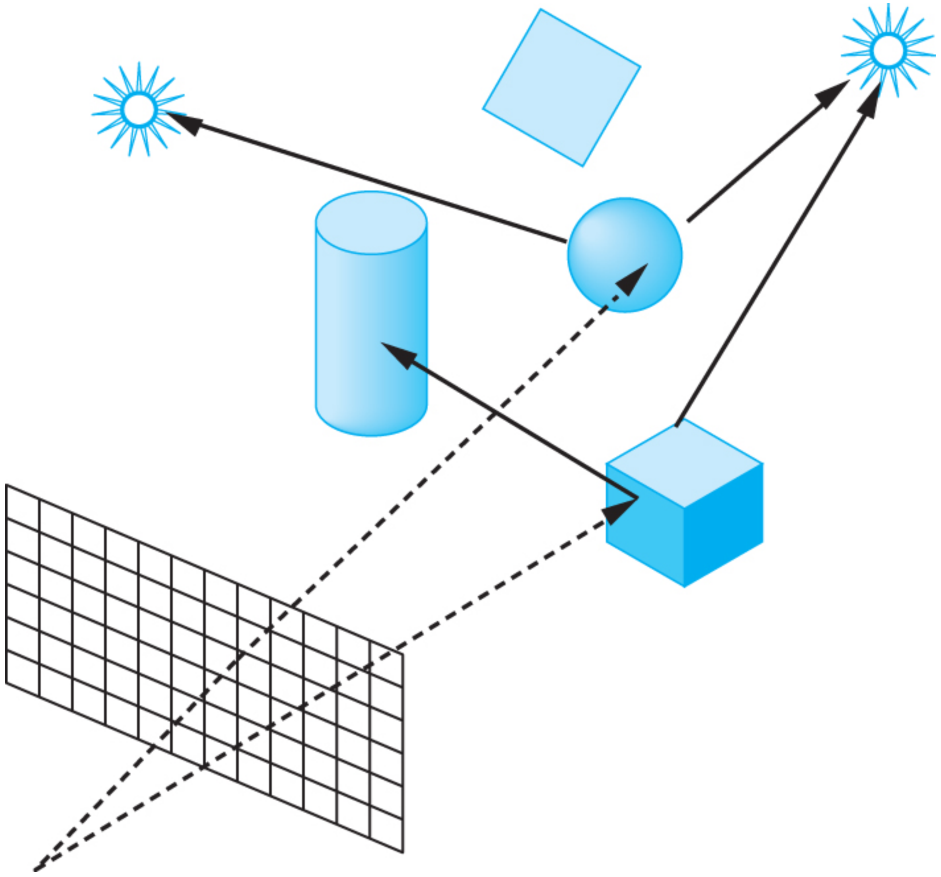
Recursive ray tracing

- Major steps of a ray tracer:

1. Determine intersection ray-objects.
2. Get intersected object material.
3. Based on material:
    1. Shade
    2. Reflect
    3. Transmit
4. When to stop? Define a maximum number of allowed recursions.

Shadows

- To check if a point is under shadow or not, cast a ray from each point to the light:
  - If it intersects something before reaching the light, it is in a shadow area.



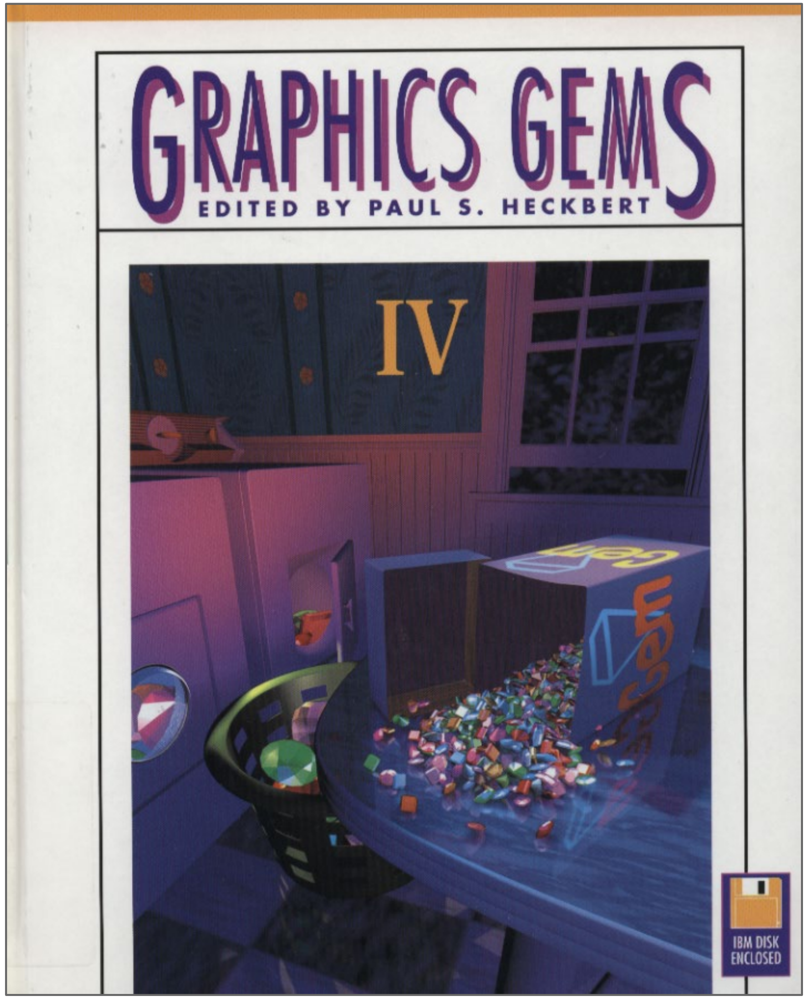
Building a ray tracer

```
trace(point, direction, step) {
    var local, reflected, transmitted;
    var p;
    var n;

    if(step > max) return background_color;
    p = intersect(point, direction, status);

    if(status == light_source) return light_source_color;
    if(status == no_intersection) return background_color;

    var n = normal(p);
    var r = reflect(p, n);
    var t = transmit(p, q);

    var local = phong(p, n, r);
    var reflected = trace(p, r, step+1);
    var transmitted = trace(p, t, step+1);

    return (local, reflected, transmitted);
}
```

A minimal ray tracer

- Contest created by Paul S. Heckbert in 1987: “***write the shortest Whitted-style ray tracing program in C***”.

PLACE	#TOKENS	AUTHOR	NOTES
		GENUINE ENTRIES	
1	916	Joe Cychosz, Purdue	compiler-dependent
1a	932	Joe Cychosz, Purdue	portable
2	956	Darwyn Peachey, Saskatchewan	portable
3	981	Michel Burgess, Montreal	portable
4	1003	Greg Ward, Berkeley	portable
		HONORABLE MENTIONS	
c1	10	Tony Apodaca, Pixar	cheater
c2	66	Greg Ward, Berkeley	cheater



UIC COMPUTER SCIENCE

A minimal ray tracer

minray.card.c

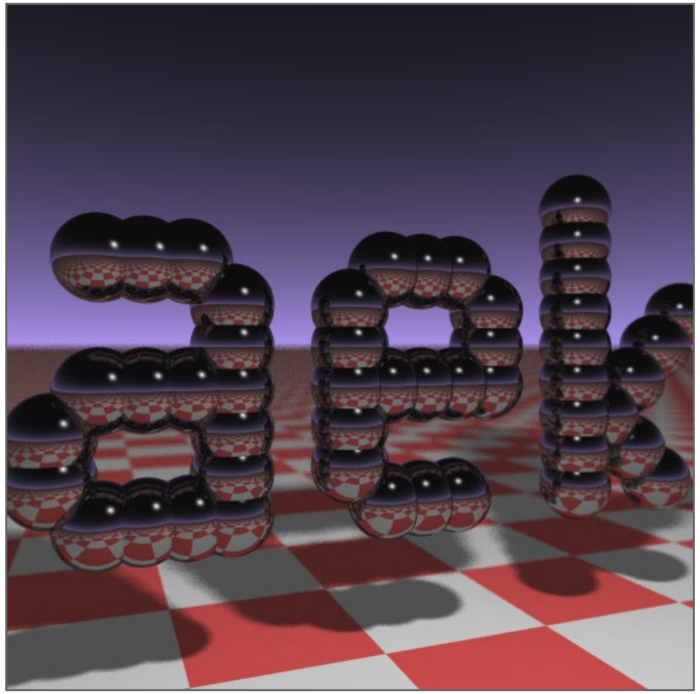
```c
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,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


UIC COMPUTER SCIENCE

A minimal ray tracer

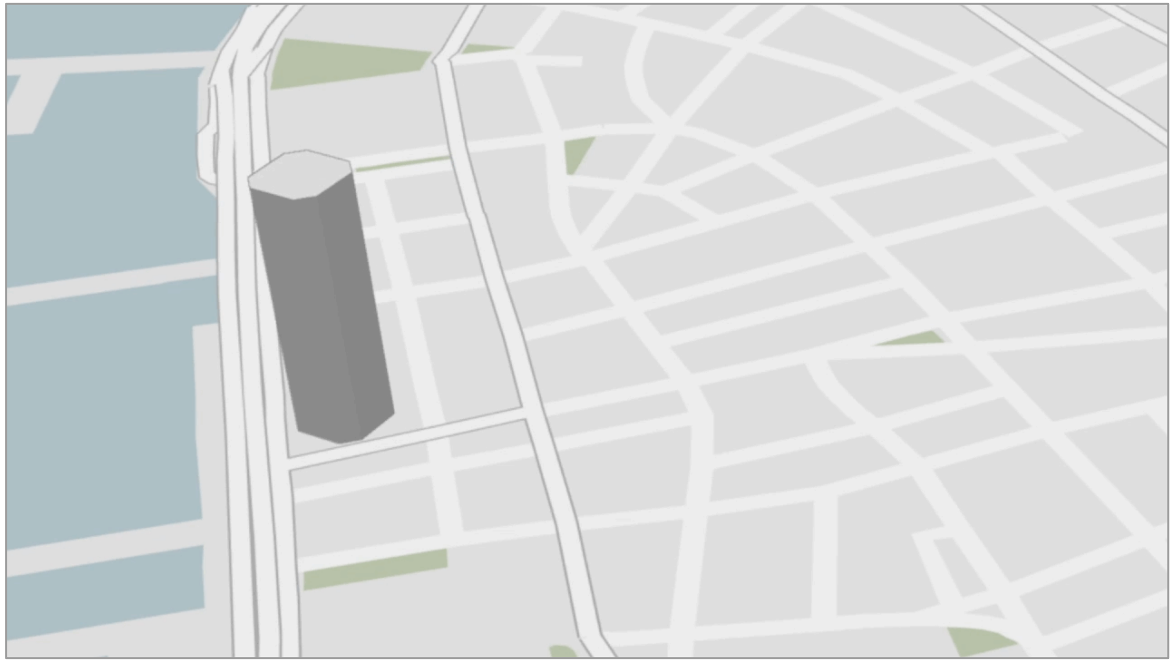
- Andrew Kensler’s business card sized raytracer

```
#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);}}
```

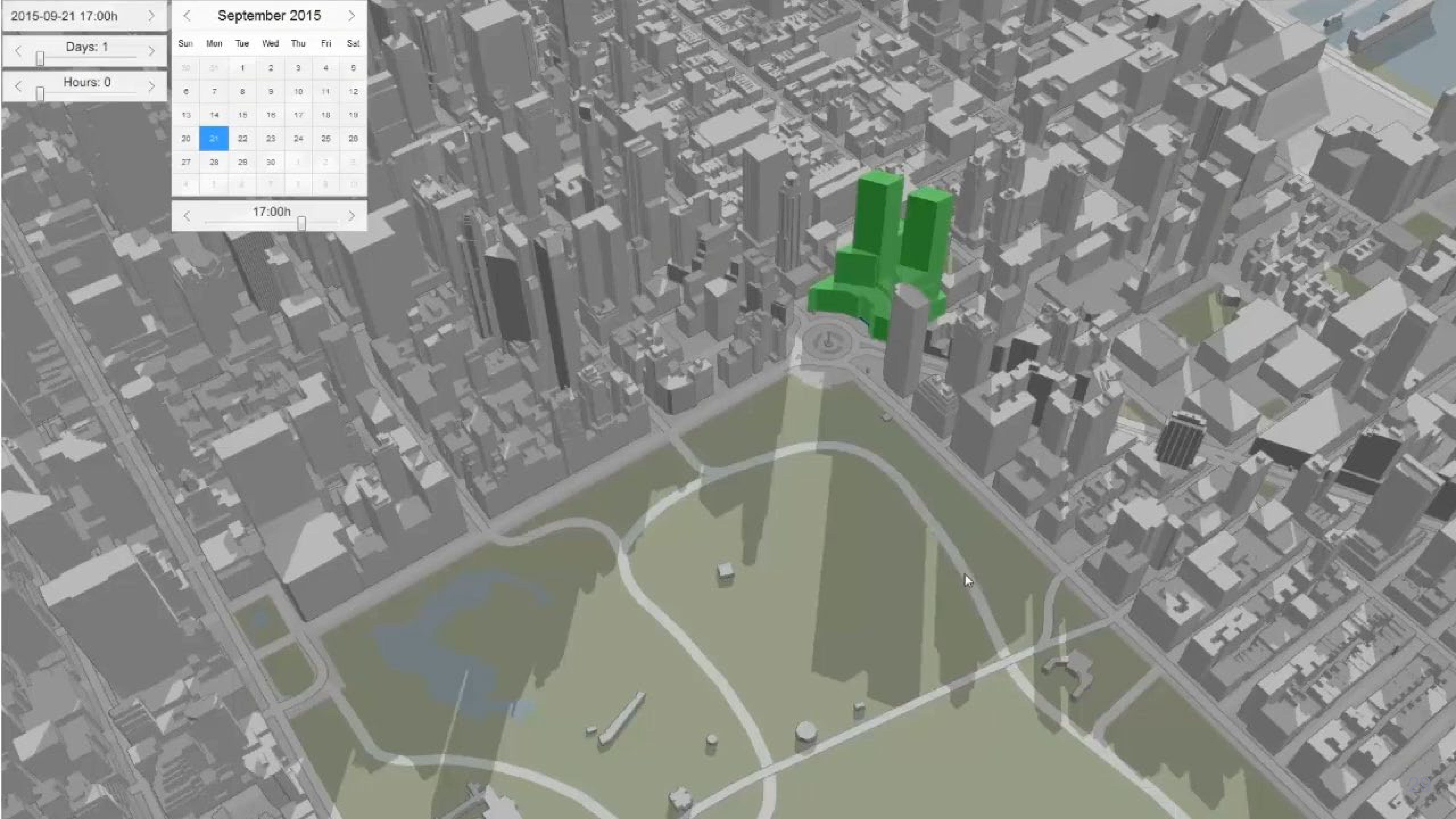


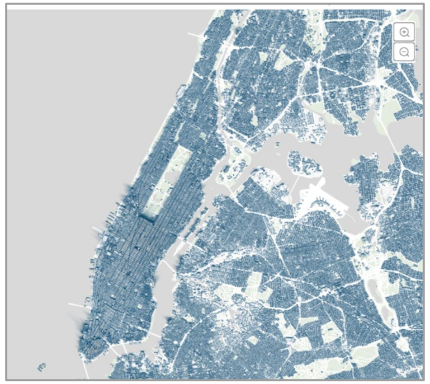
Shadow Accrual Maps

- Concrete example of ray tracing: shadow accumulation.
- “Shadow Accrual Maps: Efficient Accumulation of City-Scale Shadows Over Time”

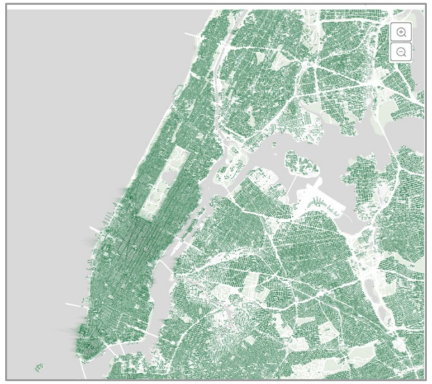


UIC COMPUTER SCIENCE


2015-09-21 17:00h
Days: 1
Hours: 0
September 2015
Sun Mon Tue Wed Thu Fri Sat
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
17:00h



Winter



Spring / Fall

Summer

OptiX: modern ray tracer

- Ray tracing API by Nvidia.
- Applications beyond graphics: optical & acoustical design, radiation, etc.

OptiX: modern ray tracer

```
#include <optix.h>

optixContext = optix::Context::create();
optixContext->setRayTypeCount(1);
optixContext->setEntryPointCount(1);
optixContext->setStackSize(4640);
optixContext->setPrintEnabled(true);
optixContext->setExceptionEnabled(RT_EXCEPTION_ALL, true);


optixContext->setRayGenerationProgram(0, optixProgramGeneration);
optixContext->setExceptionProgram(0, optixProgramException);
optixContext->setMissProgram(0, optixProgramMiss);

optix::Acceleration acceleration = optixContext->createAcceleration("Lbvh");
acceleration->setTraverser("Bvh");
```

OptiX: modern ray tracer

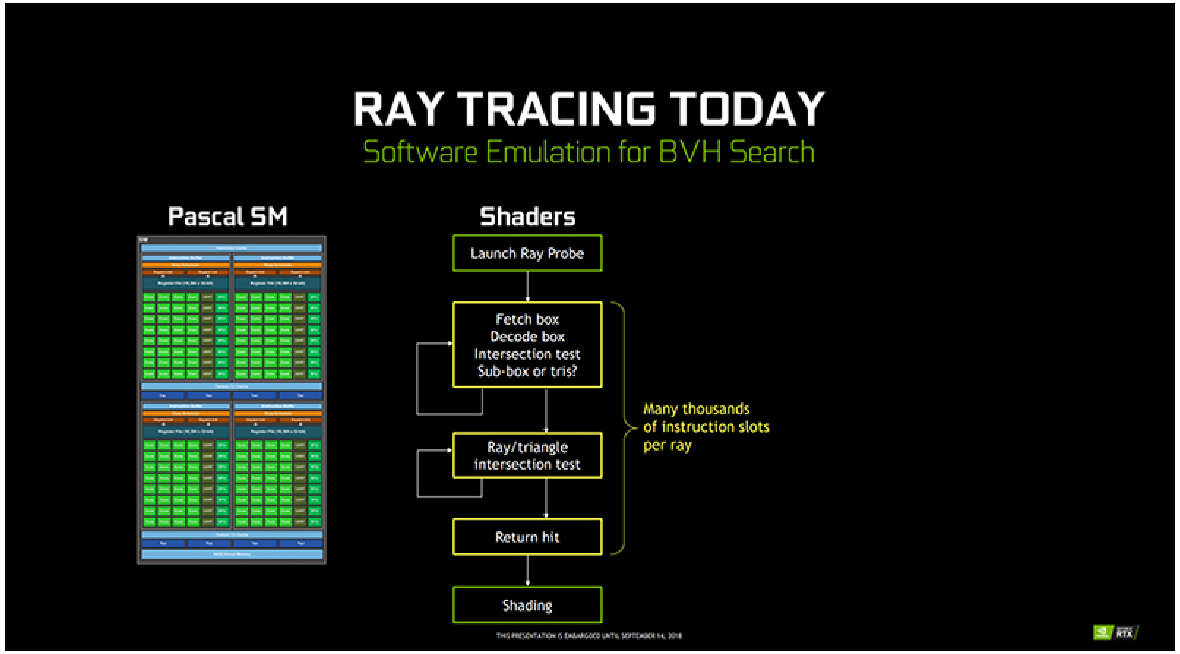
```
#include <optix.h>

struct PerRayData_radiance
{
    float result;
    float importance;
    int depth;
};

rtDeclareVariable(optix::Ray, ray, rtCurrentRay, );

RT_PROGRAM void generation() {
    float4 pos = pointBuffer[launchIndex];
    float4 dir = directionBuffer[i];
    optix::Ray ray = optix::make_Ray(make_float3(pos), make_float3(dir), radianceRayTyp
e, sceneEpsilon, RT_DEFAULT_MAX);

    PerRayData_radiance prd;
    rtTrace(topObject, ray, prd);
    outputBuffer[launchIndex] += prd.result;
}
```

OptiX: modern ray tracer

```
RT_PROGRAM void miss() {
    prdRadiance.result = 0;
}

RT_PROGRAM void anyHit() {
    prdRadiance.result = 1;
    rtTerminateRay();
}

RT_PROGRAM void closestHit() {
    prdRadiance.result = 1;
    rtTerminateRay();
}
```

Ray-tracing cores

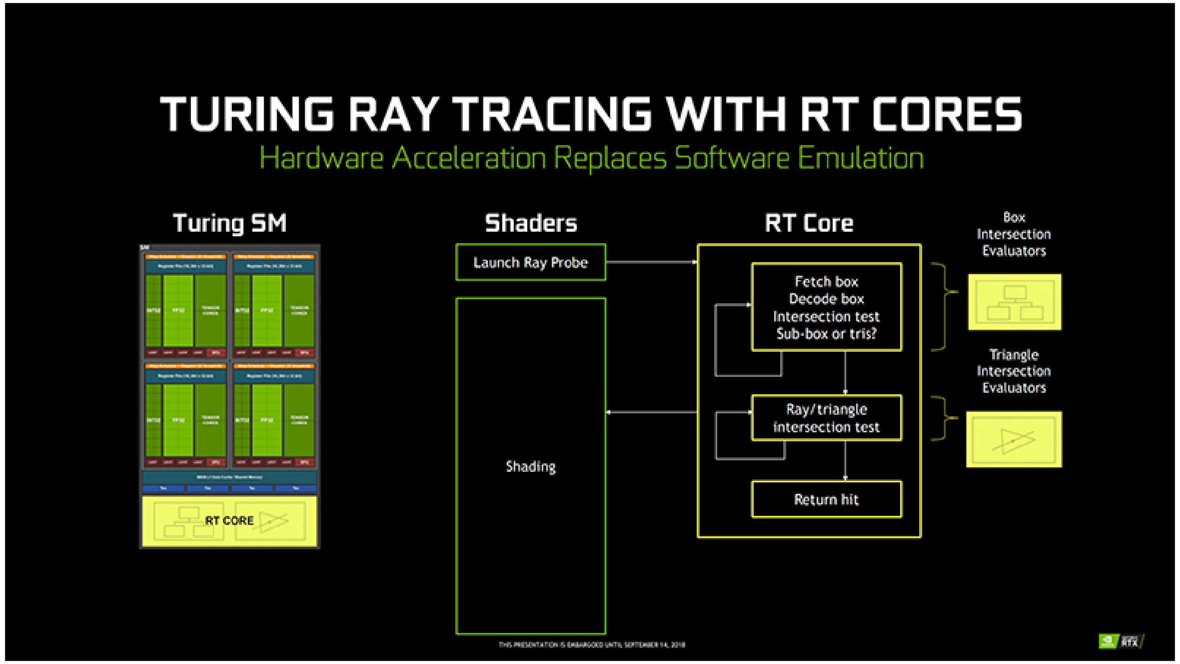
- RT cores: two specialized units for bounding box tests, and ray-triangle intersection tests.
- Frees up streaming multiprocessor (SM) from spending instruction slots per ray.



Before (Pascal microarchitecture)

Ray-tracing cores

- RT cores: two specialized units for bounding box tests, and ray-triangle intersection tests.
- Frees up streaming multiprocessor (SM) from spending instruction slots per ray.



After (Turing microarchitecture)