Lecture 15: Ray Casting

Rui Wang
Outline

- What is ray tracing?
- Basic algorithm
- Ray-shape intersection
- Constructing camera rays
- Ray tracing acceleration structures
- Illumination and shading (revisit)
- Reflection, refraction, shadows
3D Rendering

- Geometry
- Materials
- Lights
- Camera
Rasterization-based Rendering (OpenGL)

Modeling Transformation

Lighting

Viewing Transformation

Projection Transformation

Clipping

Scan Conversion

Image

Object-order: primitive by primitive
Ray tracing

1. Eye ray
2. Intersect
3. Shading
4. Output pixel
5. Finished?

Image-order: pixel by pixel
Ray tracing
Ray tracing

- Eye ray
  - Intersect
    - Shading
      - Recursion?
        - Output pixel
          - Finished?
            - Image

Image-order: pixel by pixel
Ray Tracing

- Not a new idea
  - Greek philosophers: light rays go from the eye to the light, or from the light to the eye?
  - Gauss: trace rays through lenses

- Why does ray tracing work?
  - Reciprocity: optical properties remain the same under path reversal.
  - So why do we trace from the eyes as opposed to trace from the lights?
Writing ray tracer is fun

Heckbert’s Business Card Ray Tracer

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.,0,1.2,1.,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>/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;1=sph+5;while(1-->sph)if((e=1
->kl*vdot(N,U=vunit(vcomb(1.-1.P,1.--(P,cen))))))>0&intersect(P,U)=1)color=vcomb(e
,1-->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);}$/pixar!ph*/
Writing ray tracer is fun

Heckbert's Business Card Ray Tracer
Advantages of Ray Tracing

- Easy to handle complex illumination effects (reflections, refractions, soft shadows, indirect illumination...)
- Very flexible in handling various shapes → only need an intersection routine.
- Asymptotically cheaper
Disadvantages of Ray Tracing

- Performance (due to weak object space coherence)
- Typically needs preprocessing to build fast intersection structure,
- Difficult to handle dynamic scenes
Ray Tracing

Virtual Viewpoint

Virtual Screen

Objects

Ray misses all objects: Pixel colored black
Ray intersects object: Shade using color, lights, materials
Multiple intersections: Use closest one (as does OpenGL)
Shadows

Light Source

Virtual Viewpoint

Virtual Screen

Objects

Shadow ray to light is unblocked: object visible
Shadow ray to light is blocked: object in shadow
Simple Ray Tracing

Camera camera;
vector<Shape> shapes;
vector<Light> lights;
int im_width, im_height;

Scene Definition
Simple Ray Tracing

Camera camera;
vector<Shape> shapes;
vector<Light> lights;
int im_width, im_height;

Image render(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 = camera.getCameraRay(i, j);
            hit = IntersectScene(ray, shapes);
            image[i][j] = getShadingColor(ray, hit);
        }
    }
    return image;
}
Simple Ray Tracing

Camera camera;
vector<Shape> shapes;
vector<Light> lights;
int im_width, im_height;

Image render(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 = camera.getCameraRay(i, j);
            hit = IntersectScene(ray, shapes);
            image[i][j] = getShadingColor(ray, hit);
        }
    }
    return image;
}
Simple Ray Tracing

```java
Camera camera;
vector<Shape> shapes;
vector<Light> lights;
int im_width, im_height;

Image render(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 = camera.getCameraRay(i, j);
            hit = IntersectScene(ray, shapes);
            image[i][j] = getShadingColor(ray, hit);
        }
    }
    return image;
}
```
Simple Ray Tracing

Camera camera;
vector<Shape> shapes;
vector<Light> lights;
int im_width, im_height;

Image render(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 = camera.getCameraRay(i, j);
            hit = IntersectScene(ray, shapes);
            image[i][j] = getShadingColor(ray, hit, lights);
        }
    }
    return image;
}
Ray-Shape Intersection (single shape)

- Ray-Plane
- Ray-Sphere
- Ray-Triangle
- Ray-Implicit Surface

Task: take ray, compute intersection and return hit hit structure

hitRecord Shape::hit(const Ray&);
Ray-Shape Intersection (many shapes)

```cpp
bool IntersectScene(ray, HitRecord& hit)
{
    float tmax = Float.MAX_VALUE;
    boolean hit_something = false;
    for (Shape s : shapes)
    {
        HitRecord temp_hit = s.hit(ray, 0, tmax)
        if (temp_hit != null)
        {
            tmax = temp_hit.t;
            hit.set(temp_hit);
            hit_something = true;
        }
    }
    return hit_something;
}
```