CS 543: Computer Graphics
Lecture 10 (Part II): Raytracing (Part II)

Emmanuel Agu

Where are we?

```
Define the objects and light sources in the scene
Set up the camera
for(int r = 0; r < nRows; r+= blockSize){
  for(int c = 0; c < nCols; c+= blockSize){
     1. Build the rc-th ray
     2. Find all object intersections with rc-th ray
     3. Identify closest object intersection
     4. Compute the “hit point” where the ray hits the
         object, and normal vector at that point
     5. Find  color (clr) of light to eye along ray
     glColor3f(clr.red, clr.green, clr.blue);
     glRecti(c, r, c + blockSize, r + blockSize);
  }
}
```

Find Object Intersections with rc-th ray

- Much of work in ray tracing lies in finding intersections with generic objects
- Break into two parts
  - Deal with untransformed, generic (dimension 1) shape
  - Then embellish to deal with transformed shape
- Ray generic object intersection best found by using implicit form of each shape. E.g. generic sphere is

$$F(x, y, z) = x^2 + y^2 + z^2 - 1$$

- Approach: ray r(t) hits a surface when its implicit eqn = 0
- So for ray with starting point S and direction **c**

$$r(t) = S + \mathbf{c}t$$

$$F(S + \mathbf{c}t_{hit}) = 0$$

Ray Intersection with Generic Plane

- Generic Plane?
- Yes! Floors, walls, in a room, etc
- Generic plane is *xy*-plane, or *z* = 0
- For ray

$$r(t) = S + \mathbf{c}t$$

- There exists a t_{hit} such that

$$S_z + \mathbf{c}_z t_h = 0$$

- Solving,

$$t_{hit} = -\frac{S_z}{c_z}$$

Ray Intersection with Generic Plane

- Hit point P_{hit} is given by

$$P_{hit} = S - \mathbf{c}(S_z / c_z)$$

- Numerical example?
- Where does the ray *r(t)* = (4, 1, 3) + (-3, -5, -3)*t* hit the generic plane?
- Soln:

$$t_{hit} = -\frac{S_z}{c_z} = -\frac{3}{3}$$

- And hit point is given by

$$S + \mathbf{c} = (1, -4, 0)$$

Ray Intersection with Generic Sphere

- Generic sphere has form

$$x^2 + y^2 + z^2 = 1$$

$$x^2 + y^2 + z^2 - 1 = 0$$

$$F(x, y, z) = x^2 + y^2 + z^2 - 1$$

$$F(P) = |P|^2 - 1$$

- Substituting S + **c**t in *F(P) = 0,* we get

$$|S + \mathbf{c}t|^2 - 1 = 0$$

$$|\mathbf{c}|^2 t^2 + 2(S \cdot \mathbf{c})t + (|S|^2 - 1) = 0$$

- This is a quadratic equation of the form $At^2 + 2Bt + C = 0$ where $A = |\mathbf{c}|^2$, $B = S.\mathbf{c}$ and $C = |S|^2 - 1$

Ray Intersection with Generic Sphere

- Solving

$$t_h = -\frac{B}{A} \pm \frac{\sqrt{B^2 - AC}}{A}$$

- If discrimant $B^2 - AC$ is negative, no solutions, ray misses sphere
- If discriminant is zero, ray grazes sphere at one point and hit time is –B/A
- If discriminant is +ve, two hit times t1 and t2 (+ve and –ve) discriminant
- Numerical example? See example 12.4.2 on pg. 619

What about transformed Objects

- Generic objects are untransformed:
  - No translation, scaling, rotation
- Real scene: generic objects instantiated, then transformed by a composite matrix *T*,
- We can easily find the inverse transform *T'*
- **Problem definition:** We want to find ray intersection with transformed object
- Easy by just simply finding the implicit form of the transformed object
- May be tough to find implicit form of transformed object
- Hmmm… is there an easier way?

What about transformed Objects

- Yes
- Basic idea: if object is transformed by *T*, then ray–object intersection is the same as inverse transformed ray with generic object
- Algorithm
  - Find *T'* from initial *T* transform matrix of object
  - Inverse transform the ray to get (S' + $\mathbf{c'}t$)
  - Find intersection time, t_{hit} of the ray with the generic object
  - Use the *same* t_{hit} in S + $\mathbf{c}t$ to identify the actual hit point
- This beautiful trick greatly simplifies ray tracing
- We only need to come up with code that intersects ray with *generic* object
- Remember that programmer does transforms anyway, so we can easily track and get *T*

Dealing with Transformed Objects

- Thus we want to solve the equation

$$F(T^{-1}(S+\mathbf{c}t))=0$$

- Since transform *T* is linear

$$T^{-1}(S+\mathbf{c}t)=(T^{-1}S)+(T^{-1}\mathbf{c})t$$

- Thus inverse transformed ray is

$$\tilde{r}(t)=M^{-1}\begin{pmatrix}S_x\\S_y\\S_z\\1\end{pmatrix}+M^{-1}\begin{pmatrix}c_x\\c_y\\c_z\\0\end{pmatrix}t=\tilde{S}'+\mathbf{c}'t$$

Dealing with transformed Objects

- For example if we have the following SDL commands in our file

```
translate 2 4 9
scale 1 4 4
sphere
```

- The transform matrices are (see example 12.4.3, pg 621)

$$M = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 4 & 0 & 4 \\ 0 & 0 & 4 & 9 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad M^{-1} = \begin{pmatrix} 1 & 0 & 0 & -2 \\ 0 & \frac{1}{4} & 0 & -4 \\ 0 & 0 & \frac{1}{4} & -\frac{9}{4} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Organizing a Ray Tracer

- Need data structures to store ray, scene, camera, etc
- There are many ways to organize ray tracer
- Previously in C, declare **struct**
- These days, object-oriented religion?
- Friend once wrote ray tracer as java applet in Prof. Hill's class
- SDL generates scene file
- We've developed camera class (HW3: slide, roll, etc)
- Now just add a **raytrace** method to camera class

```
void Camera::raytrace(Scene& scn, int blockSize);
```

Organizing a Ray Tracer

- Call camera raytrace method from display (redisplay) function

```
void display(void){
   glClear(GL_COLOR_BUFFER_BIT); // clear the screen
   cam.raytrace(scn, blockSize);
}
```

- Thus ray tracer fires up and starts scanning pixel by pixel (or block by block) till entire screen is ray traced
- Can insert previous **drawOpenGL** function before raytrace to give scene preview
- Subtlety: drawOpenGL uses openGL 3D pipeline, raytrace uses 2D pipeline, so do pipeline set up inside each method

Organizing a Ray Tracer

- Need Ray class with start, dir variables and methods to set them

```
Class Ray{
Public:
   Point3 start;
   Vector3 dir;
   void setStart(point3& p){start.x = p.x; etc…}
   void setDir(Vector3& v){dir.x = v.x; etc…}
   // other fields and methods
};
```

- We can now develop a basic raytrace( ) skeleton function

Camera raytrace( ) skeleton

```
void Camera::raytrace(Scene& scn, int blockSize)
{
   Ray theRay;
   Color3 clr;
   theRay.setStart(eye);
   // set up OpenGL for simple 2D drawing
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity( );
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity( );
   gluOrtho2D(0, nCols, 0, nRows); // whole screen is window
   glDisable(GL_LIGHTING);

   //begin ray tracing
```

Camera raytrace( ) skeleton

```
    for(int row = 0; row < nRows; rows += blockSize)
    for(int col = 0; col < nCols; cols += blockSize)
    {
       compute ray direction
       theRay.setDir(<direction>); // set the ray’s direction
       clr.set(scn.shade(theRay));  // find the color
       glColor3f(clr.red, clr.green, clr.blue);
       glRecti(col, row, col + blockSize, row + blockSize)
    }
}
```

- shade( ) function does most of ray tracing work

shade( ) skeleton

```
Color3 Scene::shade(Ray& ray)
{   // return color of this ray
    Color3 color; // total color to be returned
    Intersection best; // data for best hit so far
    getFirstHit(ray, best); // fill “best” record
    if(best.numHits == 0) // did ray miss all objects?
        return background;
    color.set(the emissive color of object);
    color.add(ambient, diffuse and specular); // add contrib.
    color.add(reflected and refracted components);
    return color;
}
```

- getFirstHit function returns first object hit by ray
- Intersection class used to store each object’s hit information

shade( ) skeleton

- Intersection class used to store each object's hit information

```
Class Intersection{
Public:
   int numHits;    // # of hits at positive hit times
   HitInfo hit[8]; //list of hits – may need more than 8 later
   …. various hit methods
}
```

- hitInfo stores actual hit information for each hit
- For simple convex objects (e.g. sphere) at most 2 hits
- For torus up to 4 hits
- For boolean objects, all shapes possible so no limit to number of hits

HitInfo( ) class

```
class HitInfo{
Public:
   double hitTime;        // the hit time
   GeomObj* hitObject;    // the object  hit
   bool isEntering;       // is the ray entering or exiting
   int surface;           // which surface is hit?
   Point3 hitPoint;       // hit point
   Vector3 hitNormal;     // normal at hit point
   …. various hit methods
}
```

- Surface applies if it is convenient to think of object as multiple surfaces. E.g. cylinder cap, base and side are 3 different surfaces

getFirstHit( ) method

```
Void Scene::getFirstHit(Ray& ray, Intersection& best)
{
   Intersection inter;  // make intersection record
   best.numHits = 0;    // no hits yet

   for (GeomObj* pObj = obj;pObj !=NULL;pObj = pObj->next)
   { // test each object in the scen
      if(!pObj->hit(ray, inter)) // does the ray hit pObj?
         continue;               // miss: test the next object
      if(best.numHits == 0) ||   // best has no hits yet
        inter.hit[0].hitTime < best.hit[0].hitTime)
            best.set(inter);  //copy inter into best
}
```

- Sphere, cube, plane … are all derived from base GeomObj class

getFirstHit( ) method

- Polymorphism: hit called in getFirstHit( ) is a virtual function.
- hit is implemented differently for each object based on its implicit equations
- So, sphere, cylinder, cube … all have their hit( ) functions
- Much of raytracing work lies in writing these hit( ) functions
- Next, hit( ) function for sphere

References

- Hill, chapter 12