Martin Bojinov
Kevin Smith
CS 216
11 April 2024

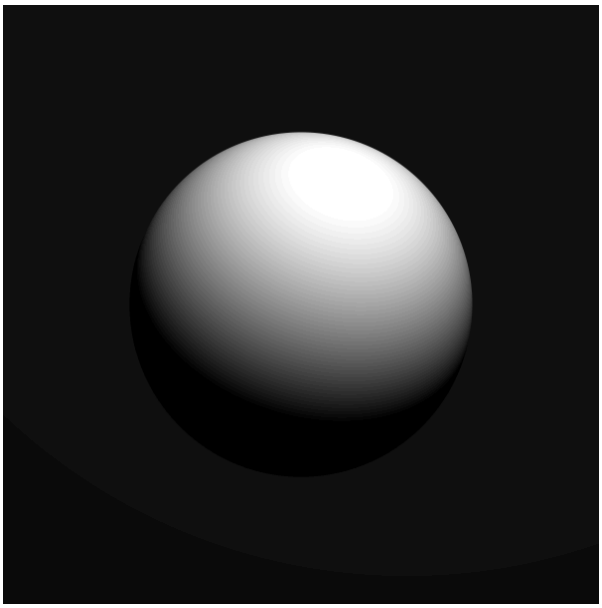
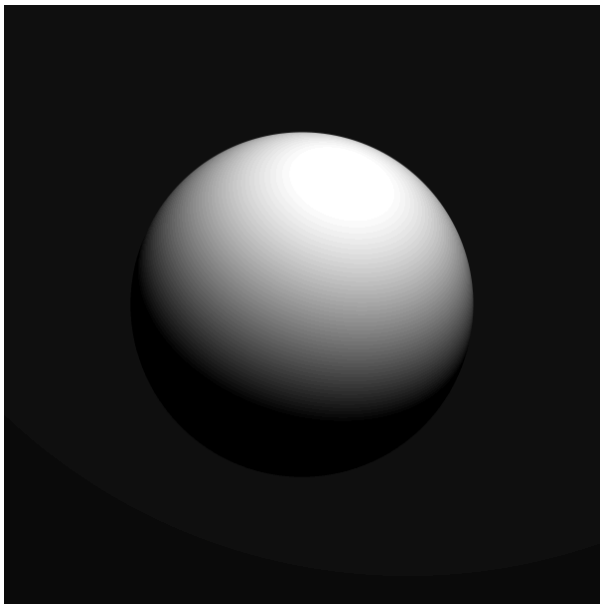
Project 2 Part 1

Step 1 - Install PBRT

Done as a prerequisite. I used PBRT v3.

Step 2 - Clone Sphere

Here are the results of this step. The left image is rendering a sphere, while the right image is rendering a “RayMarcher” that has all the same code as a sphere.

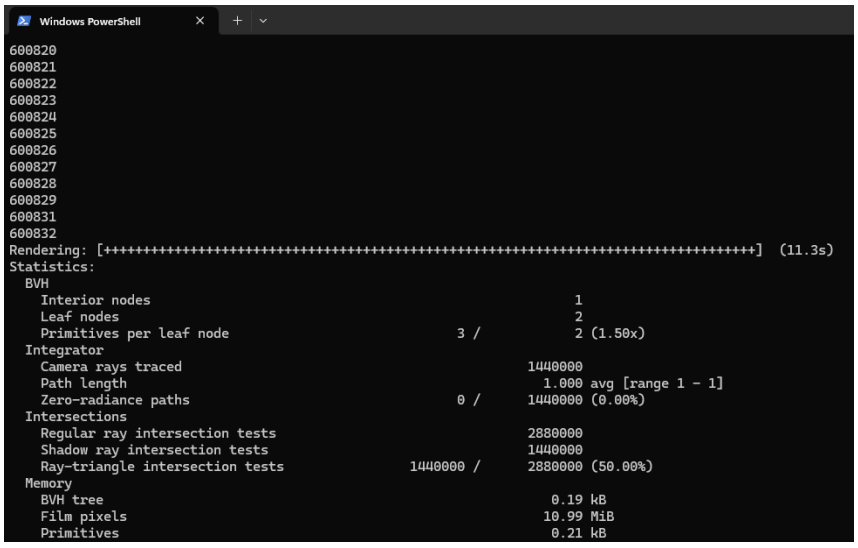
Sphere	“RayMarcher”
	

Step 3 - Strip down functions

Here are the results of using the provided code.

Step 4 - Print Intersect Function

For this step, we print a count within the intersect function. There were 600,832 rays. For me, the counts were all sequential. I tried running with multiple cores and it was still sequential. My guess is that if it were not sequential, it would be related to running it single-threaded vs multithreaded.



Step 5 - SDF Function

See implemented code (also submitted alongside pdf).

```
//  Template Method
//
Float RayMarcher::sdf(const Point3f &pos) const {
  float d = Distance(pos, Point3f(0, 0, 0));  // distance between origin and point
  return d - radius;                          // distance minus radius
}
```

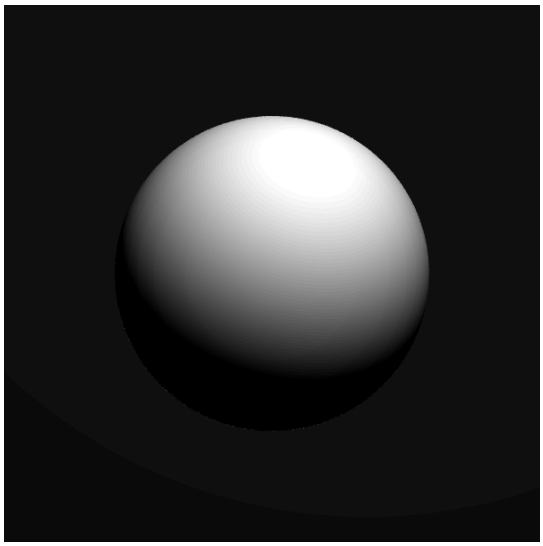
Step 6 - Ray Marcher

See implemented code (also submitted alongside pdf).

```
Vector3f dir = Normalize(r.d);  // ray direction vectors are not normalized in PBRT by default (KMS)
bool hit = false;
Point3f p = r.o;  // start point

// ray marching algorithm from the slides
for (int i=0; i < max_ray; i++) {
  float dist = this->sdf(p);
    if (dist < dist_thresh) {    // hit, return true
      hit = true;
      break;
    }
    else if (dist > max_dist) { // miss, return false
      break;
    }
    else {  // march
      p = p + (dir * dist);
    }
}

if (hit && tHit != nullptr && isect != nullptr) {
  // Thiis where you return your SurfaceInteraction structure and your tHit
  // Important Note: You must check for null pointer as Intersect is called
  // by IntersectP() with null values for these parameters.
  Vector3f default_norm = Vector3f(0, 0, 0);
  Vector3f norm = this->GetNormalRM(p, normal_eps, default_norm);

  Point3f pHit = p;
  Vector3f pError = Vector3f(10 * max_dist, 10 * max_dist, 10 * max_dist);
  Point2f uv = Point2f(0, 0);
  Normal3f dndu = Normal3f(0,0,0);
  Normal3f dndv = Normal3f(0,0,0);
  Vector3f dpdu;
  Vector3f dpdv;

  CoordinateSystem(norm, &dpdu, &dpdv);

  *isect = (*ObjectToWorld)(SurfaceInteraction(pHit, pError, uv, -r.d, dpdu, dpdv, dndu, dndv, r.time, this));

  // distance from ray start to hit point
  *tHit = (Float)Distance(r.o, p);
}
```

Step 7 - Ray Marcher Output

After implementing the Ray Marcher, SDF, and Norm functions I got the following result. I was really happy because I got it to run on the first try.



Step 8 - Implement Variables for Defined Constants

See implemented code (also submitted alongside pdf).

```
std::shared_ptr<Shape> CreateRayMarcherShape(const Transform *o2w, const Transform *w2o, bool reverseOrientation, const ParamSet &params) {
    int   max_ray     = params.FindOneFloat("max ray", 1000);
    Float dist_thresh = params.FindOneFloat("distance threshold", 0.01f);
    int   max_dist    = params.FindOneFloat("max distance", 100);
    Float normal_eps  = params.FindOneFloat("normal eps", 0.01f);

    Float radius = params.FindOneFloat("radius", 1.f);
    Float zmin = params.FindOneFloat("zmin", -radius);
    Float zmax = params.FindOneFloat("zmax", radius);
    Float phimax = params.FindOneFloat("phimax", 360.f);
    return std::make_shared<RayMarcher>(o2w, w2o, reverseOrientation, radius, zmin, zmax, phimax, max_ray, dist_thresh, max_dist, normal_eps);
}
```