Ray tracing

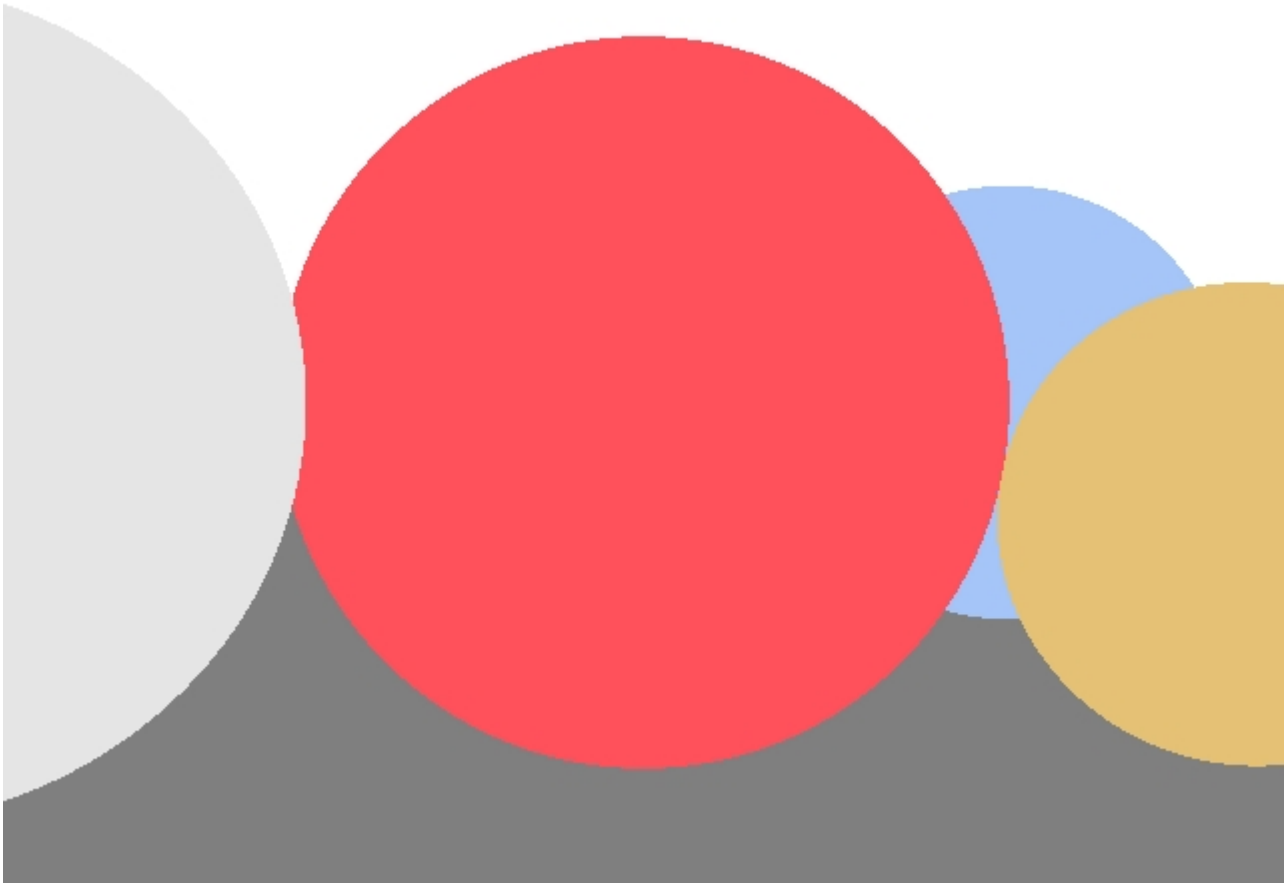
Rui Ying u1234364

Ray casting

Using function `intersect` can almost finish the task. However, we also need to pay attention to `ray` and `line` intersect with spheres.

So not only the boolean return value but also `t0` and `t1` need considering. Because `t1` is always no less than `t0`, we only need to limit `t0` here.

There is also a problem that needs paying attention. We need to cast the color on the nearest sphere to the screen pixel so that everything looks logical. And the depth relationship is correct.

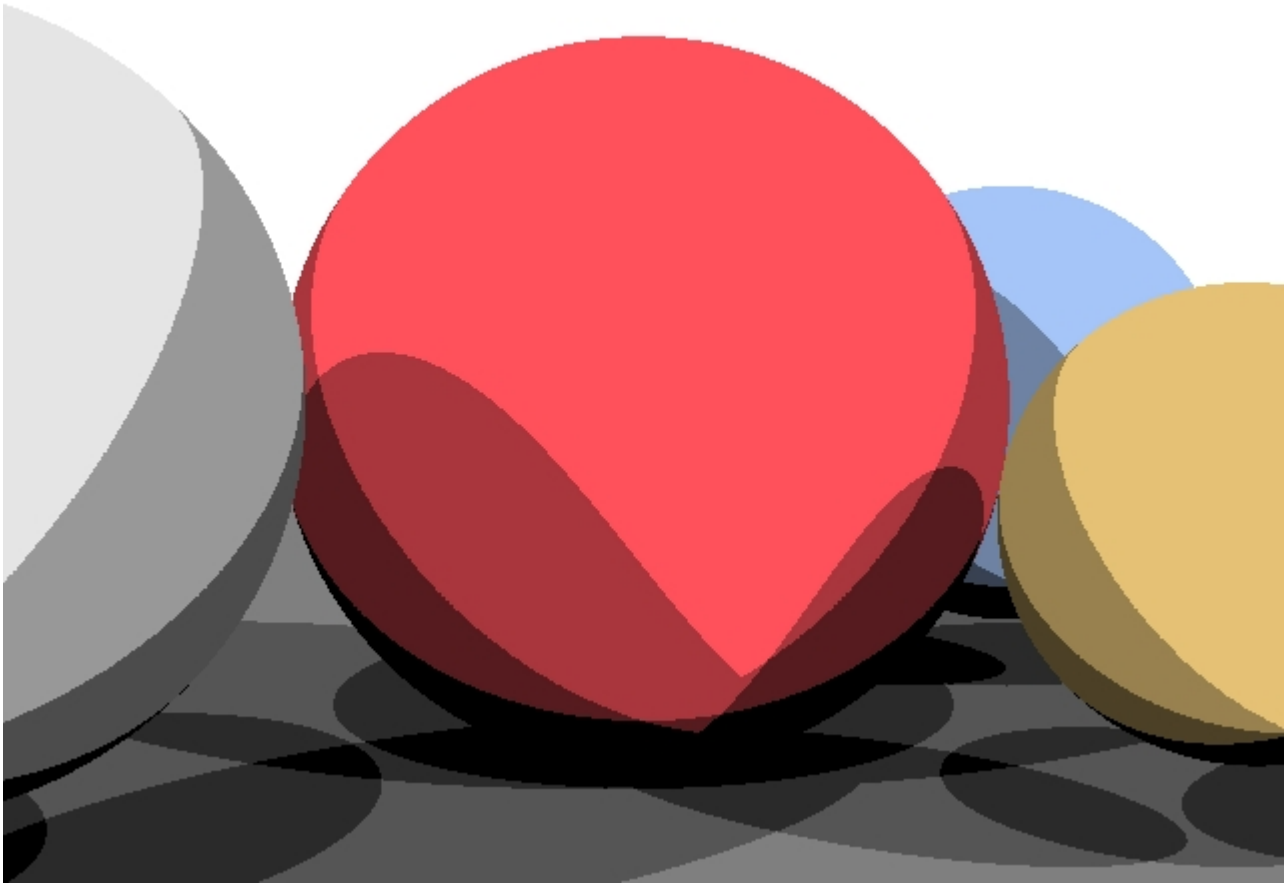


Shadow rays

I have had two problems when working this task.

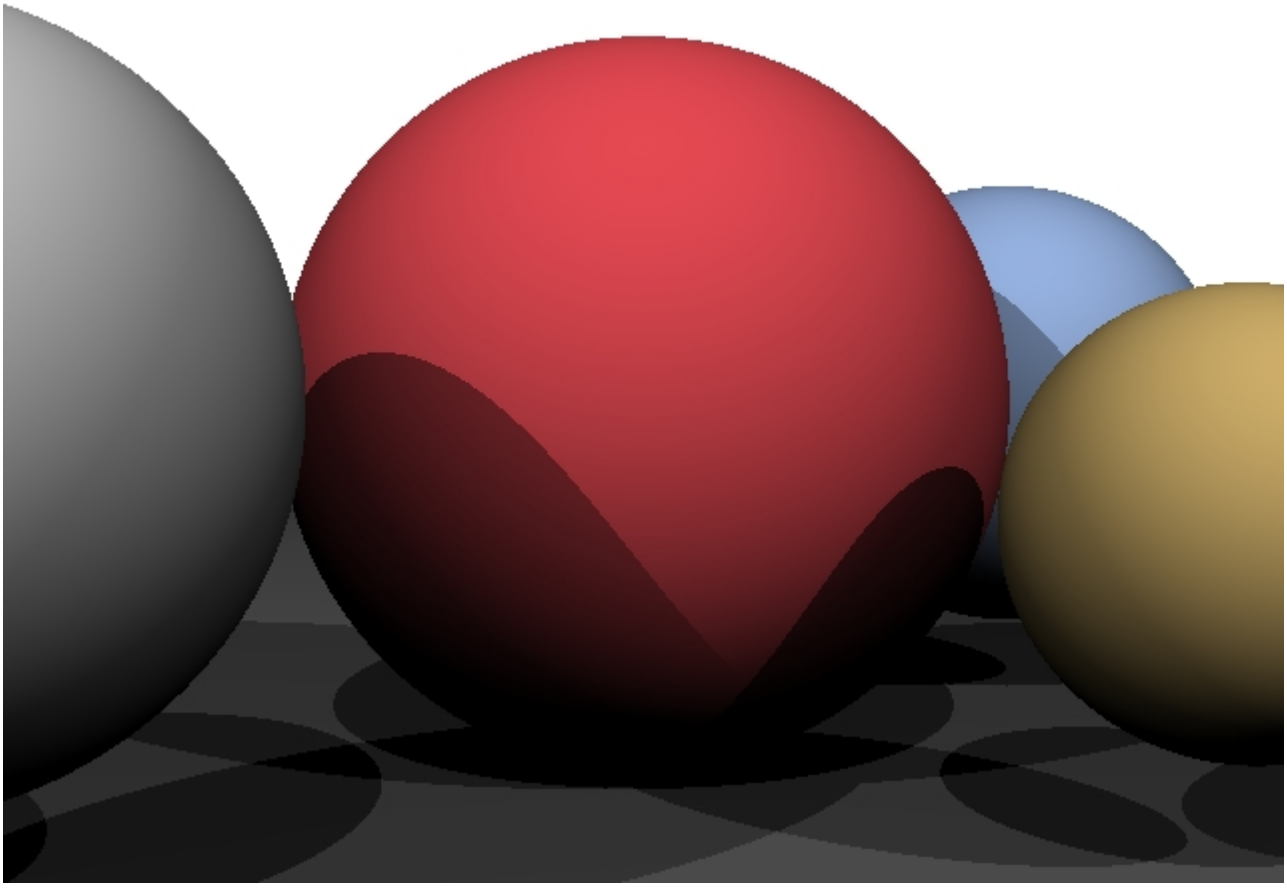
First, the description in homework PDF is not accurate. It says "If the path of light between the light source and P is blocked by some other sphere". `some other` is not accurate because the light can be blocked by the sphere which is currently hit by the ray.

Second, even after successfully implementing the whole function, shading seems strange and incomplete. I looked for info online and found the slides here is quite useful. The problem was caused by float point rounding which means it's problematic to compare a float with zero. So I use some small float number to act as zero.



Illumination models

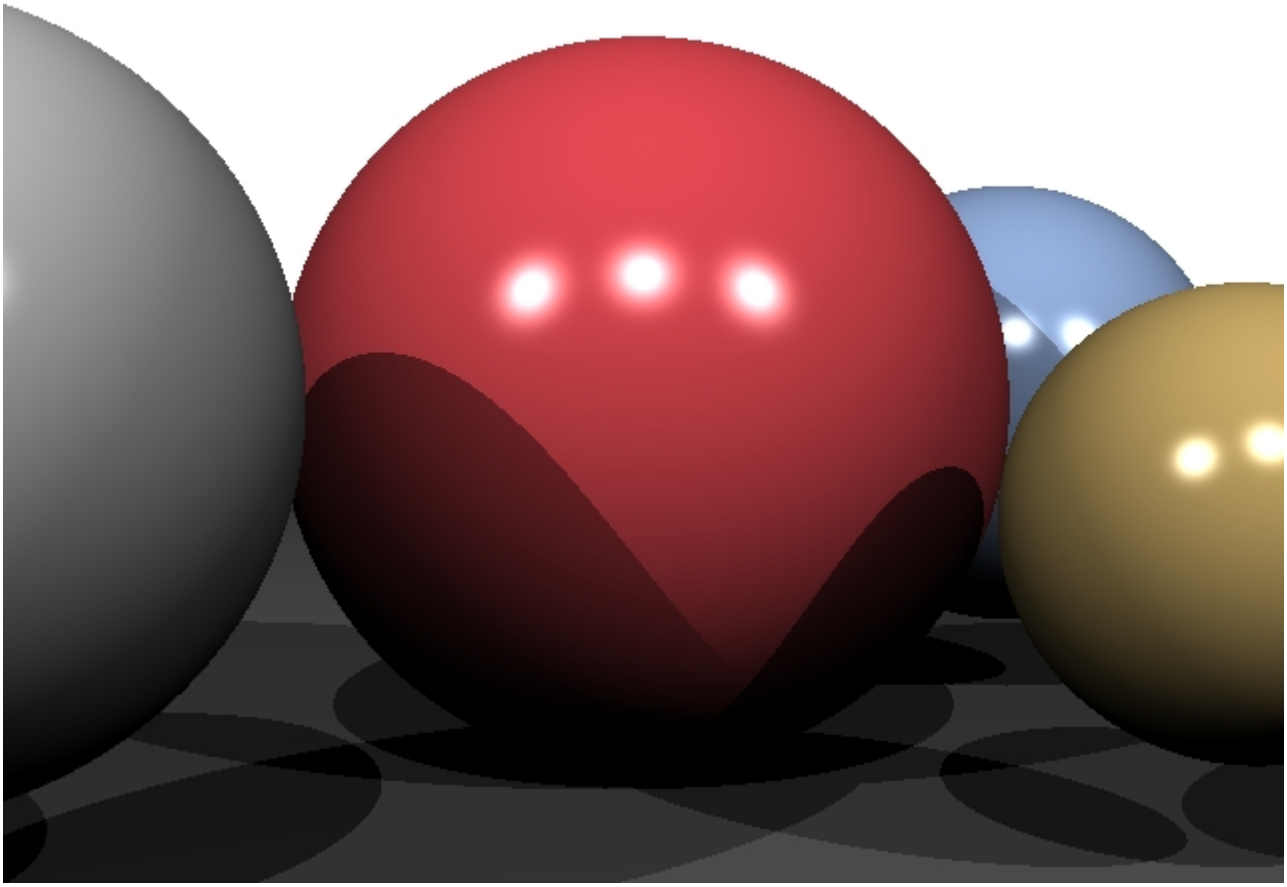
For `diffuse`, just do as the PDF say.



For `Phong` function, we need to first compute the reflecting vector by `Vector3f R = 2 * N*(N.dot(L)) - L`. Then apply the `Phong` method with `diffuse`:

```
resColor = diffuse(L, N, diffuseColor, kd) + ks * std::powf(std::max(R.dot(V), 0.f), alpha) * specularColor;
```

Here, `V = -rayDirection`.



Reference

CS4620/5620: Lecture 35 Ray Tracing (Shading)