Yingjie Lian
CS 4600
November 22th, 2018

HW5 - Raytracing

1. Ray casting

Here is my source code:

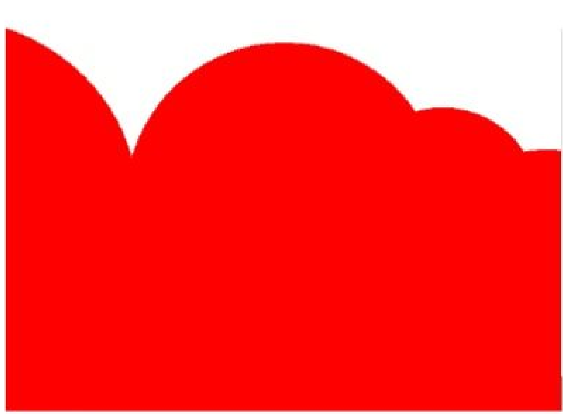
```
Vector3f trace(
    const Vector3f &rayOrigin,
    const Vector3f &rayDirection,
    const std::vector<Sphere> &spheres)
{
    //Vector3f pixelColor = Vector3f::Zero();
    // TODO: implement ray tracing as described in the homework description
    Vector3f pixelColor = bgcolor;

    Vector3f B = Vector3f::Zero();

    float t0 = INFINITY, t1 = INFINITY;
    float ct0, ct1;
    int closestSphere = -1;

    Vector3f t0Point;


    for (unsigned int s = 0; s < spheres.size(); s++) {

        bool doesIntersect = spheres[s].intersect(rayOrigin, rayDirection, ct0, ct1);

        if (doesIntersect) {
            if (t0 > ct0) {
                t0 = ct0;
                closestSphere = s;
                t0Point = rayOrigin + (t0 * rayDirection);
            }
        }

    }
```

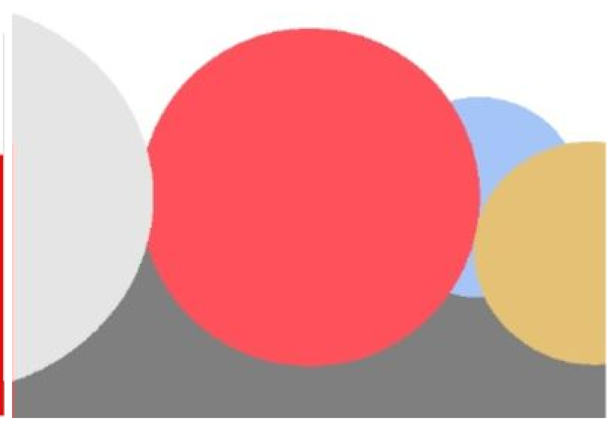
```
float l0, l1;

for (unsigned int l = 0; l < lightPositions.size(); l++) {

    Vector3f lightVector = lightPositions[l] - t0Point;
    lightVector.normalize();

    bool doesIntersect = false;

    for (unsigned int s = 0; s < spheres.size(); s++) {

        if (spheres[s].intersect(t0Point, lightVector, l0, l1)) {
            doesIntersect = true;
        }
    }
    if (!doesIntersect) {

        // This is for the orginal color image shadowed
        //B += 0.333 * spheres[closestSphere].surfaceColor;

        Vector3f surfaceNorm = t0Point - spheres[closestSphere].center;
        surfaceNorm.normalize();

        // This is for the diffuse
        //B += diffuse(lightVector, surfaceNorm, spheres[closestSphere].surfaceColor, 1);

        Vector3f rayNorm = rayDirection * -1;
        Vector3f specularColor = Vector3f::Ones();

        // This is for phong shading
        B += phong(lightVector, surfaceNorm, rayNorm, spheres[closestSphere].surfaceColor, specularColor, 1, 3, 100)
    }
}
```

```
if (closestSphere != -1) {
    pixelColor = B;
}
```

Implementing this function highlights most of the purpose of ray tracing. For each pixel, shoot a ray to determine which sphere (if any) are intersected. To do this, I made use of the **Sphere:: intersect** function and stored the shortest distance along with the corresponding sphere. Once all the spheres were visited, I determined the point (P) of the intersection.

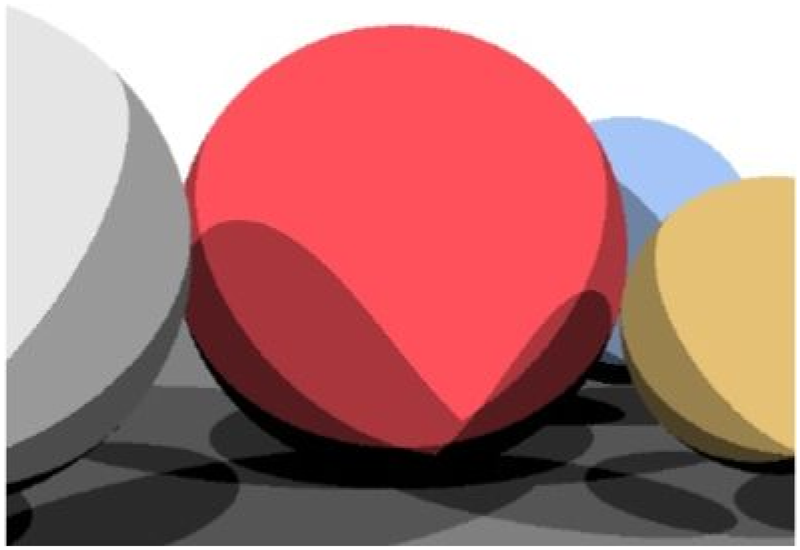
First, I returned red if there was an intersection. Then, I returned the color of the sphere.





2. Shadow rays

Had a little bit of trouble at first, then I went to the TA hours and I found that I can use the trace function. I added tracing from point P on the sphere to each light source in direction L to determine if P is lit by the source.



3. Illumination models

Here is my source code:

```
// diffuse reflection model
Vector3f diffuse(const Vector3f &L, // direction vector from the point on the surface towards a light source
    const Vector3f &N, // normal at this point on the surface
    const Vector3f &diffuseColor,
    const float kd // diffuse reflection constant
)
{
    Vector3f resColor = Vector3f::Zero();

    // TODO: implement diffuse shading model

    if (L.dot(N) > 0) {
        resColor = 0.333 * kd * L.dot(N) * diffuseColor;
    }

    return resColor;
}
```

```
// Phong reflection model
Vector3f phong(const Vector3f &L, // direction vector from the point on the surface towards a light source
    const Vector3f &N, // normal at this point on the surface
    const Vector3f &V, // direction pointing towards the viewer
    const Vector3f &diffuseColor,
    const Vector3f &specularColor,
    const float kd, // diffuse reflection constant
    const float ks, // specular reflection constant
    const float alpha) // shininess constant
{
    Vector3f resColor = Vector3f::Zero();

    // TODO: implement Phong shading model
    Vector3f R = 2 * N * N.dot(L) - L;

    float max;

    if (R.dot(V) > 0) {
        max = std::pow(R.dot(V), alpha);
    }
    else {
        max = std::pow(0, alpha);
    }

    resColor = diffuse(L, N, diffuseColor, kd) + 0.333 * specularColor * ks * max;

    return resColor;
}
```

First, I replace the pixel color determined from part 2 with a diffusion model component as presented in lecture 14 shading slides. The body is implemented with the following model:

$$E_D = I_i \cdot C_D \cdot k_d \cdot \max(\cos\alpha, 0)$$

I_i - Intensity of light i - 1/ light count
C_D – diffuse color (RGB) - di f f useCol or
k_D – diffuse coefficient (material) – kd
$\cos\alpha = L \cdot N$ – dot product of light direction and surface normal – L. dot (N)

Finally, add the specular component by implementing the Phong reflection model. The body is implemented with the following model:

$$E_S = I_i \cdot C_S \cdot k_S \cdot \max(\cos\beta, 0)^h$$

I_i - Intensity of light i - 1/ light count

C_S – highlight color (RGB) - specularColor

k_S – specular coefficient (material) – ks

$\cos\beta = R \cdot V$ – dot product of light direction and surface normal – R.dot(V)

$$R = 2N(N \cdot L) - L$$

h – shininess constant – alpha