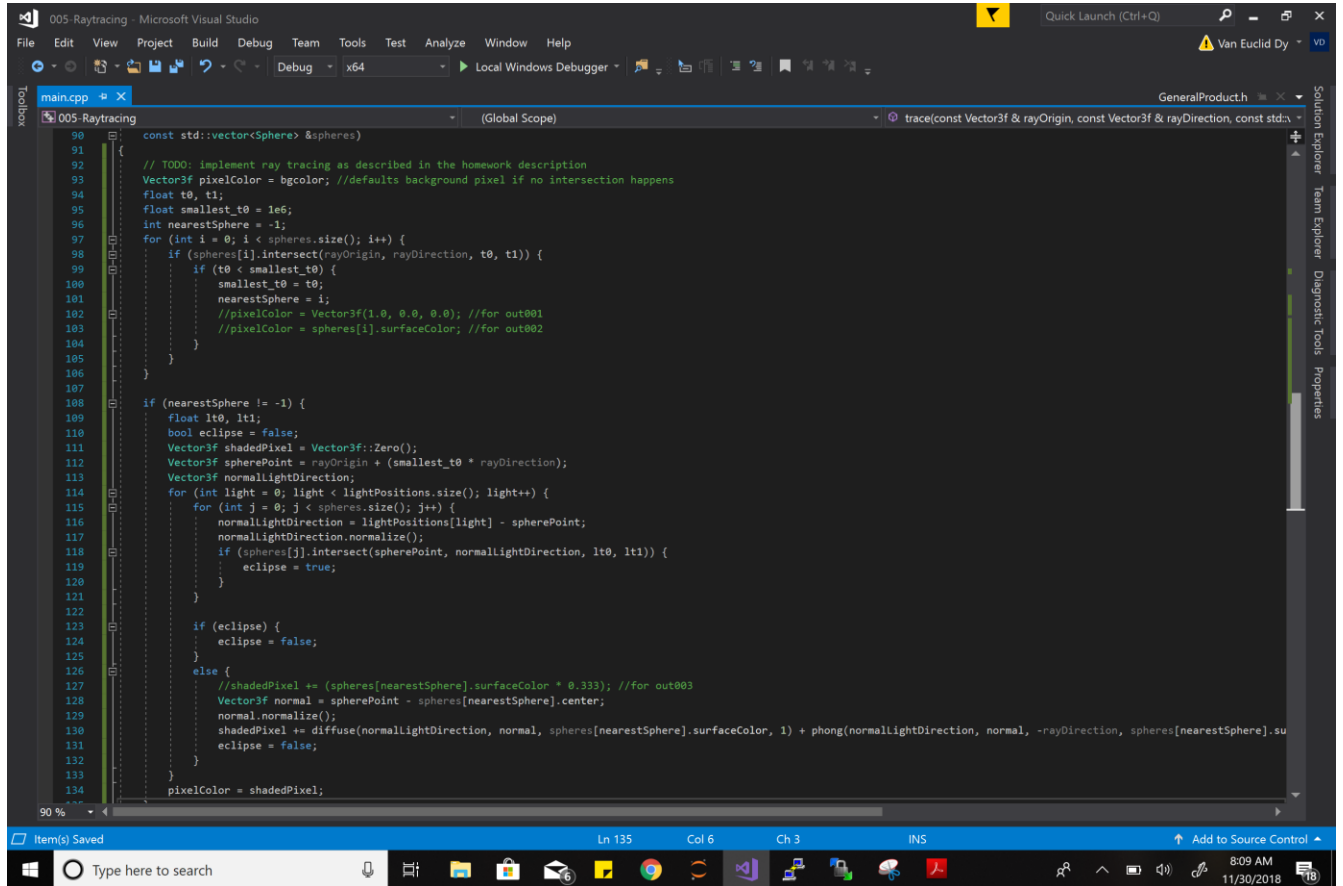


Van Euclid Dy

Raytracing write up

CS4600



```
90 const std::vector<Sphere> &spheres)
91 {
92     // TODO: implement ray tracing as described in the homework description
93     Vector3f pixelColor = bgcolor; //defaults background pixel if no intersection happens
94     float t0, t1;
95     float smallest_t0 = 1e6;
96     int nearestSphere = -1;
97     for (int i = 0; i < spheres.size(); i++) {
98         if (spheres[i].intersect(rayOrigin, rayDirection, t0, t1)) {
99             if (t0 < smallest_t0) {
100                 smallest_t0 = t0;
101                 nearestSphere = i;
102                 //pixelColor = Vector3f(1.0, 0.0, 0.0); //for out001
103                 //pixelColor = spheres[i].surfaceColor; //for out002
104             }
105         }
106     }
107
108     if (nearestSphere != -1) {
109         float lt0, lt1;
110         bool eclipse = false;
111         Vector3f shadedPixel = Vector3f::Zero();
112         Vector3f spherePoint = rayOrigin + (smallest_t0 * rayDirection);
113         Vector3f normalLightDirection;
114         for (int light = 0; light < lightPositions.size(); light++) {
115             for (int j = 0; j < spheres.size(); j++) {
116                 normalLightDirection = lightPositions[light] - spherePoint;
117                 normalLightDirection.normalize();
118                 if (spheres[j].intersect(spherePoint, normalLightDirection, lt0, lt1)) {
119                     eclipse = true;
120                 }
121             }
122
123             if (eclipse) {
124                 eclipse = false;
125             }
126             else {
127                 //shadedPixel += (spheres[nearestSphere].surfaceColor * 0.333); //for out003
128                 Vector3f normal = spherePoint - spheres[nearestSphere].center;
129                 normal.normalize();
130                 shadedPixel += diffuse(normalLightDirection, normal, spheres[nearestSphere].surfaceColor, 1) + phong(normalLightDirection, normal, -rayDirection, spheres[nearestSphere].su
131                 eclipse = false;
132             }
133         }
134         pixelColor = shadedPixel;
```

Photo above is the body of my trace function. The first for loop looks for the sphere that is closest to the ray origin. The second nested for loops is the ray to light portion of trace that calculates the light contribution to pixel colors. The long line is the diffuse and Phong color contribution calculation where it just uses the formulas presented in class. Otherwise the code is pretty straightforward, we loop through each light firing rays and then determine if any spheres intersect that ray. Eclipse determines if the ray is eclipsed by another sphere and if so we contribute zero and vice versa.