CS380 Computer Graphics

Assignment #3

20160826 윤건희

1. Colored Illuminance

I updated Light class, BlinnPhongShader, and blinn_phong.frag to give color to a light source.

Since color is composed of Red, Green, and Blue, I created three fields: r, g, and b in Light class. When creating a light source, I initialized the rgb values as shown in the code below:

```
const light0: Light ght();
light0.illuminance = 0.2;
light0.color = [255, 165, 0];
light0.r = 255;
light0.g = 165;
light0.b = 0;
light0.type = LightType.AMBIENT;
this.lights.push(light0);
```

In blinn_phong.js, I passed the rgb values to the Light struct in blinn_phong fragment shader using gl.uniform1i:

```
105             gl.uniform1i(locations.r, light.r);
106             gl.uniform1i(locations.g, light.g);
107             gl.uniform1i(locations.b, light.b);
```

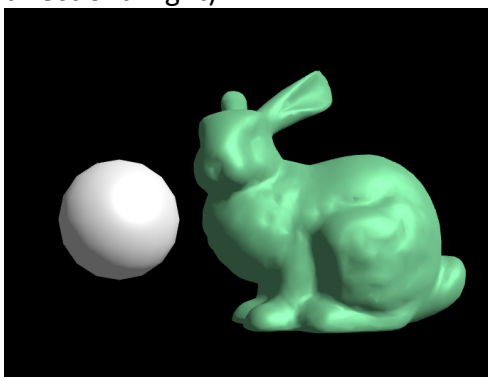
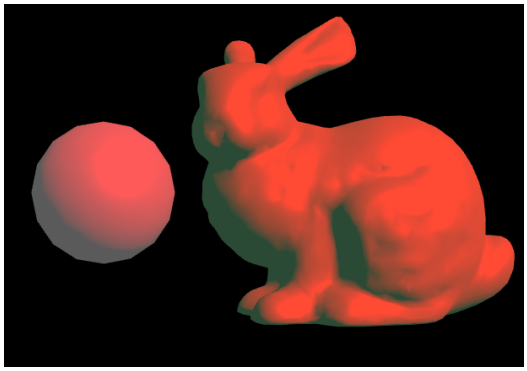
In blinn_phong.frag, I wrote color2float function, which converts integer rgb values to float rgb vec3 ranging from 0 to 1.

```
40  vec3 color2float(int r1, int g1, int b1) {
41      float r2 = float(r1);
42      float g2 = float(g1);
43      float b2 = float(b1);
44      float r = r2 / 255.0;
45      float g = g2 / 255.0;
46      float b = b2 / 255.0;
47      return vec3(r, g, b);
48  }
```

Instead of using mainColor, the fragment shader now uses color2float() to compute output color:

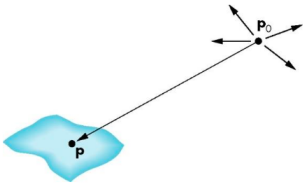
```
89          //diffuse
90          float NdotL = dot(N, L);
91          float diffuse = max(NdotL * lights[i].illuminance, 0.0f);
92          intensity += color2float(lights[i].r, lights[i].g, lights[i].b) * diffuse * point;
```

The lights now have a color! (I am using lab 6 scene for simplicity; the scene is lit by a red directional light):

2. Point Light, and Spotlight Sources

- **Point Light**



$$I(P) \propto \frac{I(P_0)}{|P - P_0|^2}$$

Using the formula given in the textbook, I figured that the light direction vector is the difference between light position and fragment position. L vector should be normalized vector of the light direction. Also, the light intensity must be inversely proportional to the distance between frag_pos and the light position.

L vector:

```
80          vec4 lit = -frag_pos + W2C * vec4(lights[i].pos, 1);
81          L = normalize((lit).xyz);
```

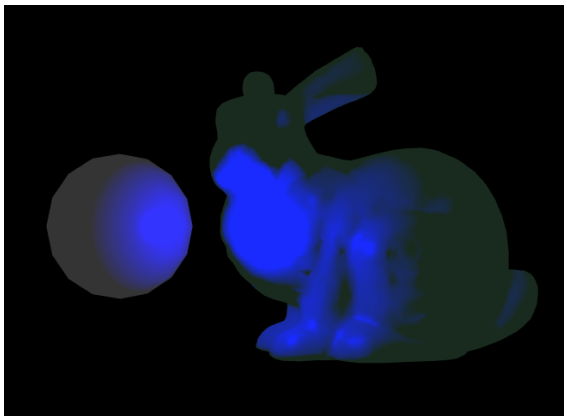
I declared “float point” to store how much the intensity should be attenuated:

```
83          //point light strength
84          float point = 0.0;
85          vec3 distanceVec = (lit).xyz;
86          float d = dot(distanceVec, distanceVec);
87          point = 1.0 / d;
```

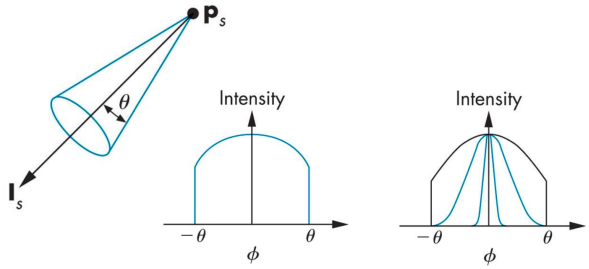
The “float point” variable is multiplied to the intensity to reduce the strength of the light by the distance:

```
91          float diffuse = max(NdotL * lights[i].illuminance, 0.0f);
92          intensity += color2float(lights[i].r, lights[i].g, lights[i].b) * diffuse * point
```

The result:



- **Spotlight**



$$I \propto \cos^e \phi$$

I figured that a spotlight has both direction and position. First thing to do is to calculate the vector from the light source to the fragment position. The angle between the calculated vector and the light direction should be less than theta, otherwise, the light intensity is 0.

The process of getting the L vector is the same as that of Point Light.

In order to calculate the angle, I used the concept that dot product of two unit vectors is the cosine value of the angle between the two vectors:

```
106             vec4 lightDir = vec4(lights[i].dir, 0);
107             vec4 lit = -frag_pos + vec4(lightPos, 1);
108             L = normalize((lit).xyz);
109             vec3 D = normalize((W2C * lightDir).xyz);
110             float angle_diff = abs(dot(L, D)); //cos
```

“angle_diff” is the cosine of the angle between the light direction and the vector from the light source to fragment position.

If “angle_diff” is larger than the cosine of the spotlight angle, then there should be light.

I declared “float spot” variable for this purpose. “float spot” is 0 when the angle difference is larger than the given spotlight angle. Otherwise, the light intensity is proportional to cosine of the angle difference to the power of smoothness.

```
113         if(angle_diff >= angle) {
114             spot = pow(angle_diff, lights[i].angleSmoothness);
115             //spot = 1.0;
116         }
```

Result:

3. Materials

I copied all the files related to BlinnPhongShader and created MyMaterialShader. The created files are: my_material.js, my_material.vert, my_material.frag.

My plan was to have separate material coefficients for each rgb values. These values are passed as arguments to the MyMaterialShader constructor:

```
49  //my material shader1
50  const materialAmbient1 = [1.0, 1.0, 1.0];
51  const materialDiffuse1 = [1.0, 1.0, 1.0];
52  const materialSpecular1 = [1.0, 1.0, 1.0];
53  const shine1 = 100.0;
54  const myMaterialShader1 = await cs380.buildShader(
55    MyMaterialShader,
56    ...materialAmbient1,
57    ...materialDiffuse1,
58    ...materialSpecular1,
59    shine1
60  );
```

These coefficients were passed to the fragment shader as uniforms.

```
71      // Set shader-specific uniforms here
72      this.setUniformVec3(kv, "mainColor", 1, 1, 1);
73      this.setUniformFloat(kv, "ambientR", this.ambientR);
74      this.setUniformFloat(kv, "diffuseR", this.diffuseR);
75      this.setUniformFloat(kv, "specularR", this.specularR);
76      this.setUniformFloat(kv, "ambientG", this.ambientG);
77      this.setUniformFloat(kv, "diffuseG", this.diffuseG);
78      this.setUniformFloat(kv, "specularG", this.specularG);
79      this.setUniformFloat(kv, "ambientB", this.ambientB);
80      this.setUniformFloat(kv, "diffuseB", this.diffuseB);
81      this.setUniformFloat(kv, "specularB", this.specularB);
82      this.setUniformFloat(kv, "shininess", this.shininess);
```

In my_material.frag, I declared a new function “color2float1” that multiplies these coefficients to corresponding rgb values:

```
54  vec3 color2float1(int r1, float rc, int g1, float gc, int b1, float bc) {
55      float r2 = float(r1);
56      float g2 = float(g1);
57      float b2 = float(b1);
58      float r = r2 * rc / 255.0;
59      float g = g2 * gc / 255.0;
60      float b = b2 * bc / 255.0;
61      return vec3(r, g, b);
62  }
```

Instead of using the original “color2float” function, in my_material.frag, this new “colro2float1” function is used like this:

```
//diffuse
float NdotL = dot(N, L);
float diffuse = max(NdotL * lights[i].illuminance, 0.0f);
intensity += color2float1(lights[i].r, diffuseR, lights[i].g, diffuseG, lights[i].b, diffuseB) * diffuse * point;
```

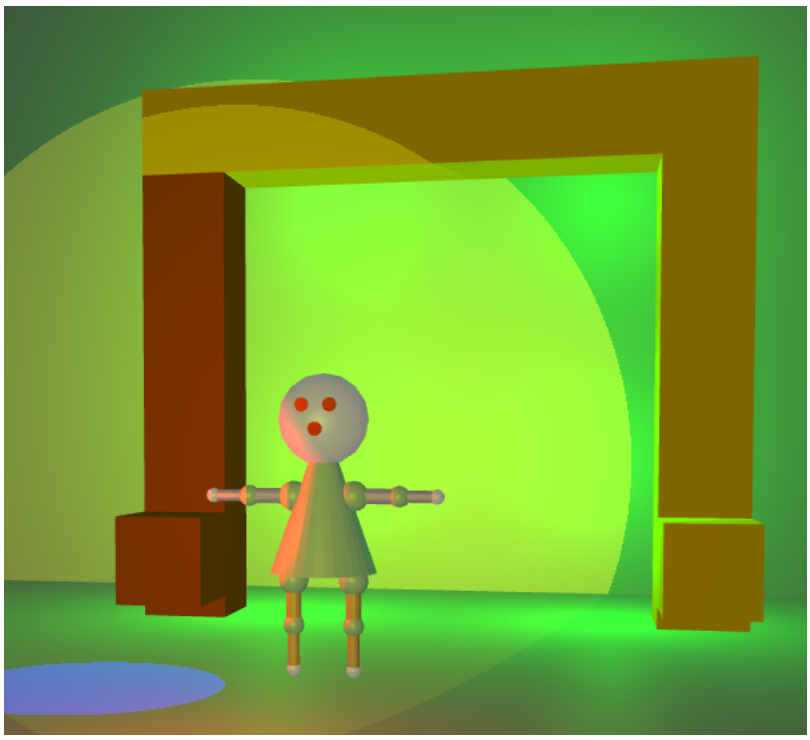
In assignment 3 scene, I rendered pillars in the background with different coefficients.

The pillar on the right has all the coefficients set to 1, so it is no different to blinn_phong shader. The pillar on the left has these coefficients:

```
63      //my material shader2
64      const materialAmbient2 = [0.2, 0.2, 0.2];
65      const materialDiffuse2 = [0.3, 0.0, 1.0];
66      const materialSpecular2 = [0.7, 0.0, 0.7];
67      const shine2 = 10.0;
```

Because the coefficients linked to Green are set to 0, the pillar is not reflecting the green light in the background. Also, yellow ambient light reflected off the pillar is brown, because the coefficients reduce each color by 0.2.

Result:

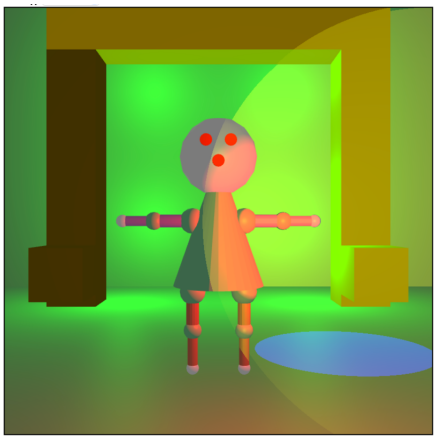


4. Creativity

My scene has one blue spotlight circling around the character, one red point light circling around the character, one yellow spotlight rotating in the middle of the room, shining the walls, and four green point lights illuminating the background.

The character is made of BlinnPhongShader, and the pillars in the background are made of MyMaterialShader.

I tried to recreate a simple version of Dark Portal from World of Warcraft.



Doesn't look as intimidating as the reference… I tried adding more lights, but my computer started to lag.