Computer Graphics Assignment #2

I. Implementation of key mapping

- Z/X: switch the model

  Same as Assignment #1
- T: switch to translation mode

  Same as Assignment #1
- S: switch to scale mode

  Same as Assignment #1
- R: switch to rotation mode

  Same as Assignment #1
- L: switch between directional/point/spot light

```
// switch between directional/point/spot light
else if(key == GLFW_KEY_L && action == GLFW_PRESS){
    // hw2
    if(light_idx ==2){
        light_idx = 0;
    } else {
        light_idx ++;
    }
}
```

- K: switch to light editing mode

```
// switch to light editing mode
else if(key == GLFW_KEY_K && action == GLFW_PRESS){
    cur_trans_mode = Light_ED;
}
```

- J: switch to shininess editing mode

```
// switch to shininess editing mode
else if(key == GLFW_KEY_J && action == GLFW_PRESS){
    cur_trans_mode = Shininess;
}
```

II. Implementation and explanation

- setupRC()
  - setShaders() → create vertex shader & fragment shader → compile vertex shader → compile fragment shader → create program → attach shader to program → link program → pass some variables to shader → initParameter() → LoadModels()
  - Lighting Equation: Intensity = Ambient + Diffuse + Specular

$$I = I_a k_a + \sum_{p=1}^{m} f_p I_p \left(k_d (N \cdot L_p) + k_s (R_p \cdot V)^n \right)$$

- Vertex Shader
  - GLSL use in to linkage into shader from previous stage, out to linkage out of a shader to next stage.

```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec3 aNormal;

out vec3 vertex_color;
out vec3 vertex_normal;
out vec3 fragment_pos; // to fragment shader
```

  - Uniform is a global shader variable.

```
uniform mat4 mvp;
uniform mat4 trans;
uniform vec3 view_pos;
```

  - **Vertex Lighting**

    All the lighting calculations are performed in **vertex shader**.

```
void main()
{
    // [TODO]
    // gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);
    gl_Position = mvp * vec4(aPos.x, aPos.y, aPos.z, 1.0);
    fragment_pos = vec3(trans * vec4(aPos, 1.0f));

    //vertex_normal = aNormal;
    vertex_normal = mat3(transpose(inverse(trans))) * aNormal;

    vec3 N = normalize(aNormal);
    vec3 V = view_pos;

    if (lightIdx == 0) { // directional
        vertex_color = directionalLight(N, V);
    } else if (lightIdx == 1) { // point
        vertex_color = pointLight(N, V);
    } else if (lightIdx == 2) { // spot
        vertex_color = spotLight(N ,V);
    }
}
```

```
vec3 spotLight(vec3 N, vec3 V){
    vec3 light_spot_pos = vec3(0, 0, 2);
    vec3 light_spot_dir = vec3(0, 0, -1);
    float exponent = 50;
    float cutoff = 0.524; // 30 degree

    float ambientstrength = 0.15f;
    vec3 La = ambientstrength * vec3(1.0,1.0,1.0);
    vec3 Ld = diffusestrength * vec3(1.0f,1.0f,1.0f);
    vec3 Ls = vec3(1.0f,1.0f,1.0f);

    vec4 lightInView = um4v * vec4(light_spot_position, 1.0);
    vec3 S = normalize(lightInView.xyz + V);
    vec3 H = normalize(S + V);
    float dis = length(lightInView.xyz + V);
    float spot = dot(S, -light_spot_dir);
    float coscutoff = cos(cutoff);
    float tmp = 0.0;
    if (spot < coscutoff) tmp=0.0;
    else tmp = pow(spot,exponent);
    float atten = tmp/(0.05 + (0.3 * dis) + (0.6 * dis * dis));
    float dc = dot(N, S);
    float sc = pow(max(dot(N, H), 0), 64);
    return La * Ka + ( dc * Ld * Kd + sc * Ls * Ks) * atten;
}
```

```
vec3 pointLight(vec3 N, vec3 V){
    float ambientstrength = 0.15f;
    vec3 La = ambientstrength * vec3(1.0,1.0,1.0);
    vec3 Ld = diffusestrength * vec3(1.0f,1.0f,1.0f);
    vec3 Ls = vec3(1.0f,1.0f,1.0f);

    vec4 lightInView = um4v * vec4(light_point_position, 1.0);
    vec3 S = normalize(lightInView.xyz + V);
    vec3 H = normalize(S + V);
    float dis = length(lightInView.xyz + V);
    float atten = 1.0/(0.01 + (0.8 * dis) + (0.1 * dis * dis));
    float dc = dot(N,S);
    float sc = pow(max(dot(N, H), 0), 64);

    return La * Ka + ( dc * Ld * Kd + sc * Ls * Ks) * atten;
}
```

```
vec3 directionalLight(vec3 N, vec3 V){
    float ambientstrength = 0.15f;
    vec3 La = ambientstrength * vec3(1.0,1.0,1.0);
    vec3 Ld = diffusestrength * vec3(1.0f,1.0f,1.0f);
    vec3 Ls = vec3(1.0f,1.0f,1.0f);

    vec4 lightInView = um4v * vec4(light_dir_position, 1.0);
    vec3 S = normalize(lightInView.xyz + V);
    vec3 H = normalize(S + V);
    float dc = dot(N,S);
    float sc = pow(max(dot(N, H), 0.0), 64);

    return La * Ka + dc * Ld * Kd + sc * Ls * Ks;
}
```

- Fragment Shader
  - GLSL use in to linkage into shader from previous stage, out to linkage out of a shader to next stage.

```
in vec3 vertex_color;
in vec3 vertex_normal;
in vec3 fragment_pos; // from vertex shader

out vec4 FragColor;
```

- Uniform is a global shader variable.

```
uniform vec3 light_dir_position;
uniform vec3 light_point_position;
uniform vec3 light_spot_position;
uniform vec3 view_pos;
uniform int lightIdx;
uniform int shininess;
uniform float diffusestrength;
uniform vec3 Ka;
uniform vec3 Kd;
uniform vec3 Ks;
uniform int vertex_or_perpixel;
```

- **Per-pixel Lighting**

  All the lighting calculations are performed in **fragment shader**.

```
void main() {
    // [TODO]
    // FragColor = vec4(vertex_normal, 1.0f);
    float attenuation = 1;
    float d;

    //Ambient
    float ambientstrength = 0.15f;
    vec3 La = ambientstrength * vec3(1.0,1.0,1.0);

    //Diffuse
    vec3 Ld = diffusestrength * vec3(1.0f,1.0f,1.0f);
    vec3 norm = normalize(vertex_normal);
    vec3 lightDir = normalize(light_dir_position);
    vec3 reflectDir = normalize(reflect(-1*light_dir_position,norm));

    if (lightIdx == 0) { // directional
        lightDir = normalize(light_dir_position);
        attenuation = 1;
        reflectDir = normalize(reflect(-1 * light_dir_position,norm));
```

```
} else if (lightIdx == 1) { // point
    lightDir = normalize(light_point_position - fragment_pos);
    d = length(light_point_position - fragment_pos);
    attenuation = min(1 / (0.01 + 0.8 * d + 0.1 * d * d), 1);
    reflectDir = normalize(reflect( -1 * light_point_position, norm));
} else { // spot
    lightDir = normalize(light_spot_position - fragment_pos);
    d = length(light_spot_position - fragment_pos);
    attenuation = min(1 / (0.05 + 0.3 * d + 0.6 * d * d), 1);
    reflectDir = normalize(reflect(-1 * light_spot_position, norm));
}
float diffuse = max(dot(norm, lightDir), 0.0);

//Specular
vec3 Ls = vec3(1.0f,1.0f,1.0f);
vec3 view_dir = normalize(view_pos - fragment_pos);
float spec = pow(max(dot(view_dir, reflectDir),0.0),shininess);

vec3 result = (La * Ka + Ld * Kd * diffuse + Ls * Ks * spec) * attenuation;
vec4 fragColor = vec4(result, 1.0);
```

- Use constant vertex_or_perpixel to decide which mode

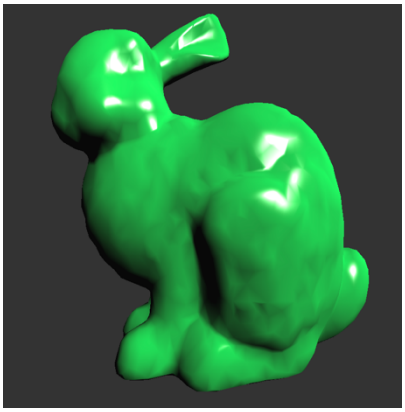
```
// vertex or perpixel
if (vertex_or_perpixel == 0) {
    FragColor = vec4(vertex_color, 1.0);
} else {
    FragColor = fragColor;
}
```

- RenderScene()

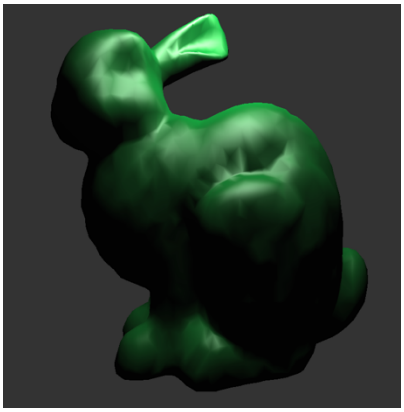
  Use glViewport to set two models side-by-side

III. Some screen shot

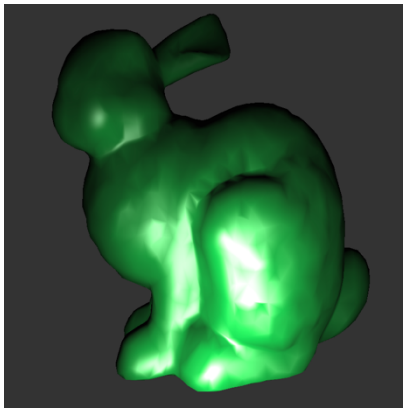
directional light



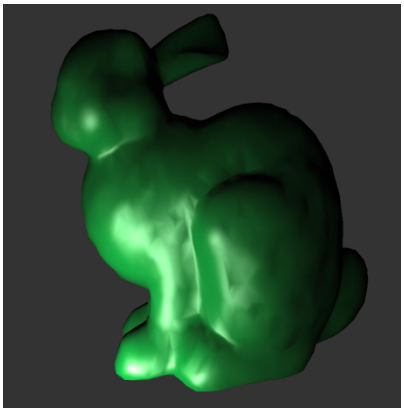
positional light

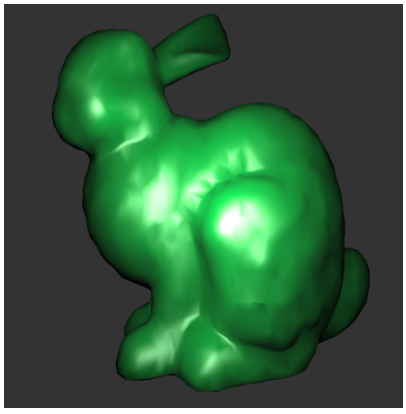


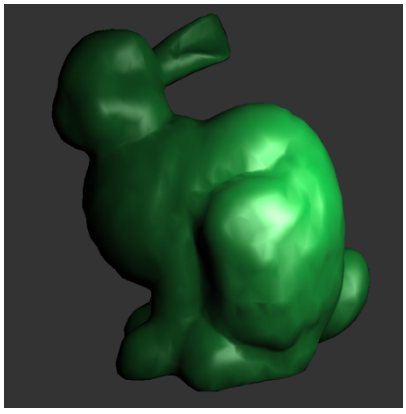
spot light



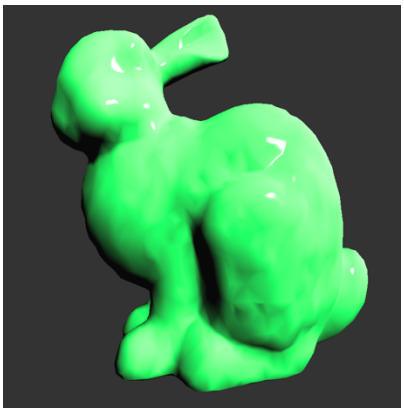
light editing mode

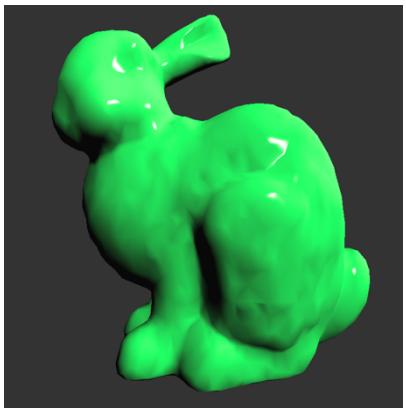


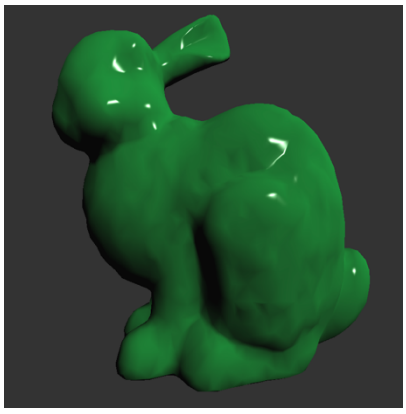




shininess editing mode







vertex lighting and per-pixel lighting