Computer Graphics Assignment #3

Create cameras from 3 directions & Use Blinn-Phong shading

202021511 소프트웨어학과 조민재

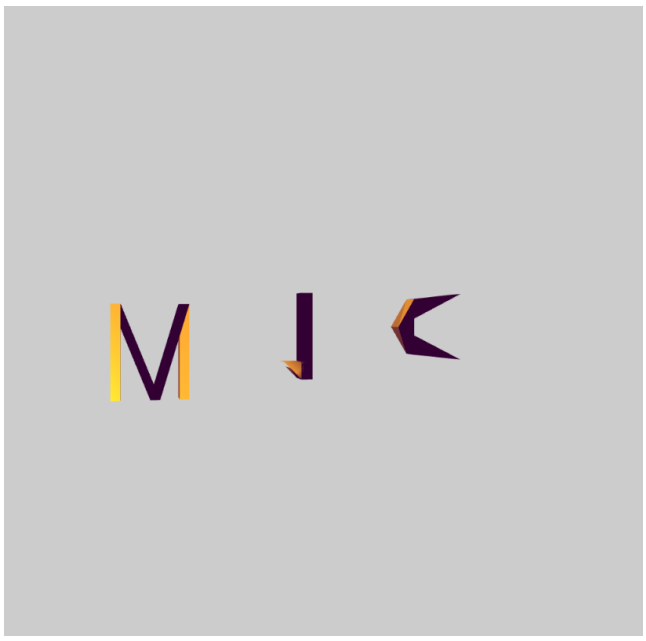
Create cameras from 3 directions (front, side, and top)

I implemented the lookat function to allow viewing from the front (default), left, right, and top perspectives by pressing each button.

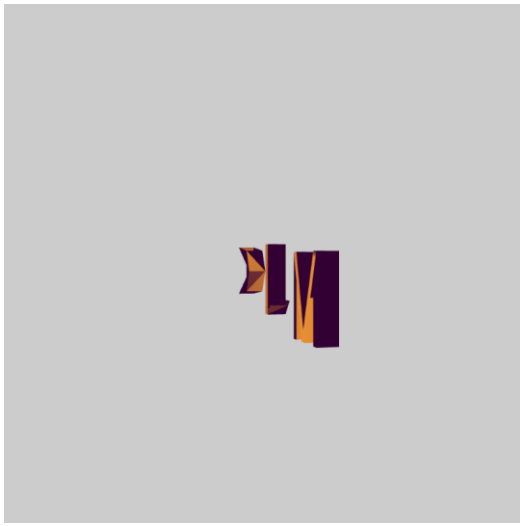
```
document.getElementById("viewFromFront").onclick = function (e) {
  cameraPos = [0, 0, 3.5];
  viewMatrix = lookAt(cameraPos, targetPos, upVector);
};
document.getElementById("viewFromRightSide").onclick = function (e) {
  cameraPos = [3.5, 0, 0];
  viewMatrix = lookAt(cameraPos, targetPos, upVector);
};
document.getElementById("viewFromLeftSide").onclick = function (e) {
  cameraPos = [-3.5, 0, 0];
  viewMatrix = lookAt(cameraPos, targetPos, upVector);
};
document.getElementById("viewFromUp").onclick = function (e) {
  cameraPos = [0, 5.5, 0.001];
  viewMatrix = lookAt(cameraPos, targetPos, upVector);
};
```

For the front view, the camera was positioned at x:0, y:0, z:3.5, and the other views were similarly configured. However, the top view was set with a very small z value of 0.001, because setting it to 0 caused an error. This occurred because the cross product result was [0,0,0] and trying to normalize this resulted in a NaN error, so I used a small value instead.

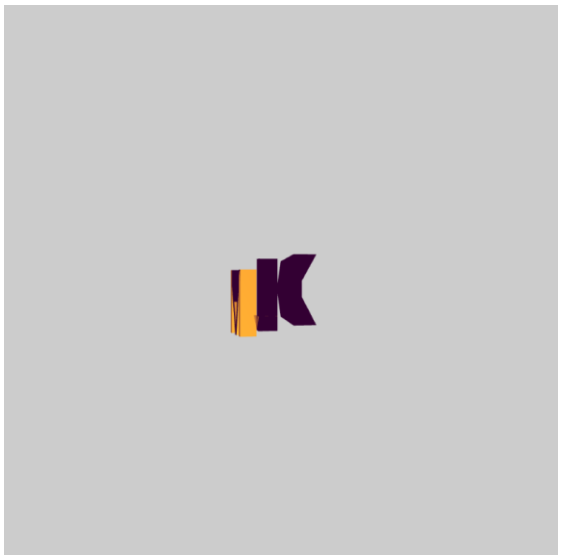
- front



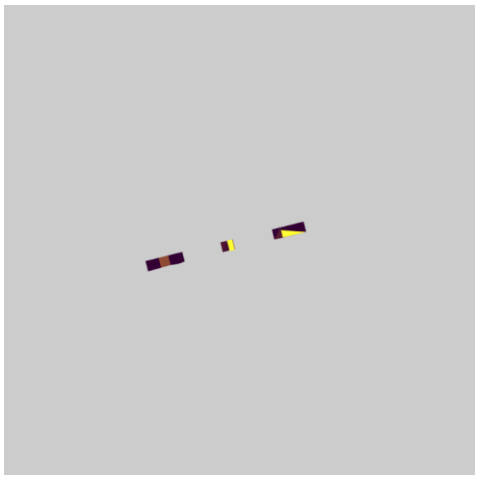
- left side



- right side



- top side



Use Blinn-Phong shading to determine the colors of the model.

To represent triangular plane primitives (each triangle having 3 vertices), I referenced the quad(a,b,c,d) function provided in the lecture materials and created the following functions.

```
function triangle(a, b, c) {
  var t1 = subtract(vertices[b], vertices[a]);
  var t2 = subtract(vertices[c], vertices[a]);

  var normal = cross(t1, t2);
  var normal = vec3(normal);
  normal = normalize(normal);

  normalsArray.push(normal);
  normalsArray.push(normal);
  normalsArray.push(normal);
}

function computeNormalsForTriangleFan(startOfFANVertices, endOfFANVertices) {
  for (let i = 1; i < endOfFANVertices - 1; i++) {
    const vec1 = subtract(vertices[i], vertices[startOfFANVertices]);
    const vec2 = subtract(vertices[i + 1], vertices[startOfFANVertices]);

    let normal = cross(vec1, vec2);
    normal = normalize(normal);

    normalsArray.push(normal);
    normalsArray.push(normal);
    normalsArray.push(normal);
  }
}
```

Using these two functions, I was able to create normal vectors for surfaces formed with gl.TRIANGLES and gl.TRIANGLE_FAN.

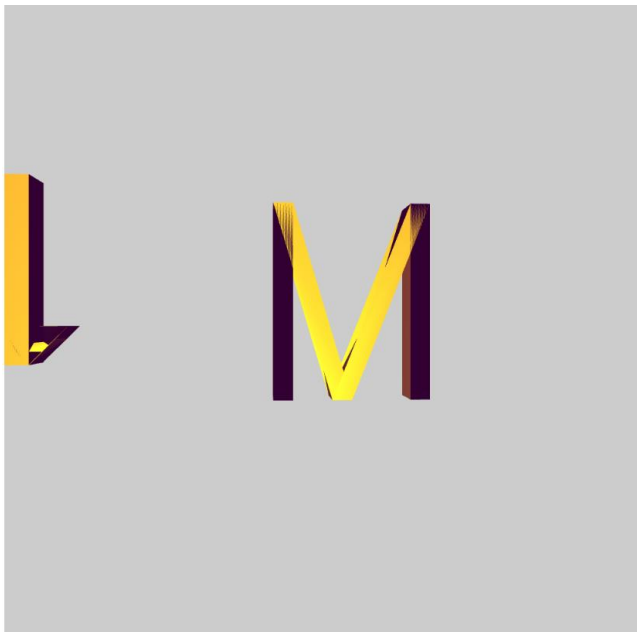
```
  <script id="vertex-shader" type="x-shader/x-vertex">
      attribute vec4 vPosition;
      attribute vec3 vNormal;
      varying vec4 fColor;

      uniform vec4 ambientProduct, diffuseProduct, specularProduct;

      uniform mat4 modelMatrix;
      uniform mat4 viewMatrix;
      uniform mat4 translationMatrix;
      uniform mat4 initialScaleMatrix;
      uniform mat4 projectionMatrix;
      uniform vec4 lightPosition;
      uniform float shininess;

      mat4 modelViewMatrix = viewMatrix * translationMatrix * initialScaleMatrix * modelMatrix;

      void main()
      {
          vec3 pos = -(modelViewMatrix * vPosition).xyz;
          vec3 light = lightPosition.xyz;
          vec3 L = normalize( light - pos );
          vec3 E = normalize( -pos );
          vec3 H = normalize( L + E );
          vec4 NN = vec4(vNormal,0);

          vec3 N = normalize( (modelViewMatrix*NN).xyz);
          vec4 ambient = ambientProduct;
          float d_val = max( dot(L, N), 0.0 );
          vec4 diffuse = d_val *diffuseProduct;
          float s_val = pow( max(dot(N, H), 0.0), shininess );
          vec4 specular = s_val * specularProduct;

          if( dot(L, N) < 0.0 ) {
              specular = vec4(0.0, 0.0, 0.0, 1.0);
          }
          gl_Position = projectionMatrix * modelViewMatrix * vPosition;
          fColor = ambient + diffuse + specular;
```

```
        fColor.a = 1.0;
    }
</script>
```

In the vertex shader, spatial transformations and lighting calculations were performed for each vertex,

```
<script id="fragment-shader" type="x-shader/x-fragment">
    precision mediump float;
    varying vec4 fColor;
    void main()
    {
        gl_FragColor = fColor;
    }
</script>
```

and the results were passed to the fragment shader to determine the colors at the pixel level.



Thus, the light was reflected to produce colors, but something seems off. The initials reflect light and sparkle, but other primitive surfaces facing the same direction do not sparkle. I think this might be due to not creating the 3D initials perfectly and possibly the positioning of the lights, or there could be an issue with how I created the function for generating normal vectors.