1 – No light → Ambient

Add uniform ambient light variable (applied to all pixels) to ***vertex*** shader

Modulate colour (add, multiply, ypur choice)

```
<script id="vertex" type="x-shader">
  attribute vec3 aVertexPosition;
  attribute vec4 aVertexColor;
  attribute vec2 aTextureCoord;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;
  uniform vec3 uAmbientColor;

  varying vec4 vColor;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    vColor = aVertexColor * vec4(uAmbientColor, 1.0);
  }
</script>
```

Get location of ambient light uniform

```
function initShaderProgram() {
  //
  shaderProgram.ambientColorUniform = gl.getUniformLocation(shaderProgram, "uAmbientColor");
  //
```

Set ambient light colour

```
function drawAll() {
  gl.uniform3f(
            shaderProgram.ambientColorUniform,
            ambientLightColour[0],
            ambientLightColour[1],
            ambientLightColour[2]
        );
```

2a – Ambient → Add Directional - vertex

Add directional light uniforms and attributes to ***vertex*** shader

```
<script id="vertex" type="x-shader">
  //
  attribute vec3 aVertexNormal;
  uniform mat3 uNMatrix;
  uniform vec3 uLightingDirection;
  uniform vec3 uDirectionalColor;
  varying vec3 vLightWeighting;

  varying vec4 vColor;
```

Determine light weighting and set colour

```
  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);

    vec3 transformedNormal = normalize(uNMatrix * aVertexNormal);
    float directionalLightWeighting = max(dot(transformedNormal, uLightingDirection), 0.0);
    vLightWeighting = uAmbientColor + uDirectionalColor * directionalLightWeighting;

    vColor = aVertexColor;
  }
</script>
```

Apply light weighting In ***fragment*** shader

```
<script id="fragment" type="x-shader">
  precision highp float;
  varying vec4 vColor;
  varying vec3 vLightWeighting;

  void main() {
    gl_FragColor = vColor * vec4(vLightWeighting, 1.0);
  }
</script>
```

Get location of direction light uniforms and attributes

```
function initShaderProgram() {
  //
  shaderProgram.ambientColorUniform = gl.getUniformLocation(shaderProgram, "uAmbientColor");

  shaderProgram.vertexNormalAttribute = gl.getAttribLocation(shaderProgram, "aVertexNormal");
  gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

  shaderProgram.nMatrixUniform = gl.getUniformLocation(shaderProgram, "uNMatrix");
  shaderProgram.lightingDirectionUniform = gl.getUniformLocation(shaderProgram, "uLightingDirection");
  shaderProgram.directionalColorUniform = gl.getUniformLocation(shaderProgram, "uDirectionalColor");

  shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
  gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
  //
}
```

Set direction light uniforms and attributes

```
function drawAll() {
  gl.uniform3f(
            shaderProgram.ambientColorUniform,
            ambientLightColour[0],
            ambientLightColour[1],
            ambientLightColour[2]
        );

  mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
  mat4.identity(mvMatrix);
  mat4.translate(mvMatrix, [0.0, -1.0, depth]);
  mat4.multiply(mvMatrix, scene.rotation);

  var lightingDirection = [
      scene.models["light"].rotation[12],
      scene.models["light"].rotation[13],
      scene.models["light"].rotation[14]
  ];
  mat4.multiplyVec3(scene.rotation, lightingDirection, lightingDirection);
  var adjustedLD = vec3.create();
  vec3.normalize(lightingDirection, adjustedLD);
  gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);

  gl.uniform3f(
      shaderProgram.directionalColorUniform,
            directionalLightColour[0],
            directionalLightColour[1],
            directionalLightColour[2]
  );

  gl.clearColor(0, 0.5, 0, 1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  for (var key in scene.models) {
    mvPushMatrix();

    model = scene.models[key];

    mat4.translate(mvMatrix, model.position);
    mat4.multiply(mvMatrix, model.rotation);
    mat4.scale(mvMatrix, model.scale);

    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
    gl.bindBuffer(gl.ARRAY_BUFFER, model.vertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, model.vertexPositionBuffer.itemSize,
gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, model.vertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, model.vertexColorBuffer.itemSize,
gl.FLOAT, false, 0, 0);

    var normalMatrix = mat3.create();
    mat4.toInverseMat3(mvMatrix, normalMatrix);
    mat3.transpose(normalMatrix);
    gl.uniformMatrix3fv(shaderProgram.nMatrixUniform, false, normalMatrix);

    gl.bindBuffer(gl.ARRAY_BUFFER, model.vertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute, model.vertexNormalBuffer.itemSize,
gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, model.vertexIndexBuffer);
    gl.drawElements(gl.TRIANGLES, model.vertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

    mvPopMatrix();
  }
}
```

2b – Directional – vertex → Directional – fragment

Move light weighting calculation to ***fragment*** shader

```
<script id="vertex" type="x-shader">
  attribute vec3 aVertexPosition;
  attribute vec4 aVertexColor;
  attribute vec2 aTextureCoord;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  attribute vec3 aVertexNormal;
  uniform mat3 uNMatrix;
  varying vec3 vTransformedNormal;

  varying vec4 vColor;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);

    vTransformedNormal = normalize(uNMatrix * aVertexNormal);

    vColor = aVertexColor;
  }
</script>
```

```
<script id="fragment" type="x-shader">
  precision highp float;
  varying vec4 vColor;
  varying vec3 vTransformedNormal;
  uniform vec3 uLightingDirection;
  uniform vec3 uDirectionalColor;
  uniform vec3 uAmbientColor;

  void main() {
    float NdotL = max(dot(vTransformedNormal,uLightingDirection),0.0);
    vec4 color = vColor;

    float directionalLightWeighting = max(dot(vTransformedNormal, uLightingDirection), 0.0);
    vec3 vLightWeighting = uAmbientColor + uDirectionalColor * directionalLightWeighting;
    color = vColor * vec4(vLightWeighting, 1.0);

    gl_FragColor = color;
  }
</script>
```