WEBGL

WEBGL是什么

- 网页中渲染三维图形的技术
- 继承自OpenGL ES
- 浏览器内置，无需其他开发环境
- JS + HTML + GLSL ES

如何编写WEBGL程序

- 获取WebGL上下文

  var gl = canvas.getContext('webgl'), 'experimental-webgl'

- 编写着色程序（顶点着色器和片断着色器）
- 传递顶点数据（坐标、颜色、法向量等）给着色程序
- gl.drawArrays() 或 gl.drawElements() 绘制一帧

着色器

- 顶点着色器
  - 计算顶点坐标
  - 传递数据给片断着色器
- 片断着色器
  - 计算像素颜色

```
const VS_SOURCE = `
  attribute vec4 a_position;

  void main() {
    gl_Position = a_position;
  }
`
```

顶点着色器

```
const FS_SOURCE = `
  void main() {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
`
```

片断着色器

GLSL ES
OpenGL ES着色器语言

存储限定词
attribute uniform varying

变量类型　　齐次坐标
float vec2 vec3 vec4 mat3 mat4

内置变量
gl_Position
gl_PointSize
gl_FragColor

着色器编译链接

```
const VS_SOURCE = `
  attribute vec4 a_position;

  void main() {
    gl_Position = a_position;
  }
`
const vShader = gl.createShader(gl.VERTEX_SHADER)
gl.shaderSource(vShader, VS_SOURCE)
gl.compileShader(vShader)
```

编译顶点着色器

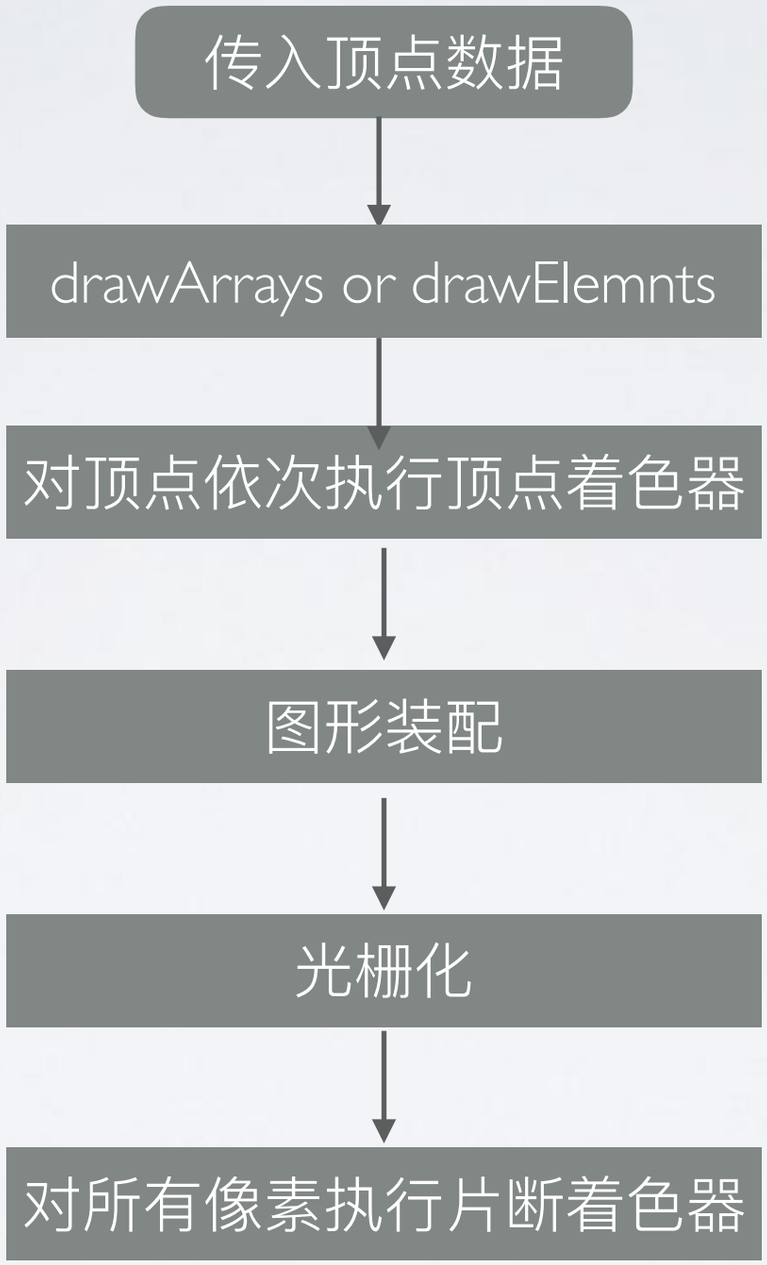
```
const FS_SOURCE = `
  void main() {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
`
const fShader = gl.createShader(gl.FRAGMENT_SHADER)
g.shaderSource(fShader, FS_SOURCE)
gl.compileShader(fShader)
```

编译片断着色器

```
const program = gl.createProgram()
gl.attachShader(program, vShader)
gl.attachShader(program, fShader)
gl.linkProgram(program)

gl.useProgram(program)
```

着色程序

着色器工作流程

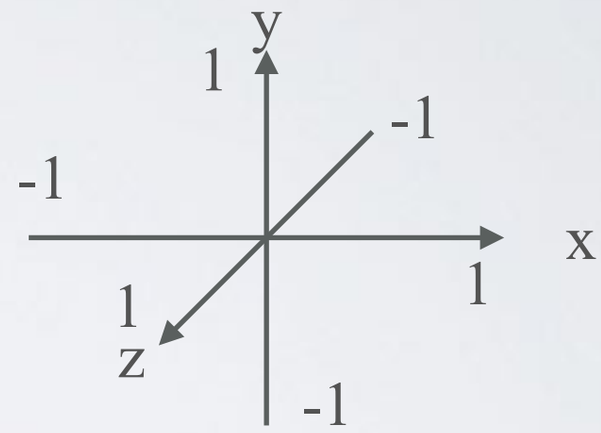

传入顶点数据
drawArrays or drawElemnts
对顶点依次执行顶点着色器
图形装配
光栅化
对所有像素执行片断着色器

传入顶点数据

```
const points = new Float32Array([
  0.5, 0.5,
  0.5, -0.5,
  -0.5, -0.5,
  -0.5, 0.5
])
```

类型化数组

WebGL坐标系统




```
const buffer = gl.createBuffer()
gl.bindBuffer(gl.ARRAY_BUFFER, buffer)
gl.bufferData(gl.ARRAY_BUFFER, points, gl.STATIC_DRAW)
```

创建缓冲区并写入顶点数据

传入顶点数据

```
const aPosition = gl.getAttribLocation(program, 'a_position')
gl.vertexAttribPointer(aPosition, 2, gl.FLOAT, false, 0, 0)
gl.enableVertexAttribArray(aPosition)
```

获取attribute变量地址
指定缓冲区读取方式
启用变量

```
gl.clearColor(0, 0, 0, 1)
gl.clear(gl.COLOR_BUFFER_BIT)

gl.drawArrays(gl.LINE_LOOP, 0, 4)
```

清空颜色缓冲区
指定绘制方式和绘制次数

```
const VS_SOURCE = `
  attribute vec4 a_position;

  void main() {
    gl_Position = a_position;
  }
`
```

顶点着色器代码

```
const FS_SOURCE = `
  void main() {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
`
```

片断着色器代码

添加颜色

```
const VS_SOURCE = `
  attribute vec4 a_position;
  attribute vec4 a_color;

  varying vec4 v_color;
  void main() {
    gl_Position = a_position;
    v_color = a_color;
  }
`
```

```
const FS_SOURCE = `
  precision mediump float;

  varying vec4 v_color;
  void main() {
    gl_FragColor = v_color;
  }
`
```

varying变量传递插值数据

```
const points = new Float32Array([
  0.5, 0.5, 1, 0, 0,
  0.5, -0.5, 0, 1, 0,
  -0.5, -0.5, 0, 0, 1,
  -0.5, 0.5, 1, 1, 1
])
```

对每个顶点指定颜色

添加颜色

```
const points = new Float32Array([
  0.5, 0.5, 1, 0, 0,
  0.5, -0.5, 0, 1, 0,
  -0.5, -0.5, 0, 0, 1,
  -0.5, 0.5, 1, 1, 1
])
```

```
const elSize = points.BYTES_PER_ELEMENT
// 顶点坐标
const aPosition = gl.getAttribLocation(program, 'a_position')
gl.vertexAttribPointer(aPosition, 2, gl.FLOAT, false, 5 * elSize, 0)
gl.enableVertexAttribArray(aPosition)
// 顶点颜色
const aColor = gl.getAttribLocation(program, 'a_color')
gl.vertexAttribPointer(aColor, 3, gl.FLOAT, false, 5 * elSize, 2 * elSize)
gl.enableVertexAttribArray(aColor)
```

指定attribute读取方式

三维变换

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

```
const matrix = new Float32Array([
  a, e, i, m,
  b, f, j, n,
  c, g, k, o,
  d, h, l, p
])
```

WebGL矩阵按列主序

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

平移

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 & 0 \\ \sin\beta & \cos\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

旋转

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

缩放

三维变换

```
const VS_SOURCE = `
  attribute vec4 a_position;
  attribute vec4 a_color;

  uniform mat4 u_modal;

  varying vec4 v_color;
  void main() {
    gl_Position = u_modal * a_position;
    v_color = a_color;
  }
`
```

```
// 变换矩阵
const uModal = gl.getUniformLocation(program, 'u_modal')

const modalMat = Matrix.rotate(30, 1, 1, 0).transpose()
gl.uniformMatrix4fv(uModal, false, modalMat.m)
```

三维图形

立方体6个面，每个面需4个顶点，共24个顶点

gl.drawElements

存储最少的顶点，利用顶点索引绘制

三维图形

```
const points = new Float32Array([
  0.5, 0.5, 0.5, 1, 0, 0,
  0.5, -0.5, 0.5, 0, 1, 0,
  -0.5, -0.5, 0.5, 0, 0, 1,
  -0.5, 0.5, 0.5, 1, 1, 0,
  -0.5, 0.5, -0.5, 1, 0, 1,
  -0.5, -0.5, -0.5, 0, 1, 1,
  0.5, -0.5, -0.5, 1, 1, 1,
  0.5, 0.5, -0.5, 0, 0, 0
])
```

```
const indices = new Uint8Array([
  0, 1, 2, 0, 2, 3,
  0, 1, 6, 0, 6, 7,
  4, 5, 6, 4, 6, 7,
  3, 2, 5, 3, 5, 4,
  0, 3, 4, 0, 4, 7,
  1, 2, 5, 1, 5, 6
])
```

```
// 顶点数据缓冲区
const buffer = gl.createBuffer()
gl.bindBuffer(gl.ARRAY_BUFFER, buffer)
gl.bufferData(gl.ARRAY_BUFFER, points, gl.STATIC_DRAW)

// 顶点索引缓冲区
const indicesBuffer = gl.createBuffer()
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indicesBuffer)
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW)
```

三维图形

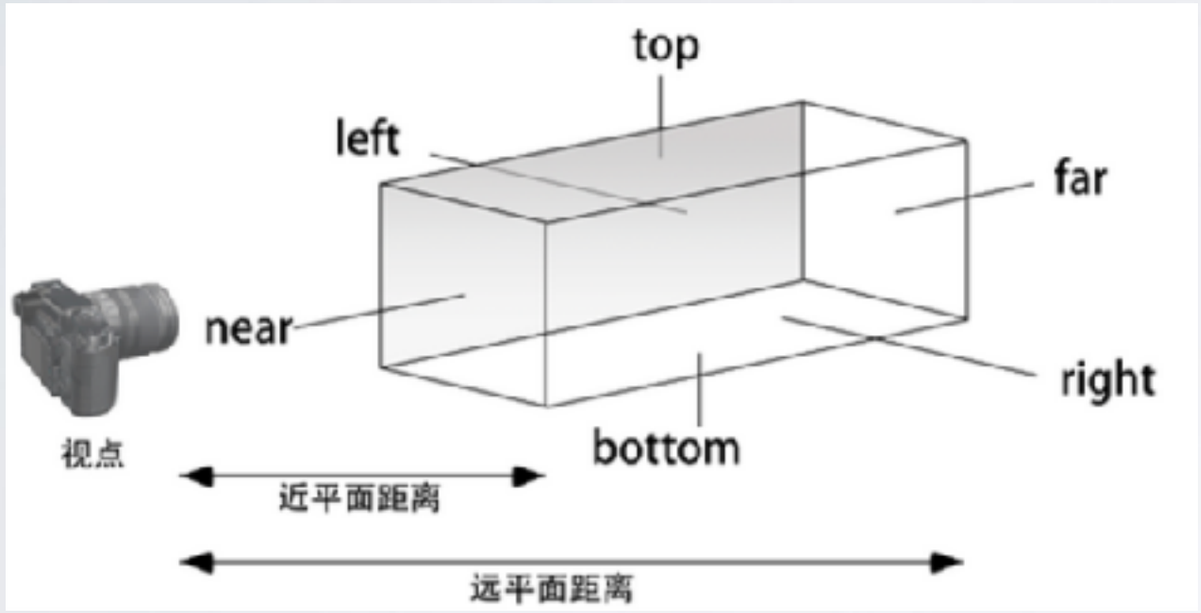
```
gl.clearColor(0.0, 0.0, 0.0, 1.0)

let modalMat = new Matrix()
const tick = () => {
  const am = Matrix.rotate(a, 1, 0, 0)
  const bm = Matrix.rotate(b, 0, 1, 0)

  modalMat = modalMat.multiply(am).multiply(bm)
  gl.uniformMatrix4fv(uModal, false, modalMat.transpose().m)

  gl.clear(gl.COLOR_BUFFER_BIT)
  gl.drawElements(gl.TRIANGLES, 36, gl.UNSIGNED_BYTE, 0)

  window.requestAnimationFrame(tick)
}

tick()
```

视图和投影

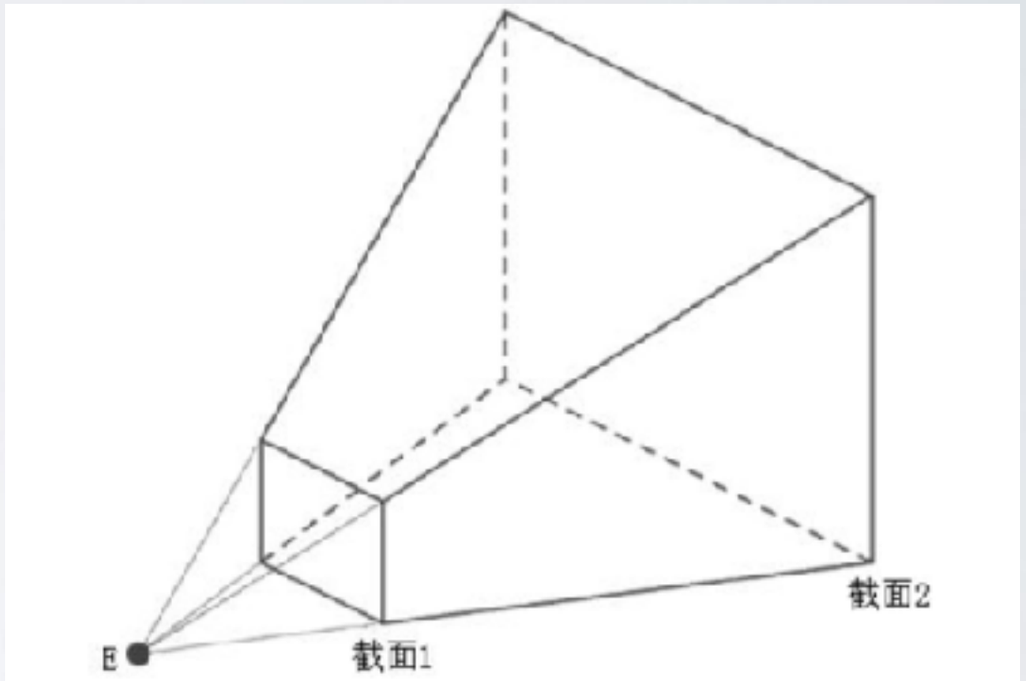
世界坐标系到摄像机坐标系的转换　　视图矩阵

视图模型矩阵 = 视图矩阵 * 模型矩阵

视图和投影



正射投影



透视投影

投影矩阵

视图和投影

```
attribute vec4 a_position;
attribute vec4 a_color;

uniform mat4 u_viewModal;
uniform mat4 u_projection;

varying vec4 v_color;

void main() {
  v_color = a_color;

  gl_Position = u_projection * u_viewModal * a_position;
}
```

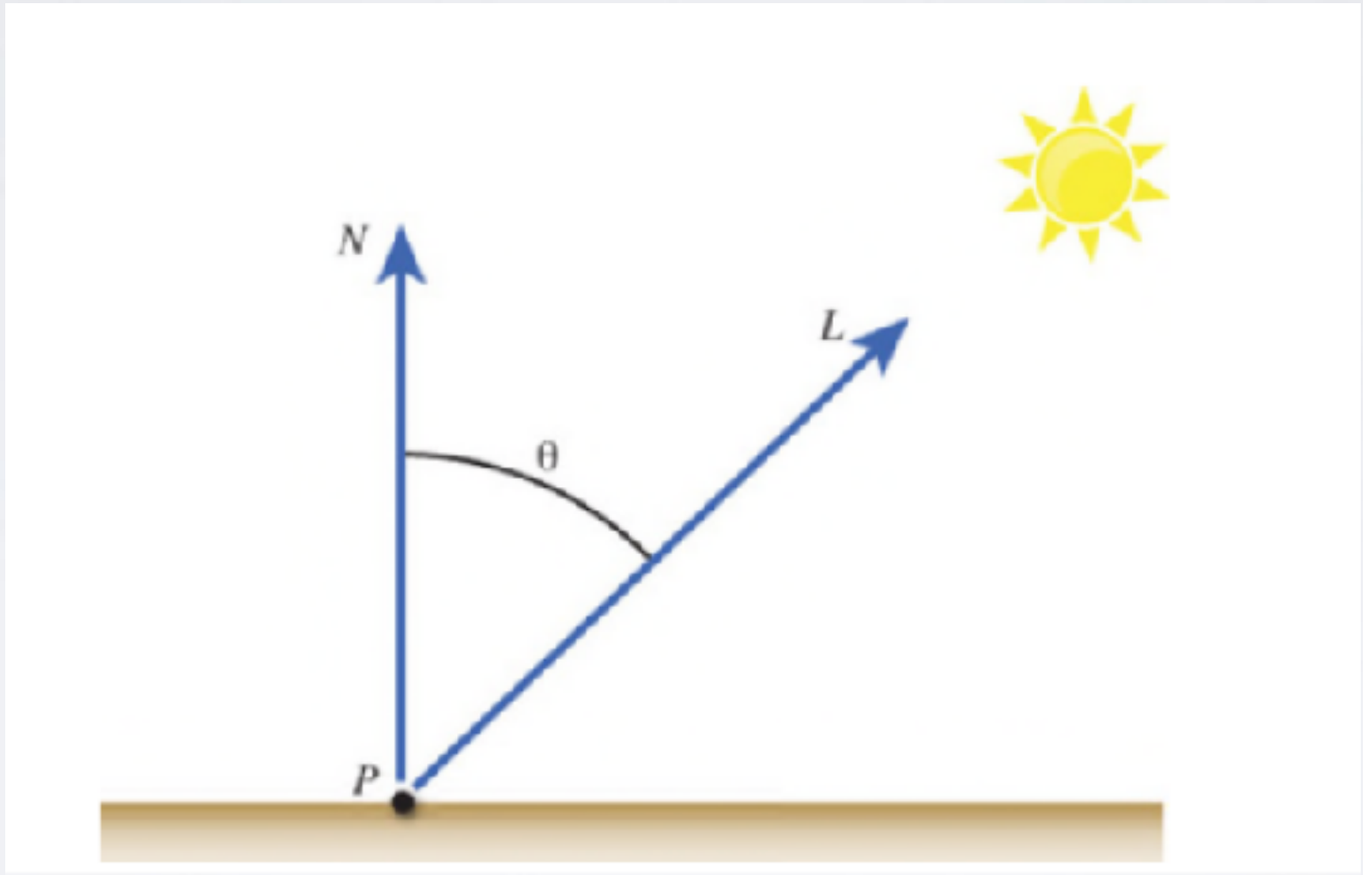
光照模型

- 平行光源,点光源,环境光源

- 漫反射,环境反射

漫反射

点光源和平行光源



漫反射光颜色 = 光源颜色 * 反射面颜色 *$cos\theta$

$cos\theta$ = dot(N, L) / (|N|*|L|)

漫反射

法向量方向随物体运动而变化(平移/缩放除外)

变换后的法向量 = 模型矩阵的逆转矩阵 * 原法向量

环境光反射

环境光反射各向均匀分布,强度相等

环境光反射颜色 = 环境光颜色 * 反射面颜色

光照模型

最终反射光颜色 = 漫反射光颜色 + 环境反射光颜色

```
attribute vec4 a_position;
attribute vec4 a_color;
attribute vec4 a_normal;

uniform mat4 u_modal;
uniform mat4 u_viewModal;
uniform mat4 u_projection;
uniform mat4 u_normalMatrix;

varying vec4 v_color;
varying vec3 v_normal;
varying vec3 v_position;

void main() {
  vec4 vertexPosition = u_modal * a_position;

  v_normal = normalize(vec3(u_normalMatrix * a_normal));
  v_color = a_color;
  v_position = vec3(vertexPosition);

  gl_Position = u_projection * u_viewModal * a_position;
}
```

光照模型

```
precision mediump float;

uniform vec3 u_ambientLightColor;
uniform vec3 u_lightColor;
uniform vec3 u_lightPosition;

varying vec4 v_color;
varying vec3 v_position;
varying vec3 v_normal;

void main() {
  vec3 lightDirection = normalize(u_lightPosition - vec3(v_position));
  float nDotL = max(dot(lightDirection, v_normal), 0.0);

  vec3 ambient = u_ambientLightColor * v_color.rgb;
  vec3 diffuse = u_lightColor * v_color.rgb * nDotL;

  gl_FragColor = vec4(diffuse + ambient, v_color.a);
}
```

其他

- 纹理
- 阴影
- 三维模型建模
- Three.js