基本原理

WebGL 是运行在 GPU 上的，它使用一种叫做 GLSL 的语言来编写着色器程序。着色器程序由两部分组成：顶点着色器（vertex shader）和片段着色器（fragment shader）。顶点着色器负责处理顶点的位置和属性，片段着色器负责处理每个像素的颜色。

几乎所有的 API 都是为这些函数来设置状态，调用 gl.drawArrays 和 drawElements 在GPU上运行你的着色器。

接收数据的四种方式
1、属性、缓冲区 顶点数组

- 缓存区以二进制数据形式的数组传给GPU。缓存区可以放任意数据，通常有位置，归一化参数，纹理坐标，顶点颜色等等
- 属性用来指定数据如何从缓冲区获取并提供给顶点着色器。比如你可能将位置信息以3个32位的浮点数据存在缓存区中， 一个特定的属性包含的信息有：它来自哪个缓存区，它的数据类型(3个32位浮点数据)，在缓存区的起始偏移量，从一个位置到下一个位置有多少个字节等等。
- 缓冲区并非随机访问的，而是将顶点着色器执行指定次数。每次执行时，都会从每个指定的缓冲区中提取下一个值并分配给一个属性。
- 属性的状态收集到一个顶点数组对象（VAO）中，该状态作用在每个缓冲区，以及如何从这些缓冲区中提取数据。

2、uniforms 传递全局变量
Uniforms是在执行着色器程序前设置的全局变量

3、纹理
纹理是能够在着色器程序中随机访问的数组数据。大多数情况下纹理存储图片数据，但它也用于包含颜色以外的数据。

4、Varying
Varyings是一种从点着色器到片段着色器传递数据的方法。根据显示的内容如点，线或三角形， 顶点着色器在Varyings中设置的值，在运行片段着色器的时候会被解析。

基本概念

1、顶点着色器
顶点着色器是 WebGL 中最基本的着色器，它负责处理顶点的位置和属性。顶点着色器接收一个顶点的位置和属性作为输入，然后通过计算生成一个新的顶点位置和属性，并将其传递给片段着色器。

2、片段着色器
片段着色器是 WebGL 中另一个基本的着色器，它负责处理每个像素的颜色。片段着色器接收一个像素的位置和属性作为输入，然后通过计算生成一个新的像素颜色，并将其输出到屏幕上。

3、缓冲区
缓冲区是 WebGL 中用于存储顶点数据、纹理数据等的数据结构。缓冲区可以以二进制数据的形式存储在 GPU 中，以便快速访问和处理。

如何工作

GPU 基本做了两部分事情： 第一部分是处理顶点(数据流)，变成裁剪空间节点；第二部分是基于第一部分的结果绘制像素。

```
gl.drawArrays(gl.TRIANGLES, 0, 9);
// 处理“9个顶点”
// 假设你在画一些三角形，每次 GPU 都会取出 3 个顶点来生成三角形。它指出三角形的 3 个点对应哪些像素，
// 然后这些像素值画出这个三角形，这个过程就叫“像素栅格化”。对于每个像素，都会调用片段着色器。
// 它有一个 vec4 类型的输出变量，它指示绘制像素的颜色是什么。
```

缓冲区是将顶点和将每个顶点数据传给GPU的方法。 gl.createBuffer创建一个缓冲区。 gl.bindBuffer将该缓冲区设置为正在处理的缓冲区。 gl.bufferData将数据复制到当前缓冲区中。gl.vertexAttribPointer告诉 WebGL 如何从缓冲区中获取数据。 gl.enableVertexAttribArray启用顶点属性数组。 gl.drawArrays绘制图形。

着色器和 GLSL 语言

属性的类型
float, vec2, vec3, vec4, mat2, mat3, mat4, int, ivec2, ivec3, ivec4, uint, uvec2, uvec3, uvec4.

uniforms

```
gl.uniform1f (floatUniformLoc, v);                 // for float
gl.uniform1fv(floatUniformLoc, [v]);               // for float or float array
gl.uniform2f (vec2UniformLoc,  v0, v1);            // for vec2
gl.uniform2fv(vec2UniformLoc,  [v0, v1]);          // for vec2 or vec2 array
gl.uniform3f (vec3UniformLoc,  v0, v1, v2);        // for vec3
gl.uniform3fv(vec3UniformLoc,  [v0, v1, v2]);      // for vec3 or vec3 array
gl.uniform4f (vec4UniformLoc,  v0, v1, v2, v4);    // for vec4
gl.uniform4fv(vec4UniformLoc,  [v0, v1, v2, v4]);  // for vec4 or vec4 array

gl.uniformMatrix2fv(mat2UniformLoc, false, [  4x element array ])  // for mat2
or mat2 array
gl.uniformMatrix3fv(mat3UniformLoc, false, [  9x element array ])  // for mat3
or mat3 array
gl.uniformMatrix4fv(mat4UniformLoc, false, [ 16x element array ])  // for mat4
or mat4 array

gl.uniform1i (intUniformLoc,   v);                 // for int
gl.uniform1iv(intUniformLoc, [v]);                 // for int or int array
gl.uniform2i (ivec2UniformLoc, v0, v1);            // for ivec2
gl.uniform2iv(ivec2UniformLoc, [v0, v1]);          // for ivec2 or ivec2 array
gl.uniform3i (ivec3UniformLoc, v0, v1, v2);        // for ivec3
gl.uniform3iv(ivec3UniformLoc, [v0, v1, v2]);      // for ivec3 or ivec3 array
gl.uniform4i (ivec4UniformLoc, v0, v1, v2, v4);    // for ivec4
gl.uniform4iv(ivec4UniformLoc, [v0, v1, v2, v4]);  // for ivec4 or ivec4 array

gl.uniform1u (intUniformLoc,   v);                 // for uint
gl.uniform1uv(intUniformLoc, [v]);                 // for uint or uint array
gl.uniform2u (ivec2UniformLoc, v0, v1);            // for uvec2
gl.uniform2uv(ivec2UniformLoc, [v0, v1]);          // for uvec2 or uvec2 array
gl.uniform3u (ivec3UniformLoc, v0, v1, v2);        // for uvec3
gl.uniform3uv(ivec3UniformLoc, [v0, v1, v2]);      // for uvec3 or uvec3 array
gl.uniform4u (ivec4UniformLoc, v0, v1, v2, v4);    // for uvec4
gl.uniform4uv(ivec4UniformLoc, [v0, v1, v2, v4]);  // for uvec4 or uvec4 array

// for sampler2D, sampler3D, samplerCube, samplerCubeShader, sampler2DShadow,
// sampler2DArray, sampler2DArrayShadow
gl.uniform1i (samplerUniformLoc,   v);
gl.uniform1iv(samplerUniformLoc, [v]);
```

图元装配 & 光栅化

图元装配： 阶段将顶点着色器输出的顶点数据组装成图元（如点、线段、三角形 几何图形的类别由 gl.drawArrays() 函数的第一个参数）。
光栅化阶段将图元转换为像素，并调用片段着色器为每个像素计算颜色。像素的颜色都是通过顶点颜色差值生成的。

深度测试

深度测试用于确定一个像素是否应该被绘制，以及如何绘制。它通过比较像素的深度值与深度缓冲区中的值来确定。如果像素的深度值小于深度缓冲区中的值，则该像素将被绘制，否则将被丢弃。

隐藏面消除

WebGL 为了加速绘制，是按照缓冲区中的顺序来处理，所以需要深度测试和隐藏面消除来保证绘制的正确性。

```
// 开启隐藏面消除功能
gl.enable(gl.DEPTH_TEST);
// 清除深度缓冲区
gl.clear(gl.DEPTH_BUFFER_BIT);
```

深度冲突

深度冲突是指两个或多个物体在同一个位置上被绘制，导致深度缓冲区中的值无法确定哪个物体应该被绘制。这通常发生在物体非常接近时，例如两个平面几乎平行且距离非常近。
为了解决这个问题，可以使用深度缓冲区的精度，或者使用深度偏移来增加物体的深度值，使其在深度缓冲区中具有更高的优先级。

WebGL 高级技术

鼠标控制物体旋转

光照

光照模型： 模拟光线在物体表面上的反射和散射，从而产生逼真的光照效果。光照模型通常包括环境光、漫反射光、镜面反射光等部分。

光照模型分为全局光照和局部光照。全局光照考虑了光源在场景中的位置和方向，以及物体之间的相互遮挡关系。局部光照则只考虑光源在物体表面的直接反射，不考虑其他物体的遮挡。

在WebGL中，可以使用GLSL语言编写光照模型，通过计算每个像素的光照强度，从而实现逼真的光照效果。

1、自发光（emissive），表示该物体自身是否会发出光线
2、环境光（ambient），表示其他所有的间接光照
3、漫反射（diffuse）,表示光线从光源直接照射到物体表面是，该表面会向每个方向散射多少能量。
4、高光反射（specular），表示光源直接照射到物体表面时，该表面会完全镜面反射多少能量。

Lambert 模型

> Lambert = ambient + diffuse

Lambert漫反射模型其实就是在泛光模型的基础之上增加了漫反射项。漫反射便是光从一定角度入射之后从入射点向四面八方反射，且每个不同方向反射的光的强度相等，而产生漫反射的原因是物体表面的粗糙，导致了这种物理现象的发生。
Lambert是只考虑漫反射而不考虑镜面反射的效果，因而对于金属、镜子等需要镜面反射效果的物体就不适应，对于其他大部分物体的漫反射效果都是适用的

Phong模型

> Phong = ambient + diffuse + specular

> Blinn-Phong = 优化性能之后的Phong

因为Phong模型的性能比较低，所以需要进行一定的优化，所谓优化就是近似的模拟，达到差不多的渲染效果，但是算法复杂度会提升很多。Blinn-Phong与风氏模型唯一的区别就是，Blinn-Phong测量的是法线与半程向量之间的夹角，而风氏模型测量的是观察方向与反射向量间的夹角。

```
// 法向量 归一化
vec3 normal = normalize(a_Normal.xyz);
// 计算点乘 cosθ
float nDotL = max(dot(u_LightDirection, normal), 0.0);
// 环境光照 a_Color 物体的颜色
vec3 ambient = u_LightAmbient * a_Color.rgb;
// 漫反射
vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;
// 兰伯特光照
// Idiffuse是漫反射光强，Kd是物体表面的漫反射属性，Id是光强，theta是光的入射角弧度。
lambert = diffuse + ambient;

// 高光
float spec = pow(max(dot(normal, halfwayDir), 0.0), 32.0);

// Phone 光照 Ispecular 镜面反射
PhoneColor = ambient + Idiffuse + Ispecular
```

法线贴图

通过把物体表面法线的方向记录在法线贴图上，在着色时使用这些法线信息来计算每个像素上的光照，这样就能够在不增加顶点数量情况下增加物体表面的凹凸细节。

在法线贴图中使用图片的 RGB 三个通道来表示纹理上每个像素点法线方向的 XYZ 值。由于法线方向的 XYZ 值的取值范围是 -1 到 1，而 RGB 的值的取值范围是 0 到 1，所以把法线方向储存到法线贴图的图片中时先要进行归一化处理。

vec3 rgb_normal = normal * 0.5 + 0.5;

纹理 mipmaps