着色器：仅用 x 和 y 坐标绘制高保真图形

黄京

Nov 23, 2025

1 探索片段着色器的核心思想，用数学公式代替像素画笔

在计算机图形学中，着色器编程代表了一种革命性的思维方式：它允许开发者通过简洁的数学表达式来定义视觉元素，而非依赖预制的纹理或逐像素绘制。本文将引导您深入理解如何利用片段着色器，仅凭归一化的 x 和 y 坐标，创造出从基础几何到复杂特效的高保真图形。读完本文，您将掌握着色器编程的核心概念，并能亲手编写代码生成各类动态图形。

想象一个场景：复杂的渐变、分形图案或动态光晕效果，这些视觉元素并非来自加载的图片，而是由一个简单的数学函数根据每个像素的 x 和 y 坐标实时计算得出。这引出了一个根本性问题：我们习惯于在画布上绘制形状或加载图像，但如果能通过定义而非绘制来创造图形，会带来怎样的可能性？本文旨在解答这一问题，聚焦于片段着色器的应用，帮助您从传统图形处理转向数学驱动的视觉创作。通过本文的学习，您将能够理解着色器编程的思维模式，并独立实现各种图形效果。

2 基础准备：我们的画布与坐标系

在着色器编程中，片段着色器扮演着核心角色。与顶点着色器处理几何顶点不同，片段着色器负责为每个屏幕像素调用一次，决定其最终颜色。这使其成为图形定义的理想工具。为了通用性，我们引入归一化坐标的概念。通过将像素坐标 `fragCoord.xy` 除以画布尺寸 `iResolution.xy`，我们得到归一化坐标 `uv`，其范围在 [0, 1] 之间。这确保了代码在不同分辨率下的适应性。

例如，以下代码演示了如何计算归一化坐标：

```
vec2 uv = fragCoord.xy / iResolution.xy;
```

在这段代码中，`fragCoord.xy` 代表当前像素的坐标，而 `iResolution.xy` 是画布的宽度和高度。除法操作将坐标映射到 [0, 1] 区间，从而创建一个与分辨率无关的画布。为进一步优化，我们可以将坐标中心调整到画布中点，并修正宽高比以防止图形拉伸。进阶代码可能如下：

```
uv -= 0.5;
uv.x *= iResolution.x / iResolution.y;
```

这里，`uv -= 0.5` 将坐标原点移至画布中心，范围变为 [-0.5, 0.5]。接着，通过乘以宽高比，我们确保画布在 x 和 y 方向上比例一致，避免变形。这些步骤为后续图形定义奠定了坚实基础。

3 核心武器：符号距离函数

符号距离函数是着色器图形定义的核心工具。它本质上是一个数学函数 f(point)，返回给定点到目标图形最近边缘的有符号距离。距离的正负值具有明确含义：正值表示点在形状外部，零值表示点在边界上，而负值表示点在形状内部。这类似于一个形状的引力场，负值区域定义了形状本身。

从符号距离函数到可视图形，我们需要借助步进函数。例如，step(edge，x) 函数在 x 小于 edge 时返回 0，否则返回 1，而 smoothstep(edge0，edge1，x) 提供平滑过渡，有助于抗锯齿。以绘制圆形为例，其符号距离函数公式为 length$(uv) - r$，其中 r 是半径。在代码中，我们可以这样实现：

```
float d = length(uv) - radius;
float circle = step(d, 0.0);
```

在这段代码中，length(uv) 计算点到原点的距离，减去半径后得到有符号距离 d。step(d，0.0) 将负值（内部）映射为 1（显示），正值（外部）映射为 0（隐藏），从而生成一个硬边缘圆形。若使用 smoothstep，则可以创建软边缘效果：

```
float circle = smoothstep(0.01, 0.0, d);
```

这里，smoothstep 在距离接近零时平滑插值，减少锯齿现象。通过直接输出 d 作为灰度，我们还可以可视化距离场，直观理解形状的分布。

常见图形的符号距离函数构成了我们的工具箱。例如，圆形的公式为 length$(p) - r$，矩形可使用 box 函数，直线可通过 sdSegment 实现。这些函数允许我们快速构建基础几何，而无需依赖外部资源。

4 融合与组合：图形布尔运算

符号距离函数的强大之处在于其支持布尔运算，从而组合复杂形状。通过数学操作，我们可以实现并集、交集和差集。并集使用 min(d1，d2) 函数，取两个距离场中的较小值，融合形状；交集使用 max(d1，d2)，仅保留两个形状重叠部分；差集则通过 max(d1，-d2) 从第一个形状中减去第二个。

以绘制吃豆人图形为例，我们可以分步实现。首先，定义一个圆形的符号距离函数：

```
float circle_d = length(uv) - radius;
```

接着，定义一个矩形作为嘴巴部分：

```
float mouth_d = sdBox(uv, mouthSize);
```

其中 sdBox 是矩形的距离函数。最后，应用差集运算：

```
float pacman_d = max(circle_d, -mouth_d);
float final_shape = step(pacman_d, 0.0);
```

在这段代码中，max(circle_d，-mouth_d) 实现了从圆形中减去矩形的效果，因为 -mouth_d 将矩形内部变为负值，外部变为正值，结合 max 操作后，仅当点在圆形内且不在矩形内时结果为负。step 函数将其转换为可视图形。这种方法展示了如何通过简单数学组合出复杂设计。

5 超越几何：着色与质感

一旦定义了几何形状，我们可以通过着色添加颜色和质感。动态颜色可以通过将坐标与颜色通道挂钩实现。例如，使用 uv.x 和 uv.y 驱动 RGB 值，创建线性渐变：

```
vec3 color = vec3(uv.x, uv.y, 0.5);
```

这段代码将 x 坐标映射为红色通道，y 坐标映射为绿色通道，生成一个从左上到右下的渐变。若结合三角函数，如 sin 和 cos，可以创建条纹或波状图案：

```
float pattern = sin(uv.x * 10.0) * cos(uv.y * 10.0);
vec3 color = vec3(pattern);
```

这里，sin 和 cos 函数生成周期性变化，输出波状纹理。
为模拟光照效果，我们引入法线向量和漫反射模型。符号距离函数的梯度近似为法线，可通过 fwidth 函数计算：

```
vec3 normal = normalize(vec3(dfdx(d), dfdy(d), 1.0));
```

在这段代码中，dfdx(d) 和 dfdy(d) 计算距离场在 x 和 y 方向的偏导数，构成法线向量的 x 和 y 分量，z 分量设为 1.0 以标准化。接着，定义光源方向 lightDir，并应用漫反射模型：

```
float diff = max(dot(normal, lightDir), 0.0);
vec3 color = vec3(diff);
```

dot(normal, lightDir) 计算法线与光源的点积，max 确保非负，生成亮度变化。这使得一个平面圆形呈现出立体球体效果。进一步结合镜面反射，可以创建金属质感：

```
float spec = pow(max(dot(reflectDir, viewDir), 0.0), 10.0);
vec3 final_color = diff + spec;
```

这里，reflectDir 是反射方向，viewDir 是视角方向，pow 函数增强高光强度。通过这些步骤，寥寥几行代码便能实现逼真的材质效果。

6 进阶魔法：引入噪声与时间

为增加图形的复杂性和动态性，我们可以引入噪声函数和时间变量。噪声函数，如 Simplex 或 Perlin 噪声，通过伪随机扰动打破规则性。例如，用噪声扰动圆形边界创建云朵效果：

```
float noise = snoise(uv * 10.0);
float cloud_d = length(uv) - radius + noise * 0.1;
float cloud = smoothstep(0.0, 0.01, cloud_d);
```

在这段代码中，snoise 是噪声函数，输出值用于调整距离场，生成不规则形状。类似地，噪声可用于模拟大理石或木材纹理，通过调制颜色或法线实现。
时间变量 iTime 允许图形动态变化。例如，创建一个脉冲发光的球体：

```
float pulse = sin(iTime) * 0.1;
float d = length(uv) - (radius + pulse);
float circle = smoothstep(0.0, 0.01, d);
```

这里，sin(iTime) 生成周期性变化，叠加到半径上，使球体大小随时间脉冲。旋转效果可通过修改坐标实现：

```
float angle = iTime;
vec2 rotated_uv = vec2(uv.x * cos(angle) - uv.y * sin(angle), uv.x * sin(angle) + uv.
    ↪ y * cos(angle));
```

这段代码应用旋转矩阵到坐标 uv，生成动态旋转图形。这些技术将静态图形转化为生动动画，扩展了创作可能性。

回顾本文，归一化坐标 uv 作为万能画笔，符号距离函数作为图形定义语言，结合数学运算和 GLSL 内置函数，构成了着色器编程的核心框架。这种方法优势显著：图形基于数学定义，支持无限分辨率缩放；计算高度并行，契合 GPU 架构；动态和参数化调整简便，激发创意表达。

我们鼓励读者实践这些概念，从修改参数开始，逐步尝试组合不同符号距离函数，最终创造独特图形。资源如 Shadertoy 平台提供丰富示例，可供学习参考。在着色器的世界里，您不再是画家，而是世界的定义者 —— 想象力是唯一的边界。通过持续探索，您将解锁更多视觉魔法，从简单公式中孕育出无限复杂。