**头发渲染管线**

# 概述

在移动端使用的头发渲染技术主要是基于Kajiya-Kay模型，主要使用多面片叠加的方式进行的，此模型的计算公式不复杂，资源占用较少，是目前移动平台的主流头发渲染模型。下面将具体介绍渲染方法。

# 资源制作

2.1制作头发面片

根据角色的头发样式进行面片和发型制作，但是需要控制面片的个数和头发的层数，不宜过多，否则在alpha blending阶段会占用很多资源。

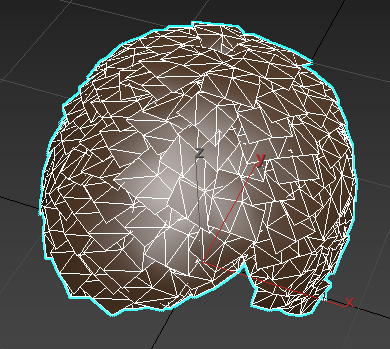
 

图1 头发模型

2.2 头发纹理、法线贴图制作

根据角色的头发颜色、杂乱度等制作相应的头发纹理贴图，此纹理需要根据头发丝的透明度设置相应的alpha通道，必须有alpha=1的部分。

根据发丝的生长方向制作切线空间的纹理贴图，此贴图主要用来计算相应的diffuse和specular。

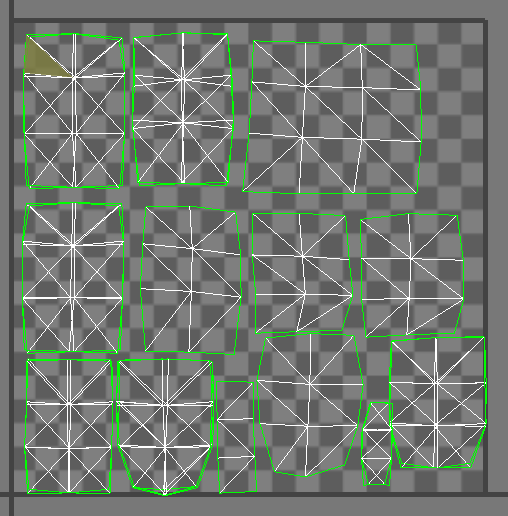
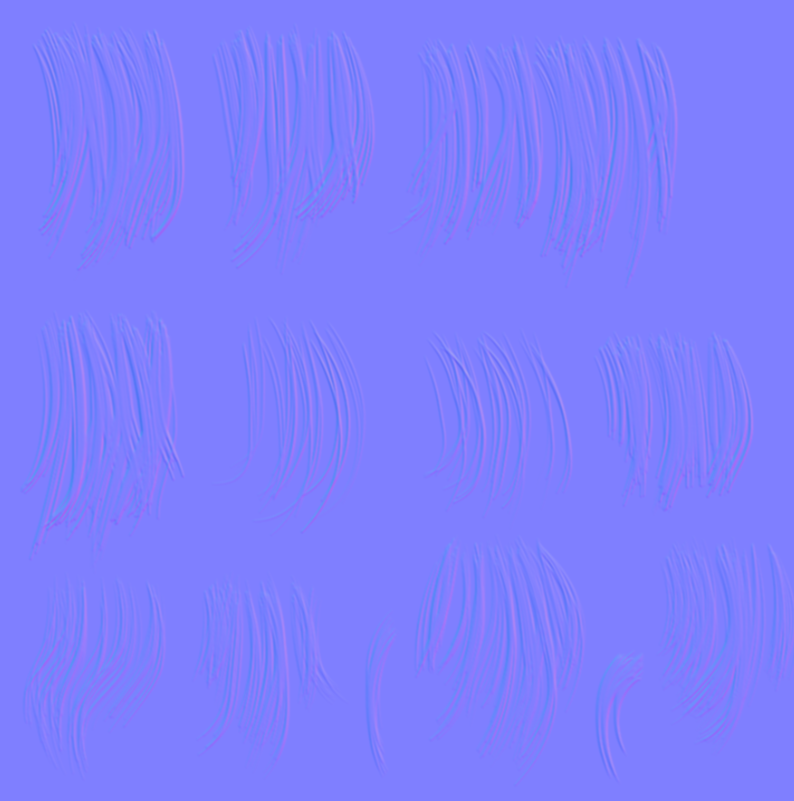


图2 头发纹理、法线贴图、UV展开

2.3 头发生长方向贴图

根据头发的生长方向制作相应的生长方向贴图，此贴图的RGB通道为发丝在切线空间下的生长方向，用于计算光照带来的高光偏移。



图3 高光扰动贴图

# 渲染流程

3.1光照计算原理

Kajiya-Kay模型的主要计算是基于头发的生长方向（切线）计算的，这样计算的好处是避免了圆柱形发丝表面法线方向的不一致性。基于实际的观察，如图4，可以发现头发主要包含两层高光反射，为了体现这两层高光反射，可以将头发的生长方向沿着头发的法线方向进行一定的扰动。



图4头发的两层高光

头发的制作主要是基于面片的，在进行切线扰动时我们使用的扰动法线是面片的法线方向，diffuse和specular计算时则使用头发的法线贴图。主要的光照计算如下：

可以根据需求添加多层高光效果。

3.2渲染顺序

头发属于半透明物体，合理的渲染顺序是十分必要的。使用多pass的alpha blending进行渲染，主要的渲染顺序如下：

(1)开启Z-Write ，Z-Test模式为less，剔除背面的三角形，绘制头发完全不透明的部分（使用alpha test的模式，关闭color buffer的写入）

(2)关闭Z-Write，Z-Test模式为equal，剔除背面的三角形，再次绘制完全不透明的部分（仍然使用alpha-test模式，这次写color buffer）

(3)关闭Z-Write，Z-Test模式改less，剔除正面的三角形，先绘制头发的背面（使用alpha blend的模式）

(4)开启Z-Write ，Z-Test模式改less，剔除背面的三角形，绘制头发的正面（使用alpha blend的模式）

其中（1）（2）可以合并在一起。

3.3渲染结果

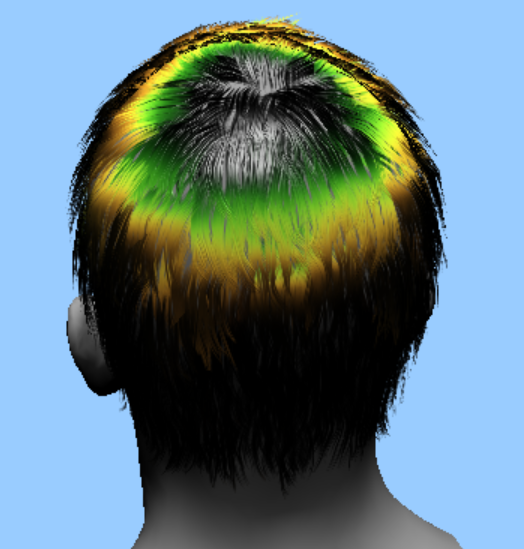
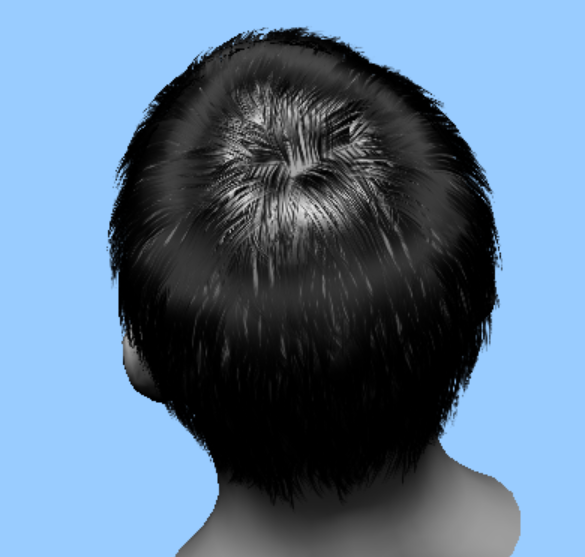
 

图5渲染结果

附录：

Vertex Shader

#version 100

uniform highp mat4 MVPMatrix;

uniform mediump vec3 LightDir;

uniform highp vec3 EyePos;

attribute highp vec3 inVertex;

attribute mediump vec3 inNormal;

attribute mediump vec3 inTangent;

attribute mediump vec3 inBiNormal;

attribute highp vec2 inTexCoords1;

varying mediump vec3 NormalModel;

varying mediump vec2 TexCoords1;

varying mediump mat3 ModelToTangentMatrix;

varying mediump vec3 LightDir\_TangentSpace;

varying mediump vec3 ViewDir\_TangentSpace;

void main()

{

// Transform position

gl\_Position = MVPMatrix \* vec4(inVertex, 1.0);

// texture coordinates

TexCoords1 = inTexCoords1;

//lightDir

highp vec3 LightDir\_ModelSpace = normalize(LightDir);

// viewDir

highp vec3 ViewDir\_ModelSpace = normalize(EyePos - inVertex.xyz);

// normal

NormalModel = normalize(inNormal);

highp vec3 tangentModel = normalize(inTangent);

highp vec3 binormalModel = normalize(inBiNormal);

ModelToTangentMatrix = mat3(tangentModel.x, binormalModel.x, NormalModel.x,

tangentModel.y, binormalModel.y, NormalModel.y,

tangentModel.z, binormalModel.z, NormalModel.z);

LightDir\_TangentSpace = normalize(ModelToTangentMatrix \* LightDir\_ModelSpace);

ViewDir\_TangentSpace = normalize(ModelToTangentMatrix \* ViewDir\_ModelSpace);

}

Fragment Shader

#version 100

uniform sampler2D diffuseMap;

uniform sampler2D normalMap;

uniform sampler2D specFlowMap;

uniform lowp float Pass;

varying mediump vec3 NormalModel;

varying mediump vec2 TexCoords1;

varying mediump mat3 ModelToTangentMatrix;

varying mediump vec3 LightDir\_TangentSpace;

varying mediump vec3 ViewDir\_TangentSpace;

const mediump vec3 \_Spec1Color = vec3(0.1, 0.1, 0.1); //偏移高光1的颜色

const mediump float \_Spec1EX = 72.0; //偏移高光1的Gloss

const mediump float \_Spec1Shift = 0.5; //高光1的偏移量

const mediump vec3 \_Spec2Color = vec3(0.2, 0.2, 0.2); //偏移高光2的颜色

const mediump float \_Spec2EX = 48.0; //偏移高光2的Gloss

const mediump float \_Spec2Shift = 0.4;//高光2的偏移量

const mediump vec3 \_LightColor = vec3(1.0, 1.0, 1.0);

const mediump float \_SpecBlinnEX = 72.0;

mediump float lerp(mediump float a, mediump float b, mediump float w) {

return a + w\*(b-a);

}

mediump float saturate(mediump float x){

if(x<0.0) return 0.0;

if(x>1.0) return 1.0;

return x;

}

mediump vec3 UnpackScaleNormal(mediump vec4 packedNormal){

mediump vec3 normal;

normal.xyz = packedNormal.xyz \* 2.0 - 1.0;

return normal;

}

void main()

{

highp vec2 texcoords1 = TexCoords1;

texcoords1.y = 1.0 - texcoords1.y;

highp vec4 Albedo = texture2D(diffuseMap, texcoords1).rgba;

if(Pass == 1.0){

if(Albedo.a <1.0){

discard;

}

}//多个Pass共用一个Shader，第一个Pass只渲染非透明部分，透明部分进行剔除

highp vec3 Normal\_TangentSpace\_normalized = normalize(UnpackScaleNormal(texture2D(normalMap, texcoords1)));

highp vec3 tanFlow = texture2D(specFlowMap, texcoords1).rgb \* 2.0 - 1.0;//切线空间下的发丝生长方向

highp vec3 normalTangent = normalize(ModelToTangentMatrix \* NormalModel);

highp vec3 tanDir1 = normalize(tanFlow + normalTangent \* \_Spec1Shift);//高光偏移1

highp vec3 tanDir2 = normalize(tanFlow + normalTangent \* \_Spec2Shift);//高光偏移2

highp vec3 halfDirTangent = normalize(LightDir\_TangentSpace + ViewDir\_TangentSpace);//halfView

highp float nh = max(dot(Normal\_TangentSpace\_normalized, halfDirTangent), 0.0);

highp float nl = max(dot(Normal\_TangentSpace\_normalized, LightDir\_TangentSpace), 0.0);

highp float diffuseTerm = nl;

highp float specularTerm = pow(nh , \_SpecBlinnEX);

highp float th1 = dot(tanDir1, halfDirTangent);

highp float th2 = dot(tanDir2, halfDirTangent);

highp vec3 spec1 = clamp(0.0, 1.0, nl) \* \_Spec1Color \* pow(sqrt(1.0 - th1 \* th1), \_Spec1EX);

//此处clamp(0.0, 1.0, nl)可选择去掉，加上时背光部分将不会有高光反射

highp vec3 spec2 = clamp(0.0, 1.0, nl) \* \_Spec2Color \* pow(sqrt(1.0 - th2 \* th2), \_Spec2EX);

highp vec3 specBlinn = clamp(0.0, 1.0, nl) \* \_LightColor \* specularTerm;

highp vec3 diffuseColor = Albedo.rgb \* vec3(0.0, 0.0, 0.0);

highp vec3 color = vec3(0.0, 0.0, 0.0);

color += diffuseColor \* diffuseTerm;

color += spec1;

color += spec2;

color += specBlinn;

gl\_FragColor = vec4(color, Albedo.a);

}