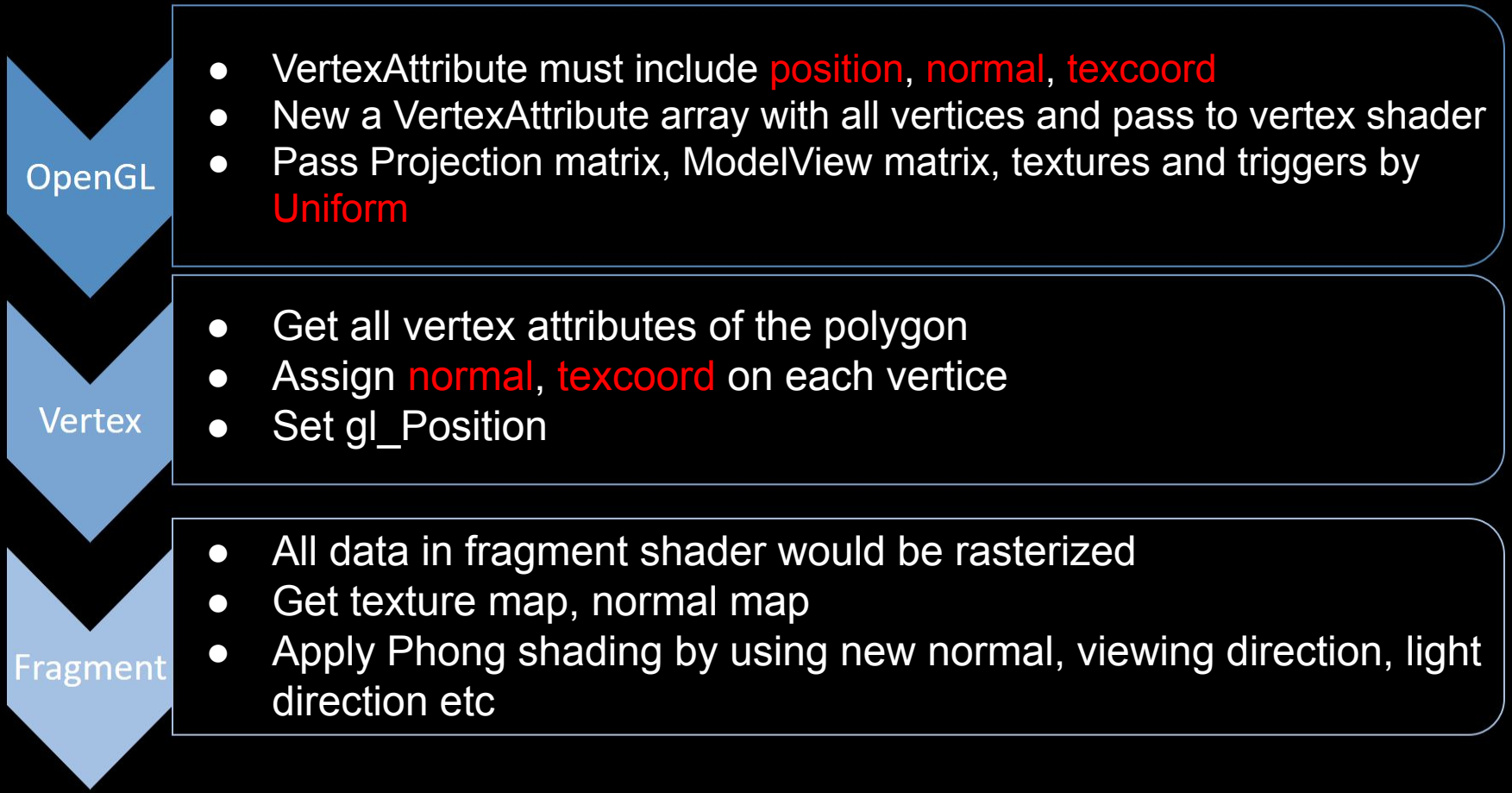
HW2_Hint

Overview


OpenGL
VertexAttribute must include position, normal, texcoord
New a VertexAttribute array with all vertices and pass to vertex shader
Pass Projection matrix, ModelView matrix, textures and triggers by Uniform
Vertex
Get all vertex attributes of the polygon
Assign normal, texcoord on each vertice
Set gl_Position
Fragment
All data in fragment shader would be rasterized
Get texture map, normal map
Apply Phong shading by using new normal, viewing direction, light direction etc

OpenGL

```
glDrawArrays(GL_TRIANGLES, first, count);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute),
                      (void*)(offsetof(VertexAttribute, position)));
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute),
                      (void*)(offsetof(VertexAttribute, normal)));
glUniformMatrix4fv(glGetUniformLocation(currentProgram, "M"),
                   1, GL_FALSE, &M[0][0]);
```

Vertex

```
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
uniform mat4 M;
out vec2 uv;
```

Fragment

```
in vec2 uv;
out vec4 FragColor;
```

Texture

OpenGL

```
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute),
                      (void*)(offsetof(VertexAttribute, texcoord)));

glActiveTexture(GL_TEXTURE0+0);
glBindTexture(GL_TEXTURE_2D), TextureID);
glUniformMatrix4fv(glGetUniformLocation(currentProgram,"mainTex"), 0);
```

Vertex

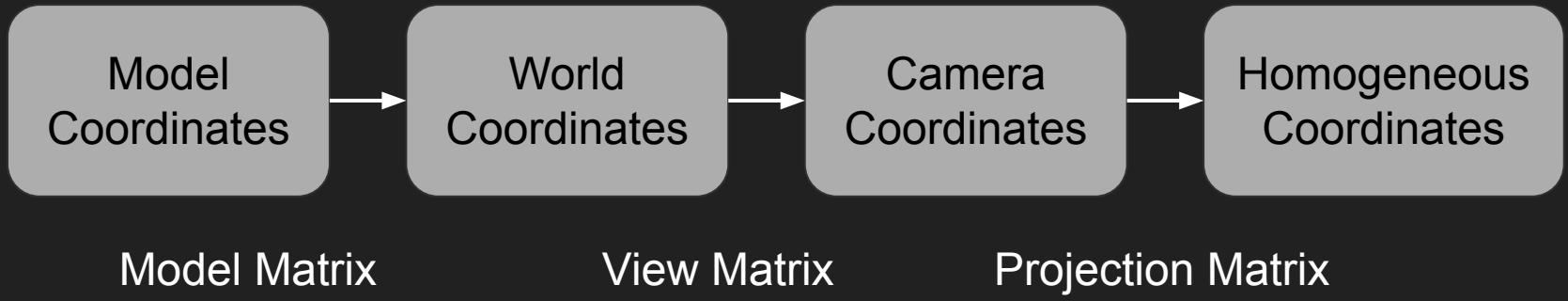
```
layout(location = 2) in vec3 texcoord;
out vec2 uv;
uv = texcoord;
```

Fragment

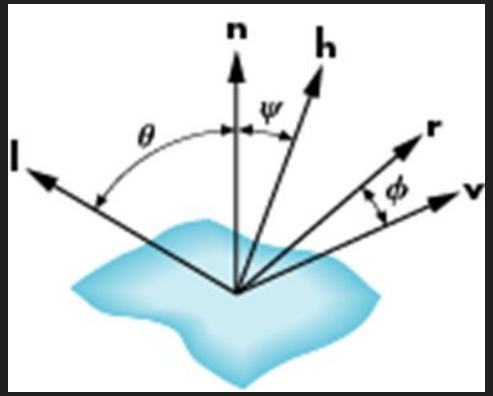
```
uniform sampler2D mainTex;
in vec2 uv;
out vec4 FragColor;
FragColor = texture2D(mainTex, uv);
```

View Space

- gl_Position = Projection * (View * (Model * vertex));



phongFragment.frag



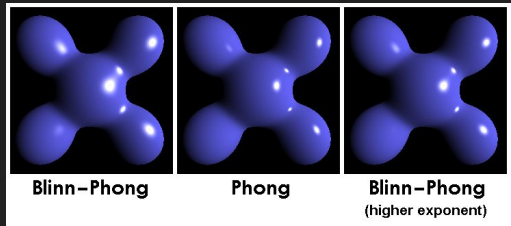

```
void main(){

    albedo = texture2D(mainTex, texcoord);

    ambient = La * Ka * albedo;
    diffuse = Ld * Kd * albedo * dot(L,N); // must > 0

    specularPhong = Ls * Ks * pow(dot(V,R), gloss/4.0);
    specularBlinn = Ls * Ks * pow(dot(N,H), gloss);
    specular = mix(specularPhong, specularBlinn, 0);
    // change to 1 see different between phong and blinn

    color = ambient + diffuse + specular;
    // out color must be vec4
}
```



Dissolve.frag

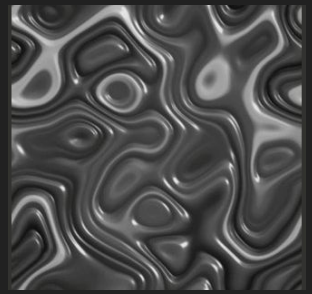
```
void main(){

    albedo = texture2D(mainTex, texcoord);
    noise = texture(noiseTex, texcoord).x;

    if(noise - _Threshold < 0.0)
        discard;

    // use EdgeLength/2 as threshold to prevent
    // exactly _Threshold + _EdgeLength - noise is 0
    flag = step(_EdgeLength/2, _Threshold+_EdgeLength-noise);

    color = mix(albedo, _EdgeColor, flag)
    // out color must be vec4
}
```



noise texture

Toon.frag

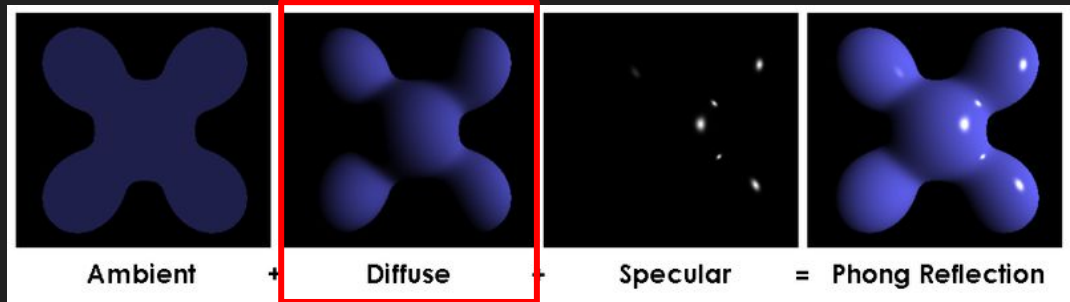
ramp texture

```
void main(){

    albedo = texture2D(mainTex, texcoord);
    rampCoord = dot(N,L) * 0.5 + 0.5; // must between (0,1)

    diffuse = texture(rampTex, vec2(rampCoord,rampCoord));

    color = diffuse * Kd * albedo;
    // out color must be vec4
}
```



Note

- Normalize all direction vector to smooth transition zone.
- fragment shader裡若沒有將法向量正規化，也就是只做vec3 n = normal;而不做vec3 n = normalize(normal);那麼顏色的過度區域就會不平滑。

只做vec3 n = normal;所呈現出來的亮色區為不平滑狀況。

做vec3 n = normalize(normal);所呈現出來的亮色區為平滑貌。