Renderização não-fotorrealística com shaders de vértice e fragmentos

José Morista C. da Silva

Introdução

Renderização não-fotorrealística

A renderização não-fotorrealista (NPR) é uma área de computação gráfica que se concentra em permitir uma grande variedade de estilos expressivos para a arte digital. Ao contrário da computação gráfica tradicional, focada no fotorrealismo, esta é inspirada em estilos artísticos, como pintura, desenho, ilustração técnica e desenhos animados.

Renderização não-fotorrealística

Em muitos cenários, o fotorrealismo pode não ser desejado. No caso de ilustrações técnicas por exemplo, técnicas de renderização não-fotorrealistas podem ser utilizadas para aprimorar o entendimento da cena ou do objeto que está sendo desenhado, de forma a não ocultar recursos importantes, como contornos.

Shaders de vértice e fragmentos

- O **Vertex Shader** (shader de vértice) é um programa que é processado no pipeline da GPU (graphic processing unit), ele é capaz de trabalhar nos vértices do modelo 3D e com isso modificar suas estruturas a tempo de execução.

- O **Fragment Shader** (shader de fragmentos) é outra etapa programável do pipeline da GPU, este é capaz de trabalhar na estrutura dos pixels depois que um modelo 3D é rasterizado.

Objetivos

Neste trabalho, serão implementados shaders de vértices e fragmentos que realizem a renderização de modelos tridimensionais por diferentes técnicas não-fotorrealistas.

Técnicas aplicadas

Ferramentas utilizadas

- ReactJs
- WebGl2
- Visual Studio Code

Código fonte disponível em:

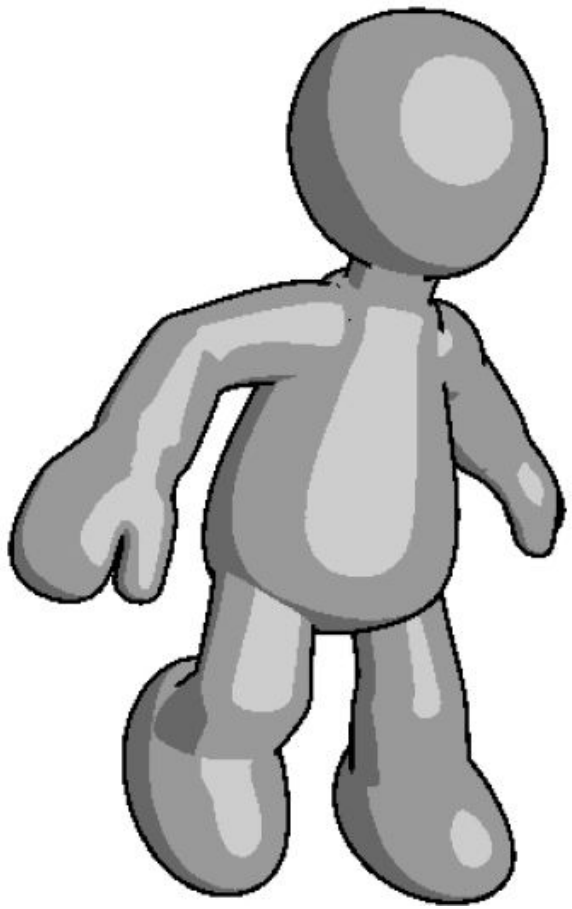
https://github.com/josemorista/react-hooks-webgl2

Preparativos

- Todos os modelos foram normalizados e submetidos a uma mesma iluminação pontual de cor branca a uma distância fixa do modelo.
- A câmera utilizada na cena também estava a uma distância fixa do modelo e em coordenadas iguais a fonte de luz.
- Para medida de comparação, foram utilizados modelos renderizados com iluminação de Phong.

Cartoon

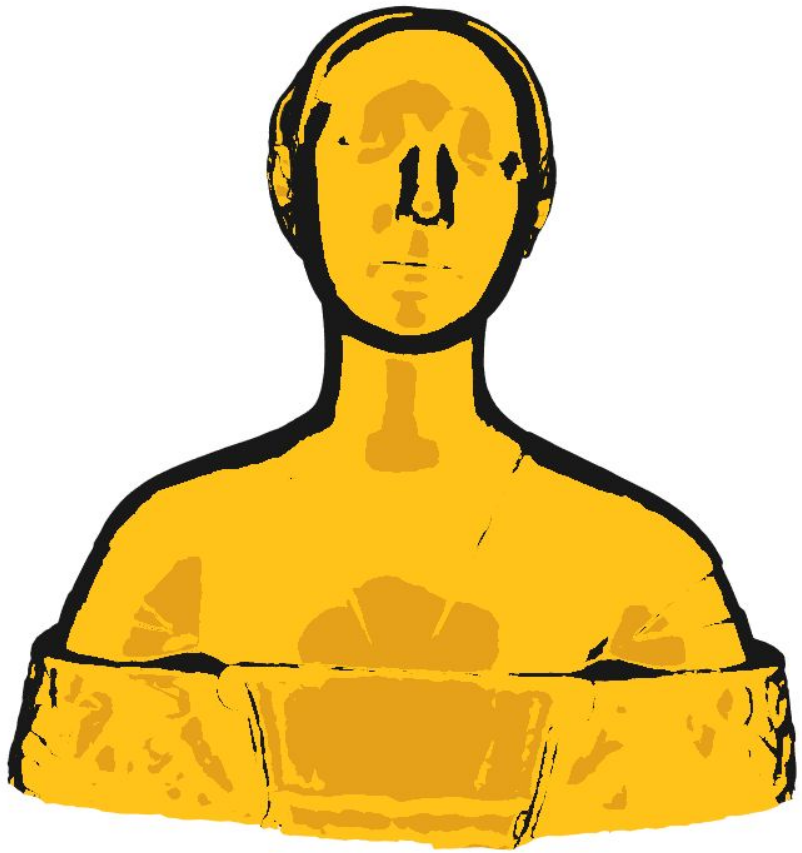
Também conhecido como cel-shading, provê uma representação cartunesca do modelo. É composto por etapas de outline e highlight.



Phong Lighting

Phong Lighting

Phong Lighting



Cartoon

Código fonte: shader de vértices

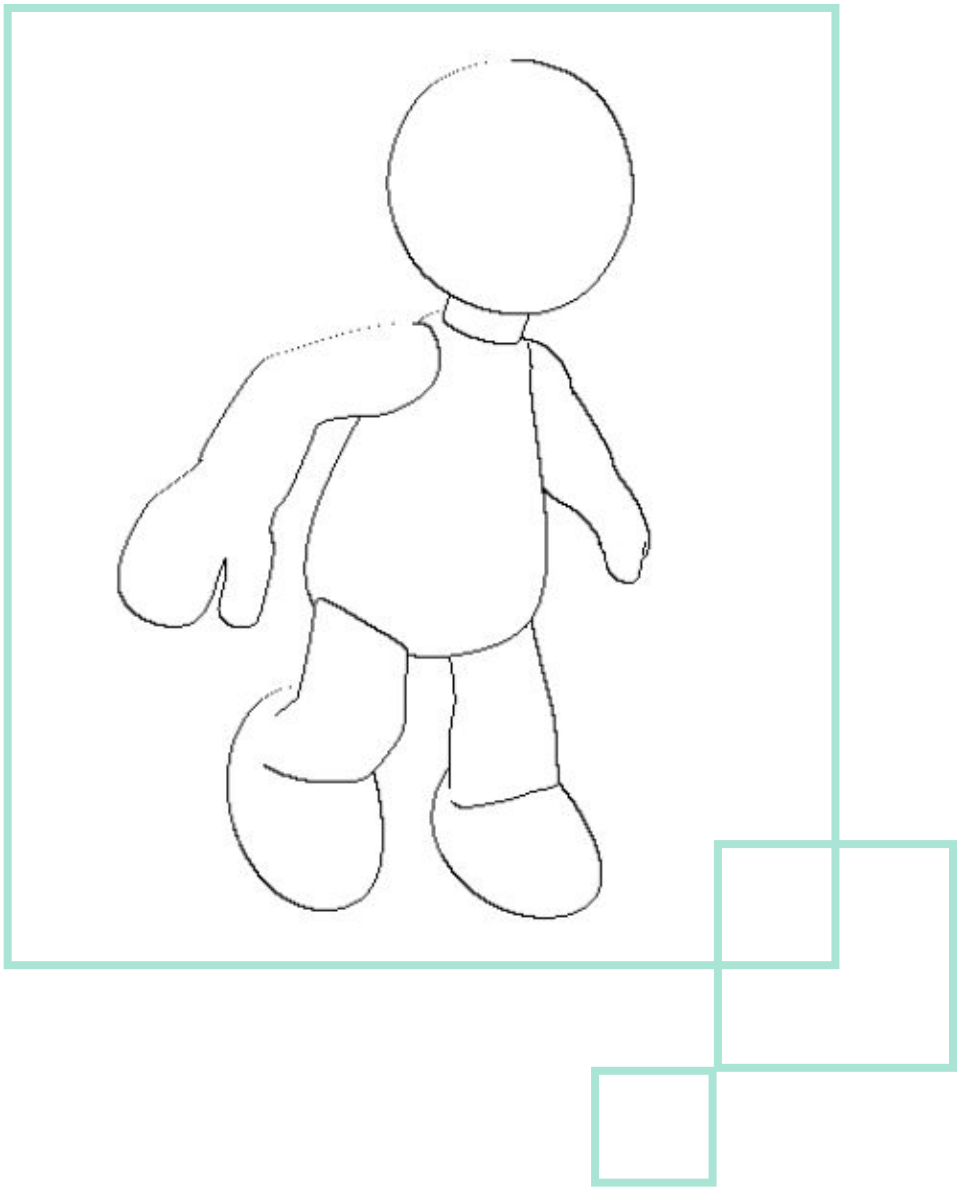
```
#version 300 es
precision mediump float;
in vec3 aVertexPosition;
in vec3 aVertexNormal;
uniform mat4 uModelViewTransformationMatrix;
uniform mat4 uProjectionTransformationMatrix;
uniform mat4 uNormalTransformationMatrix;
out vec3 fragPos;
out vec3 fragNormal;
void main () {
  fragPos = (uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0)).xyz;
  fragNormal = (uNormalTransformationMatrix * vec4(aVertexNormal, 1.0)).xyz;
  gl_Position = uProjectionTransformationMatrix * uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0);
}
```

Código fonte: shader de fragmentos

```
#version 300 es
precision mediump float;
uniform vec3 uLightPosition;
uniform vec3 uLightColor;
uniform float uLightAmbientIntensity;
uniform float uLightDiffuseIntensity;
uniform float uLightSpecularIntensity;
uniform vec3 uMaterialDiffuse;
uniform vec3 uMaterialSpecular;
uniform vec3 uMaterialAmbient;
uniform float uMaterialSpecularPower;
in vec3 fragPos;
in vec3 fragNormal;
out vec4 fragColor;
void main()
{
  vec3 N = normalize(fragNormal);
  vec3 V = normalize(-fragPos);
  vec3 vertexToLightSource = fragPos- uLightPosition;
  float distance = length(vertexToLightSource);
  float attenuation = 1.0 / distance; // linear attenuation
  vec3 lightDirection = normalize(vertexToLightSource);
  // Ambient Lighting
  vec3 Ia = uMaterialAmbient * uLightAmbientIntensity;
  // Diffuse Lighting
  vec3 fragmentColor = vec3(uLightColor) * vec3(uMaterialDiffuse) *
uLightDiffuseIntensity;
  // Outline
  if (dot(V, N) < mix(0.4, 0.4, max(0.0, dot(N, -lightDirection)))) {
    fragmentColor = fragmentColor * vec3(uLightColor) * vec3(0,0,0);
  }
  // HighLights
  if (dot(-lightDirection, N) > 0.0 && attenuation * pow(max(0.0,
dot(reflect(lightDirection, N), V)), uMaterialSpecularPower) > 0.5) {
    fragmentColor = uLightSpecularIntensity * vec3(0.8) * fragmentColor;
  }
  fragColor = vec4(Ia + fragmentColor, 1.0);
}
```

Draw

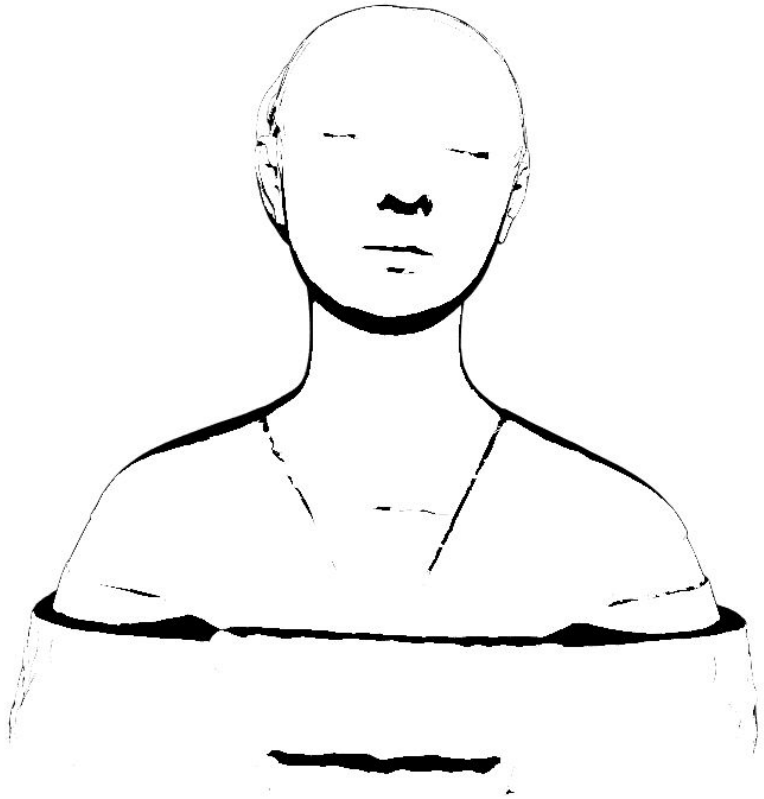
Derivado do cel-shading, provê uma representação com aspecto de esboço de um desenho. Marcado por um fino outline e remoção de especularidades.



Phong Lighting

Phong Lighting

Phong Lighting



Draw

Código fonte: shader de vértices

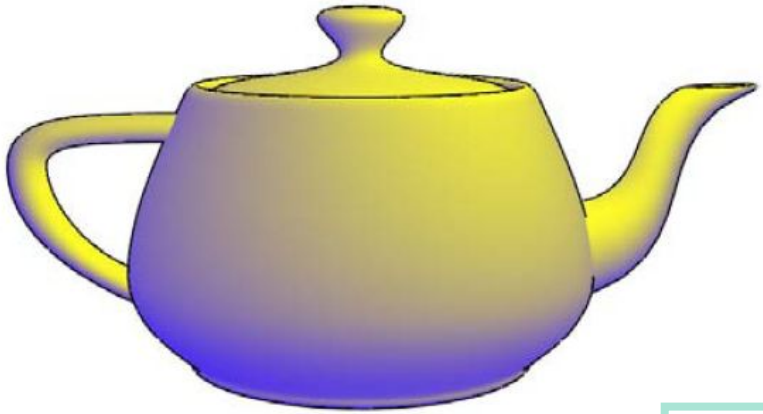
```
#version 300 es
precision mediump float;
in vec3 aVertexPosition;
in vec3 aVertexNormal;
uniform mat4 uModelViewTransformationMatrix;
uniform mat4 uProjectionTransformationMatrix;
uniform mat4 uNormalTransformationMatrix;
out vec3 fragPos;
out vec3 fragNormal;
void main () {
  fragPos = (uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0)).xyz;
  fragNormal = (uNormalTransformationMatrix * vec4(aVertexNormal, 1.0)).xyz;
  gl_Position = uProjectionTransformationMatrix * uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0);
}
```

Código fonte: shader de fragmentos

```
#version 300 es
precision mediump float;
uniform vec3 uLightPosition;
uniform vec3 uLightColor;
in vec3 fragPos;
in vec3 fragNormal;
out vec4 fragColor;
void main()
{
  vec3 N = normalize(fragNormal);
  vec3 V = normalize(-fragPos);
  vec3 vertexToLightSource = fragPos - uLightPosition;
  vec3 lightDirection = normalize(vertexToLightSource);
  vec3 fragmentColor = vec3(1, 1, 1);
  if (dot(V, N) < mix(0.088, 0.8, max(0.0, dot(N, -lightDirection)))) {
    fragmentColor = fragmentColor * vec3(uLightColor) * vec3(0,0,0);
  }
  fragColor = vec4(fragmentColor, 1.0);
}
```

Gooch

O modelo de iluminação Gooch foi projetado para fornecer iluminação sem obscurecer a forma do modelo, as linhas ou os destaques especulares. Realiza uma mistura entre uma cor quente e fria.



Phong Lighting

Phong Lighting

Phong Lighting

Gooch

Código fonte: shader de vértices

```
#version 300 es
  precision mediump float;
  in vec3 aVertexPosition;
  in vec3 aVertexNormal;
  uniform vec3  uLightPosition;
  uniform mat4 uModelViewTransformationMatrix;
  uniform mat4 uProjectionTransformationMatrix;
  uniform mat4 uNormalTransformationMatrix;
  out vec3  N;
  out vec3  L;
  out vec3  R;
  out vec3  E;
  void main()
  {
     N = normalize(((uNormalTransformationMatrix * vec4(aVertexNormal, 1.0)).xyz));
     vec3 pos = (uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0)).xyz;
     E = normalize(-pos); // we are in Eye Coordinates, so EyePos is (0,0,0)
     L = normalize(pos - uLightPosition);
     R = reflect(L, N);
     gl_Position = uProjectionTransformationMatrix * uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0);
  }
```

Código fonte: shader de fragmentos

```
#version 300 es
precision mediump float;
uniform vec3  uMaterialDiffuse;
in vec3  N;
in vec3  L;
in vec3  R;
in vec3  E;
out vec4 fragColor;
void main()
{
    vec3 cool    = min(vec3(0.0, 0.0, 0.6) + 0.35 * uMaterialDiffuse, 1.0);
    vec3 warm    = min(vec3(0.6, 0.6, 0.0) + 0.45 * uMaterialDiffuse, 1.0);
    vec3 final   = mix(cool, warm, dot(N, -L));
    vec3 nreflect = normalize(R);
    vec3 nview    = normalize(E);
    float spec    = max(dot(nreflect, nview), 0.0);
    spec          = pow(spec, 32.0);
    fragColor = vec4(min(final + spec, 1.0), 1.0);
}
```

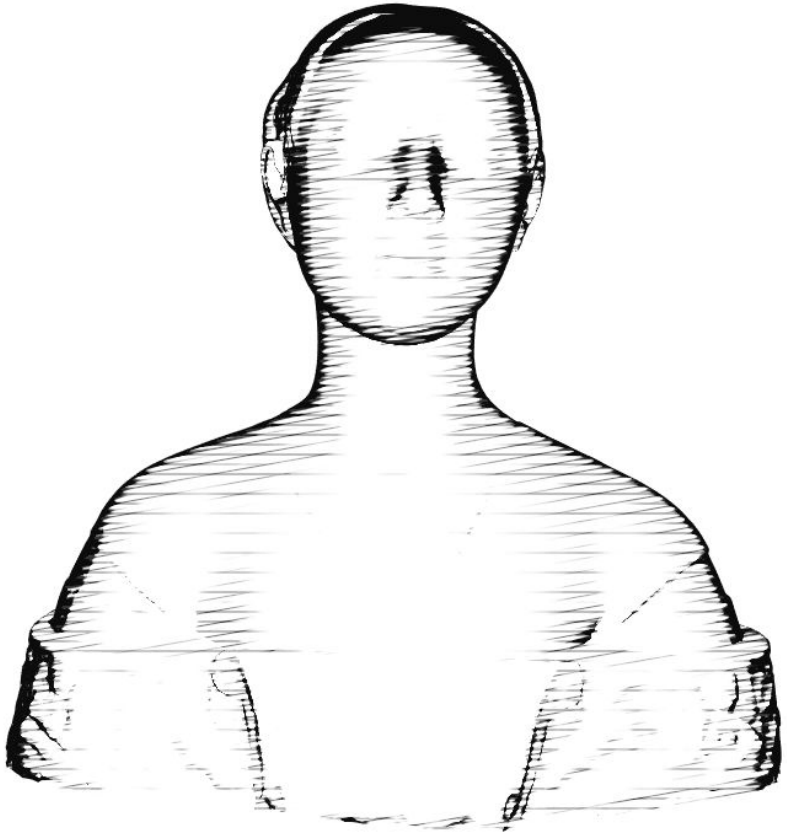
Hatching

Outro método utilizado é o hatching, comumente utilizado em desenhos de caneta e tinta para mostrar a forma e diferenciar as regiões iluminadas e apagadas de um objeto.

Phong Lighting

Phong Lighting

Phong Lighting



Hatch

Código fonte: shader de vértices

```
#version 300 es
precision mediump float;
in vec3 aVertexPosition;
in vec3 aVertexNormal;
uniform mat4 uModelViewTransformationMatrix;
uniform mat4 uProjectionTransformationMatrix;
uniform mat4 uNormalTransformationMatrix;
out vec3 fragPos;
out vec3 fragNormal;
void main () {
  fragPos = (uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0)).xyz;
  fragNormal = (uNormalTransformationMatrix * vec4(aVertexNormal, 1.0)).xyz;
  gl_Position = uProjectionTransformationMatrix * uModelViewTransformationMatrix * vec4(aVertexPosition, 1.0);
}
```

Código fonte: shader de fragmentos

```
#version 300 es
precision mediump float;
in vec3 fragNormal;
in vec3 fragPos;
uniform vec3 uLightPosition;
out vec4 fragColor;
mat2 rotate(float angle) { return mat2(cos(angle), -sin(angle), sin(angle), cos(angle));}
float horizontalLine(vec2 pixel, float y_pos, float width) {
            return 1.0 - smoothstep(-1.0, 1.0, abs(pixel.y - y_pos) - 0.5 * width);
          }
void main () {
  vec3 N = normalize(fragNormal);
  vec3 L = normalize(fragPos - uLightPosition);
  float df = clamp(dot(N, -L), 0.0, 1.0);
  vec2 pos = gl_FragCoord.xy;
  // Define the group of lines
  float lineWidth = 7.0 * (1.0 - smoothstep(0.0, 0.3, df)) + 0.5;
  float linesSep = 16.0;
  vec2 grid_pos = vec2(pos.x, mod(pos.y, linesSep));
  float line_1 = horizontalLine(grid_pos, linesSep / 2.0, lineWidth);
grid_pos.y = mod(pos.y + linesSep / 2.0, linesSep);
    float line_2 = horizontalLine(grid_pos, linesSep / 2.0, lineWidth);
    // Rotate the coordinates
    pos = rotate(radians(10.0)) * pos;
    // Define the group of lines
    linesSep = 10.0;
    grid_pos = vec2(pos.x, mod(pos.y, linesSep));
    float line_3 = horizontalLine(grid_pos, linesSep / 2.0, lineWidth);
    grid_pos.y = mod(pos.y + linesSep / 2.0, linesSep);
    float line_4 = horizontalLine(grid_pos, linesSep / 2.0, lineWidth);
    float color = 1.0;
    color -= 0.8 * line_1 * (1.0 - smoothstep(0.5, 0.75, df));
    color -= 0.8 * line_2 * (1.0 - smoothstep(0.4, 0.5, df));
    color -= 0.8 * line_3 * (1.0 - smoothstep(0.4, 0.65, df));
    color -= 0.8 * line_4 * (1.0 - smoothstep(0.2, 0.4, df));
    color = clamp(color, 0.05, 1.0);
    fragColor = vec4(vec3(color), 1.0);
  }
```

Conclusões

Conclusões

O trabalho realizado apresentou com sucesso a implementação e execução de shaders capazes de executar as técnicas de renderização não-fotorrealista Cartoon, Draw, Gooch e Hatching sobre modelos 3d.

Trabalhos futuros

- Maior possibilidade parametrização dos shaders implementados.
- Implementação de maior nível de abstração ao projeto para facilitar a renderização dos modelos e parametrização dos shaders.
- Execução das técnicas apresentadas em modelos texturizados.
- Adição de novas técnicas NPR.

Dúvidas?

Referências

[1] Non-Photorealistic Rendering with Pixel and Vertex Shaders
Jason L. Mitchell
http://developer.amd.com/wordpress/media/2012/10/ShaderX_NPR.pdf

[2] Non-photorealistic rendering
https://en.wikipedia.org/wiki/Non-photorealistic_rendering

[3] GLSL Programming/Unity/Toon Shading
https://en.wikibooks.org/wiki/GLSL_Programming/Unity/Toon_Shading

[4] react-hooks-webgl2
https://github.com/josemorista/react-hooks-webgl2