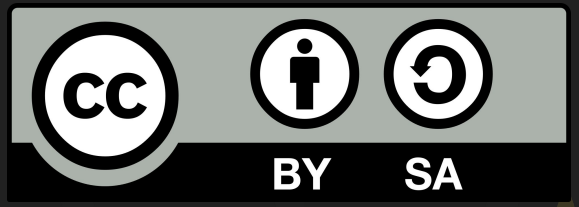
Computação Gráfica para Jogos Eletrônicos

Métodos de renderização e shaders

Slides por: Gustavo Ferreira Ceccon (gustavo.ceccon@usp.br)
Assistentes: Leonardo Tórtoro Pereira (leonardop@usp.br),
Gabriel Simmel (gabriel.simmel.nascimento@usp.br) e Ítalo Tobler (italo.tobler.silva@usp.br)



Este material é uma criação do
Time de Ensino de Desenvolvimento de Jogos Eletrônicos (TEDJE)
Filiado ao grupo de cultura e extensão Fellowship of the Game (FoG), vinculado ao ICMC - USP

Objetivos

- ➔ Básico de GPU e OpenGL, além de guia de aprendizado para GLSL
- ➔ Técnicas de render e shading: forward, deferred, Lambert e PBR
- ➔ Conceitos e algoritmos básicos por trás das principais técnicas utilizadas na área, além de exemplos de utilização

Índice

1. Introdução

1. Introdução

- OpenGL (Kronos) vs. DirectX (Microsoft) vs. Vulkan (Kronos) vs. Metal (Apple)
  - APIs diferentes com o mesmo “propósito”
- GLSL vs. HLSL vs. CG
  - Linguagens de shader legíveis

1. Introdução

- Quem quiser aprender mais:
- https://learnopengl.com/ (curso de OpenGL)
- https://www.youtube.com/watch?v=V5XFrIhLpGQ (shader graph)
- https://unity3d.com/pt/learn/tutorials/topics/graphics/gentle-introduction-shaders (Unity shaders)
- https://docs.unity3d.com/Manual/Shaders.html (manual geral)
- https://docs.unity3d.com/Manual/SL-Reference.html (especificação CG)

2. Modelos de Iluminação

2. Modelos de Iluminação

- Materiais
  - Instâncias de um programa (um ou mais shaders)
  - Descrevem como o objeto deve se comportar visualmente
  - Podem conter informações como reflexão, transparência, quão metálico, quão liso, etc.

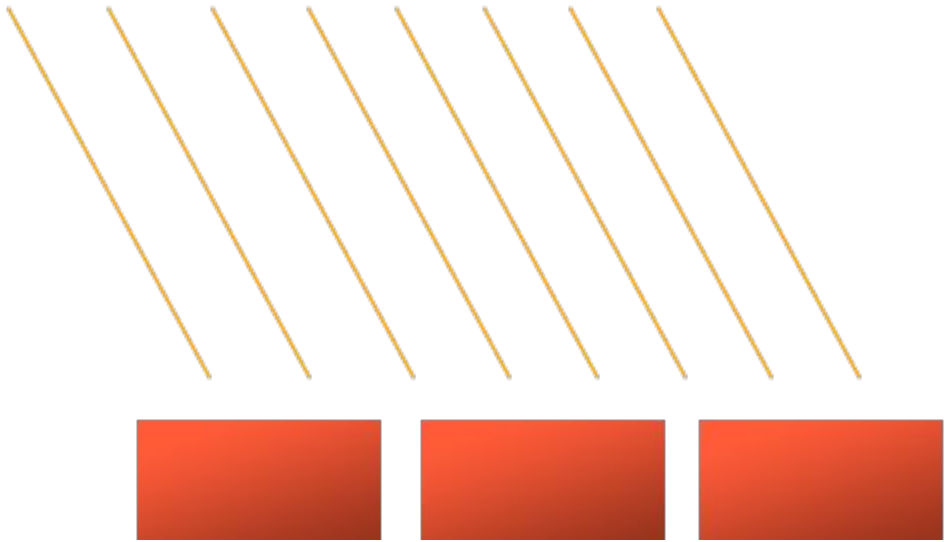
2. Modelos de Iluminação

- Modelos de iluminação são os modelos matemáticos e físicos de como os objetos interagem com a luz
  - Lambert Blinn-Phong
  - Physically Based Rendering (PBR)

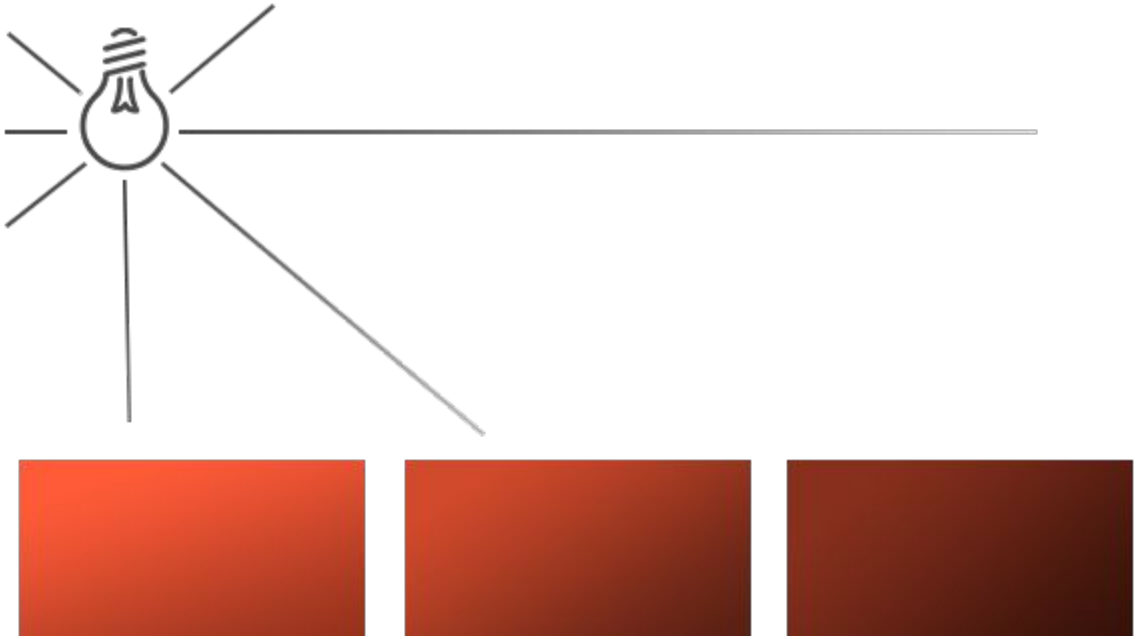
2. Modelos de Iluminação

- Difusa
  - Cor que objeto recebe sob luz direta
  - Mais forte na direção da luz e esmaece conforme o ângulo da superfície aumenta
- Especular
  - Cor de destaque de um objeto.
  - Aparece como reflexão da luz na superfície
- Exemplo

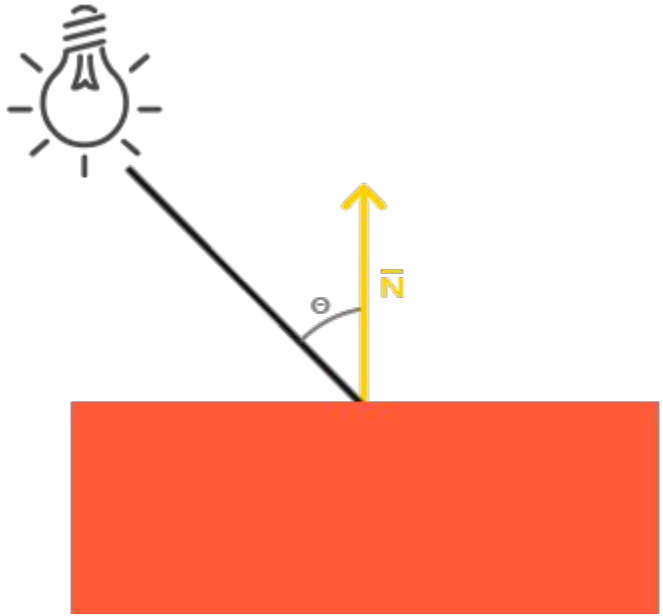
Luz Direcional



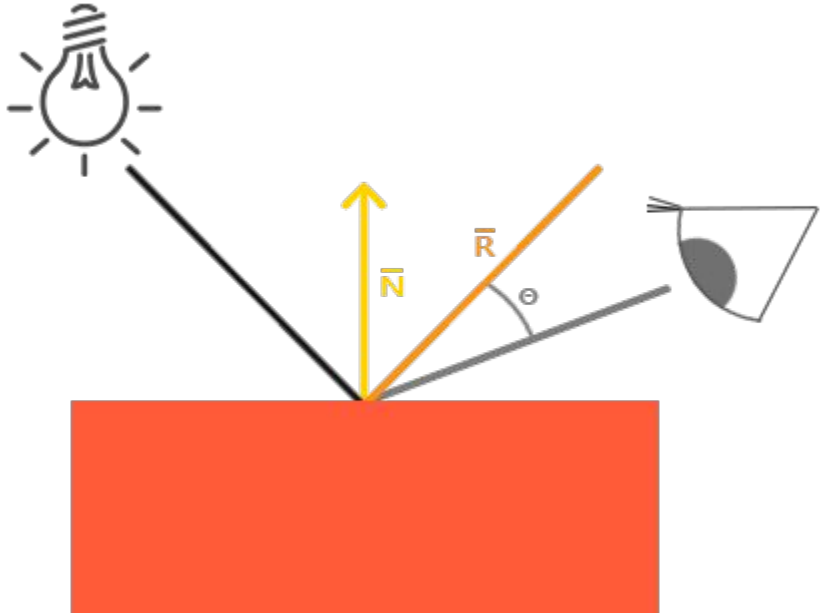
Ponto de Luz



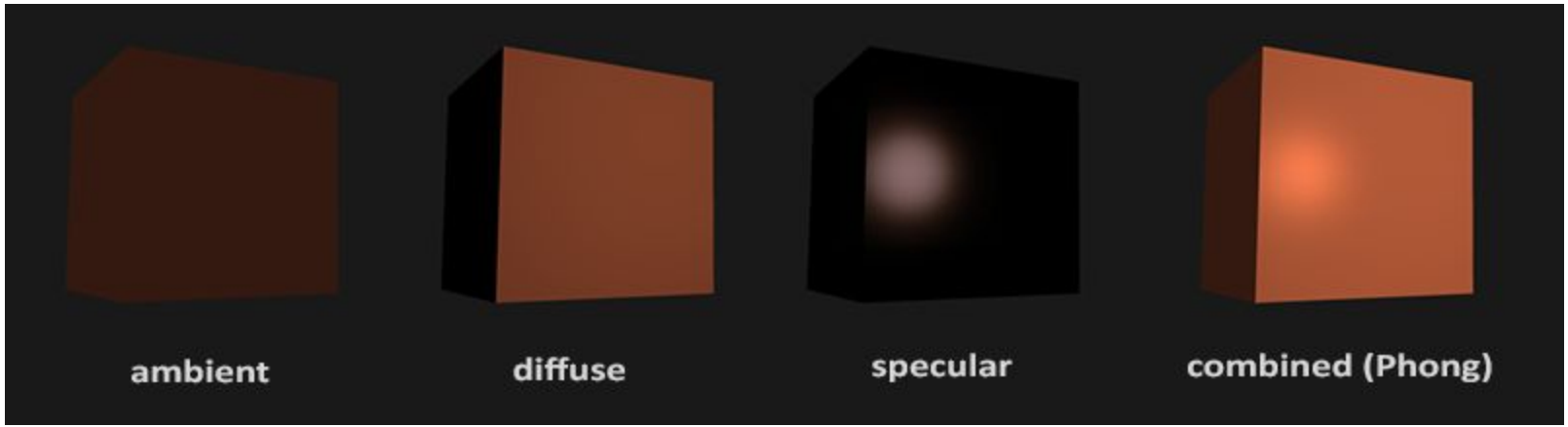
Difusa



Especular



Modelos de iluminação



2. Modelos de Iluminação

- Physically Based Rendering (PBR)
  - Modelo de iluminação que segue algumas das regras da física, como conservação de energia, Fresnel e oclusão
  - Chamado Standart Shader na Unity, é o modelo de iluminação padrão
  - Dois principais parâmetros: metallic e smoothness

2. Modelos de Iluminação

- The metallic parameter of a material determines how “metal-like” the surface is. When a surface is more metallic, it reflects the environment more and its albedo colour becomes less visible. At full metallic level, the surface colour is entirely driven by reflections from the environment. When a surface is less metallic, its albedo colour is more clear and any surface reflections are visible on top of the surface colour, rather than obscuring it.

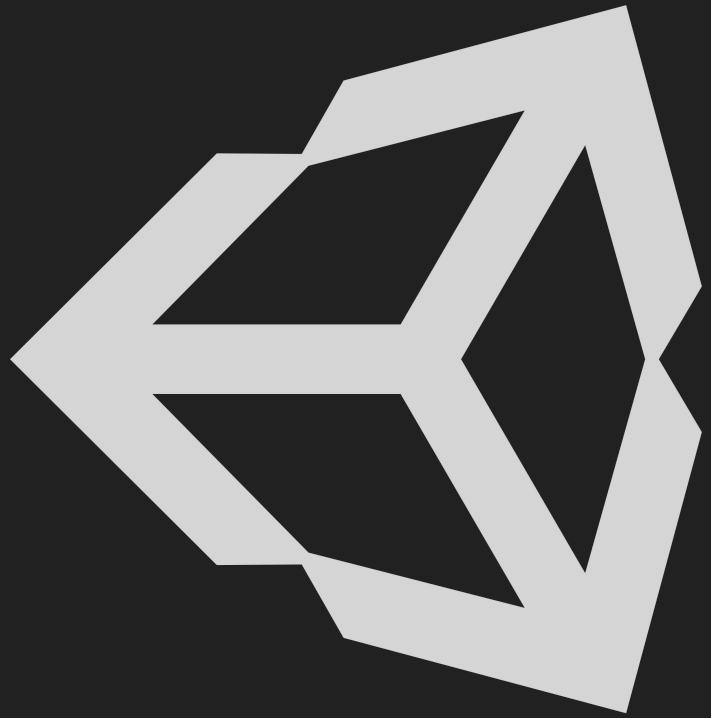
2. Modelos de Iluminação

- The effect of this microsurface detail is represented by the amount the light that is diffused as it bounces off the object. With a smooth surface, all light rays tend to bounce off at predictable and consistent angles. Taken to its extreme, a perfectly smooth surface reflects light like a mirror. Less smooth surfaces reflect light over a wider range of angles (as the light hits the bumps in the microsurface), and therefore the reflections have less detail and are spread across the surface in a more diffuse way.

2. Modelos de Iluminação

Tabela de valores

UNITY TIME !!!! - Materiais e Shared



3. Renderização

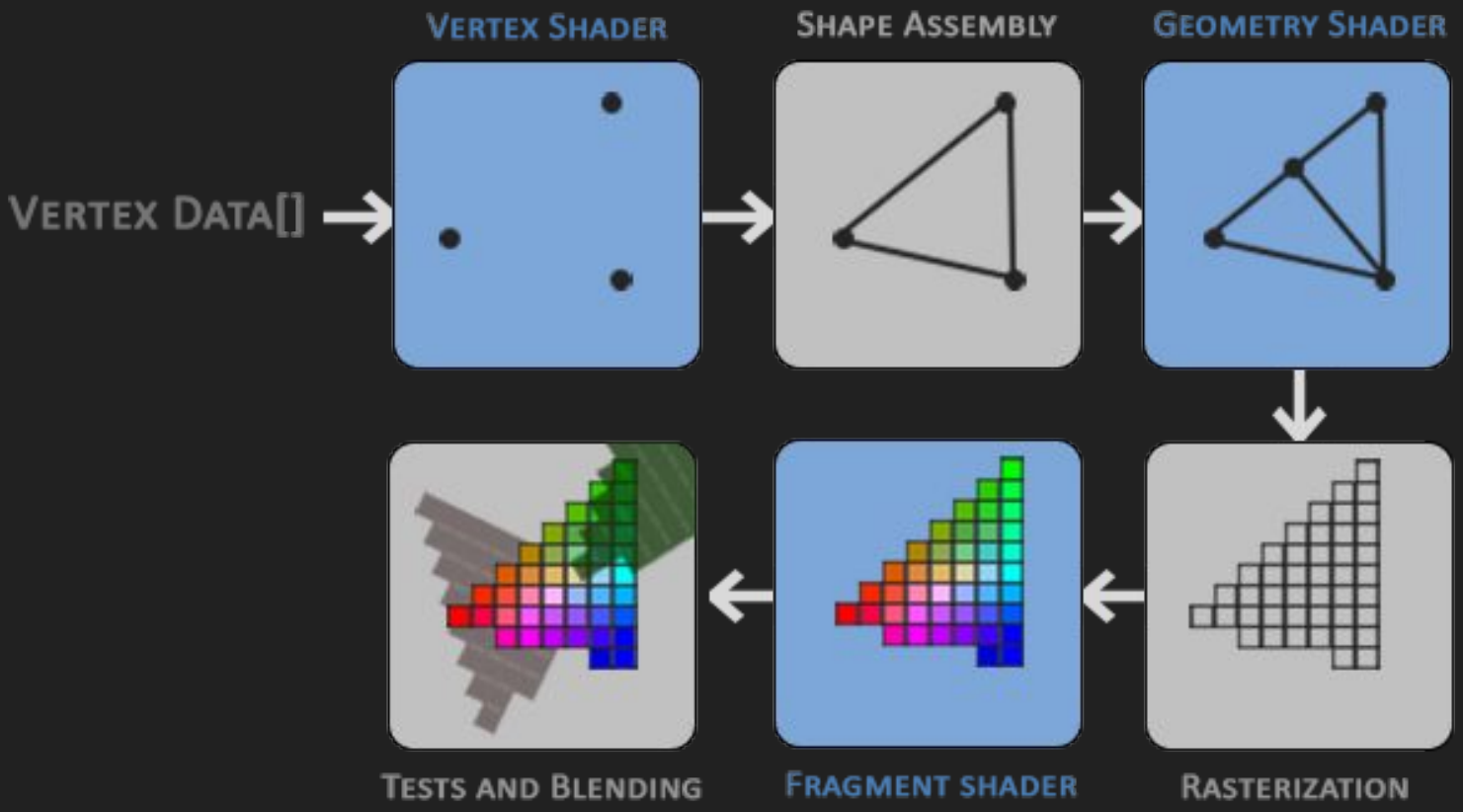
1. Introdução

- Shaders
  - São literalmente programas (instruções) que passamos para a placa de vídeo
  - GPU compila e guarda o programa na memória (limitações)
    - Por isso alguns jogos demoram para iniciar
  - Como tudo é paralelizado e independente, não temos muita informações de coisas vizinhas

3. Renderização

- São quatros tipos principais de shaders:
  - Vertex Shader
  - Geometry Shader
  - Tesselation Shader
  - Fragment/Pixel Shader

Pipeline de renderização

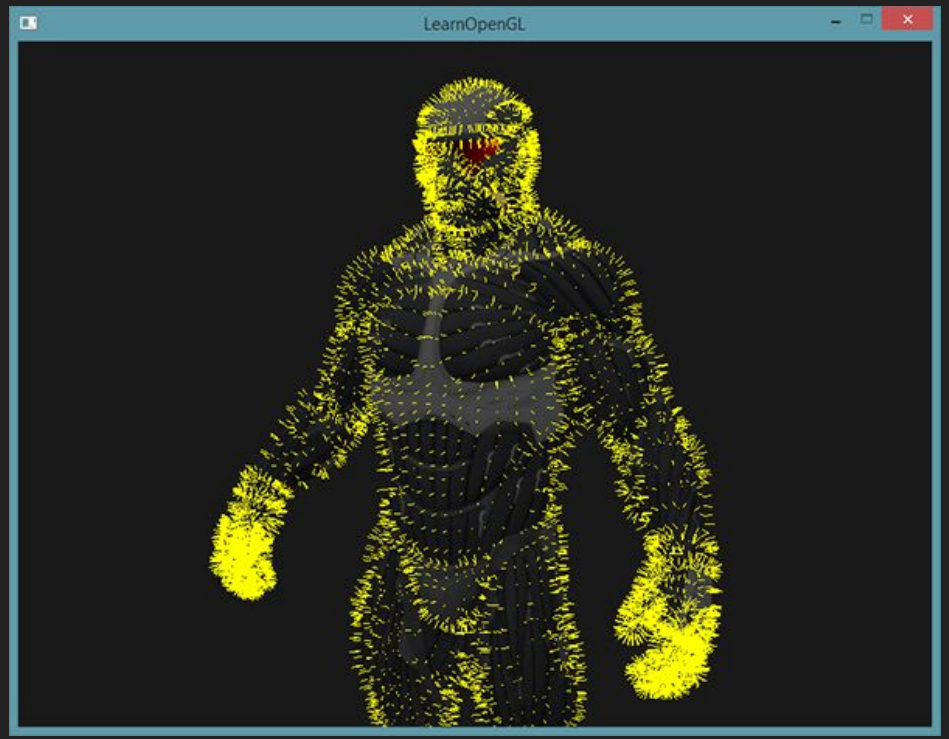


http://learnopengl.com/#!Getting-started/Hello-Triangle

3. Renderização

- Vertex Shader
  - Transformações de mundo
  - Matriz MVP (Model View Projection)
  - Mapeamento de coordenadas
- Geometry Shader
  - Altera os vértices
  - Cria novos vértices

Geometry Shader

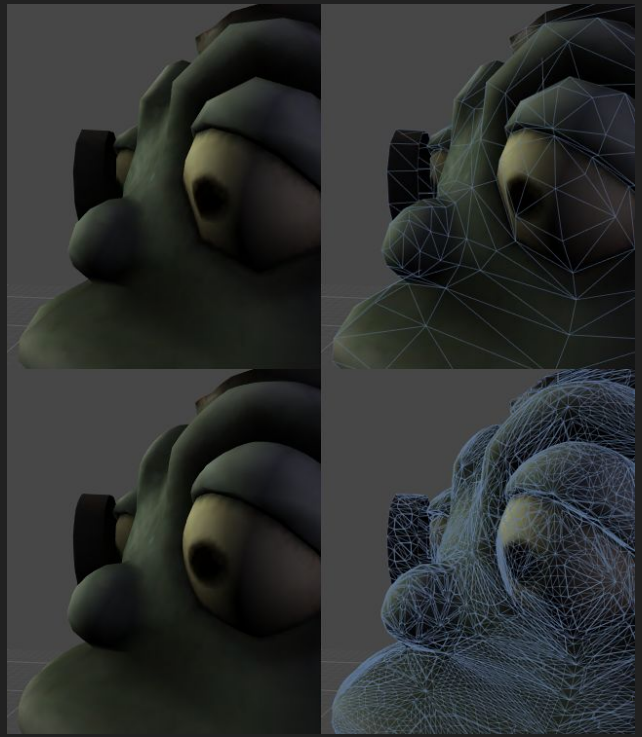


https://learnopengl.com/Advanced-OpenGL/Geometry-Shader

3. Renderização

- Tesselation Shader
  - Quebra a geometria
  - Adiciona detalhes
- Fragment Shader
  - Processamento paralelo
  - Interpolação dos vértices
  - Define a cor do pixel

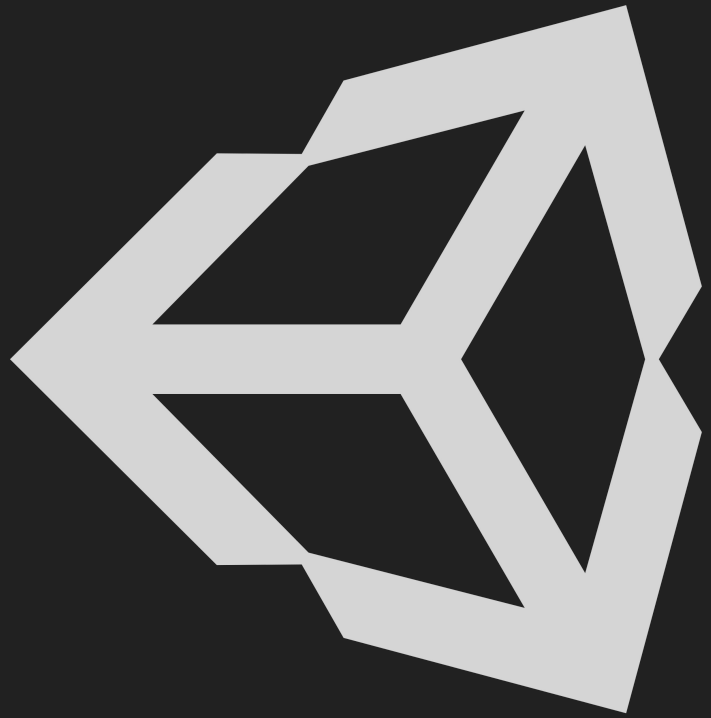
Geometry Shader



https://docs.unity3d.com/Manual/SL-SurfaceShaderTessellation.html

KODELIFE TIME !!!! - GLSL

UNITY TIME !!!! - Surface e Unlit



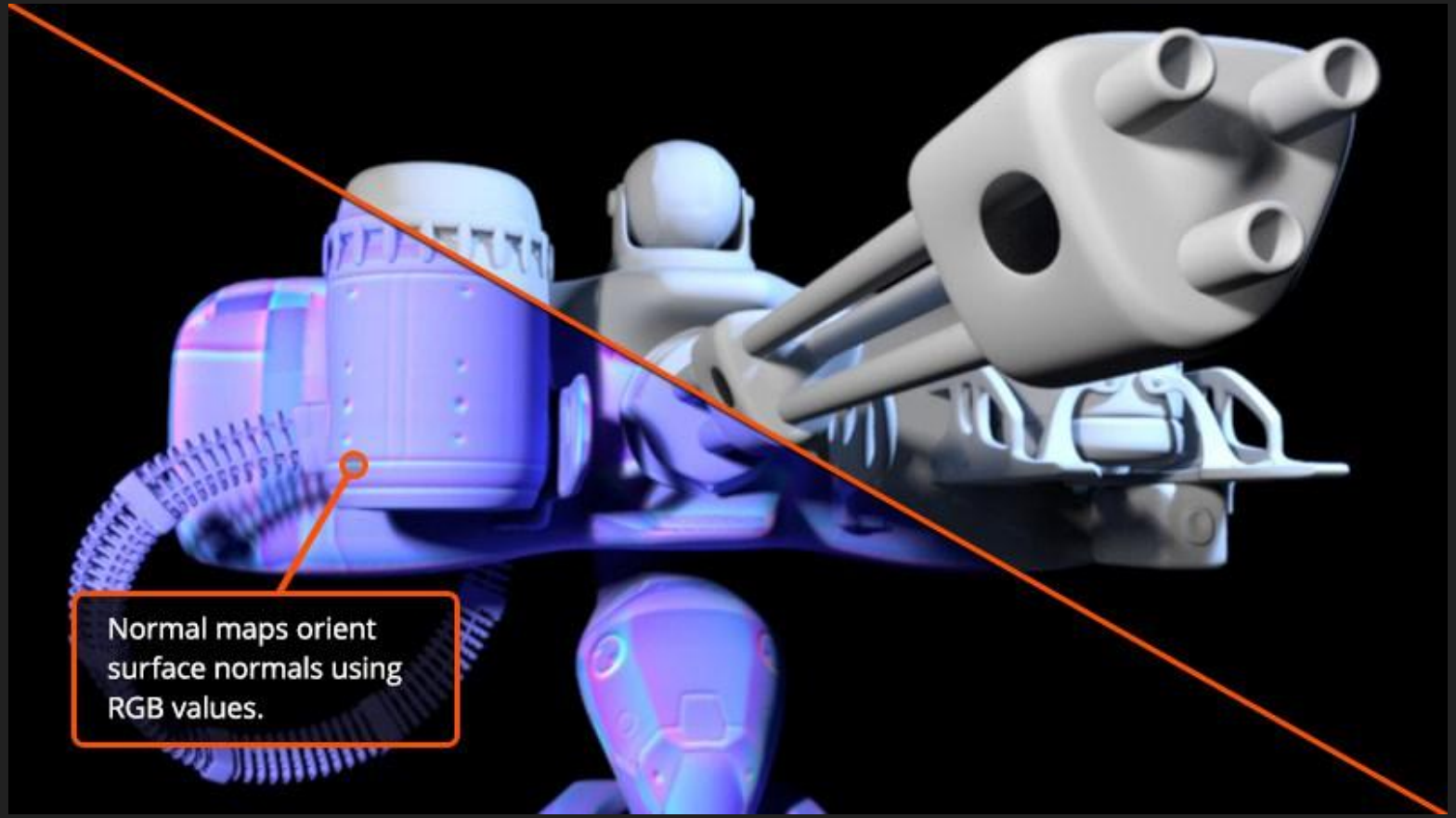
3. Renderização

- ➔ Mapeamentos (Mapping)
  - ◆ Texturas
  - ◆ Bump
    - Displacement
    - Normal
    - Parallax
    - Height
  - ◆ Cube
  - ◆ Shadow

3. Renderização

- ➔ Normal Map
  - ◆ Textura onde os pixels são vetores (r,g,b) => (x,y,z), usa o mesmo UV da textura
  - ◆ Altera o vetor normal no determinado pixel (fragmento), mudando a direção que a luz reflete e consequentemente a cor

Normal Map

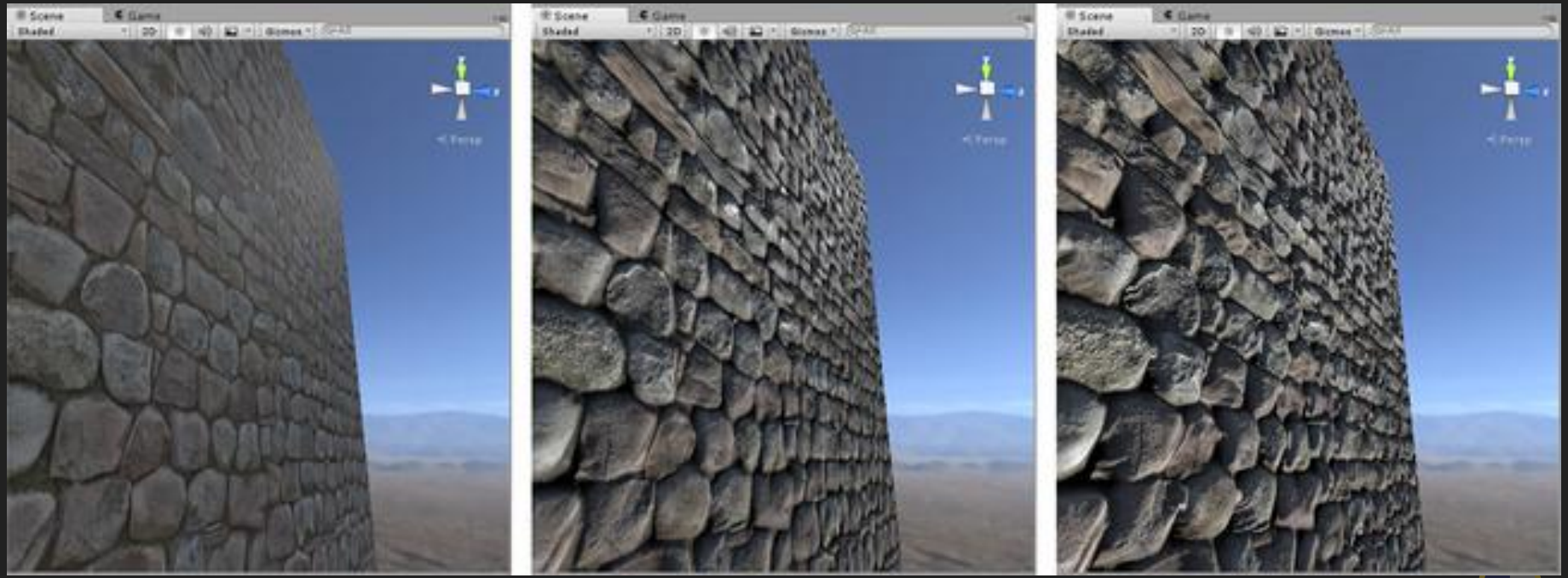


http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

3. Renderização

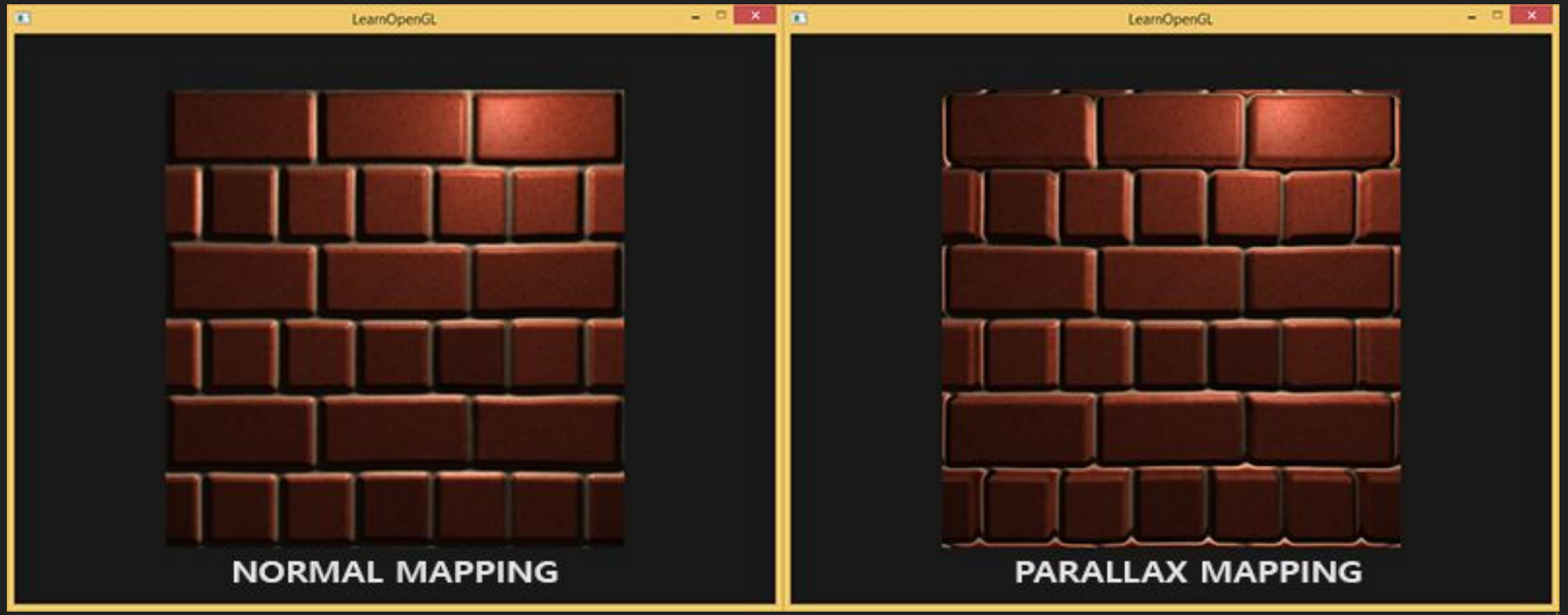
- Heigth ou Parallax map
  - Similar ao normal, mas mais complexo (e mais pesado)
  - Normalmente usado em conjunto com mapas normais
  - Move áreas da textura da superfície visível
    - Alcança um efeito a nível de superfície de oclusão
  - Protuberâncias terão suas partes próximas (frente à camera) exagerados, e a outra parte reduzida

Heightmap ou Parallax Map



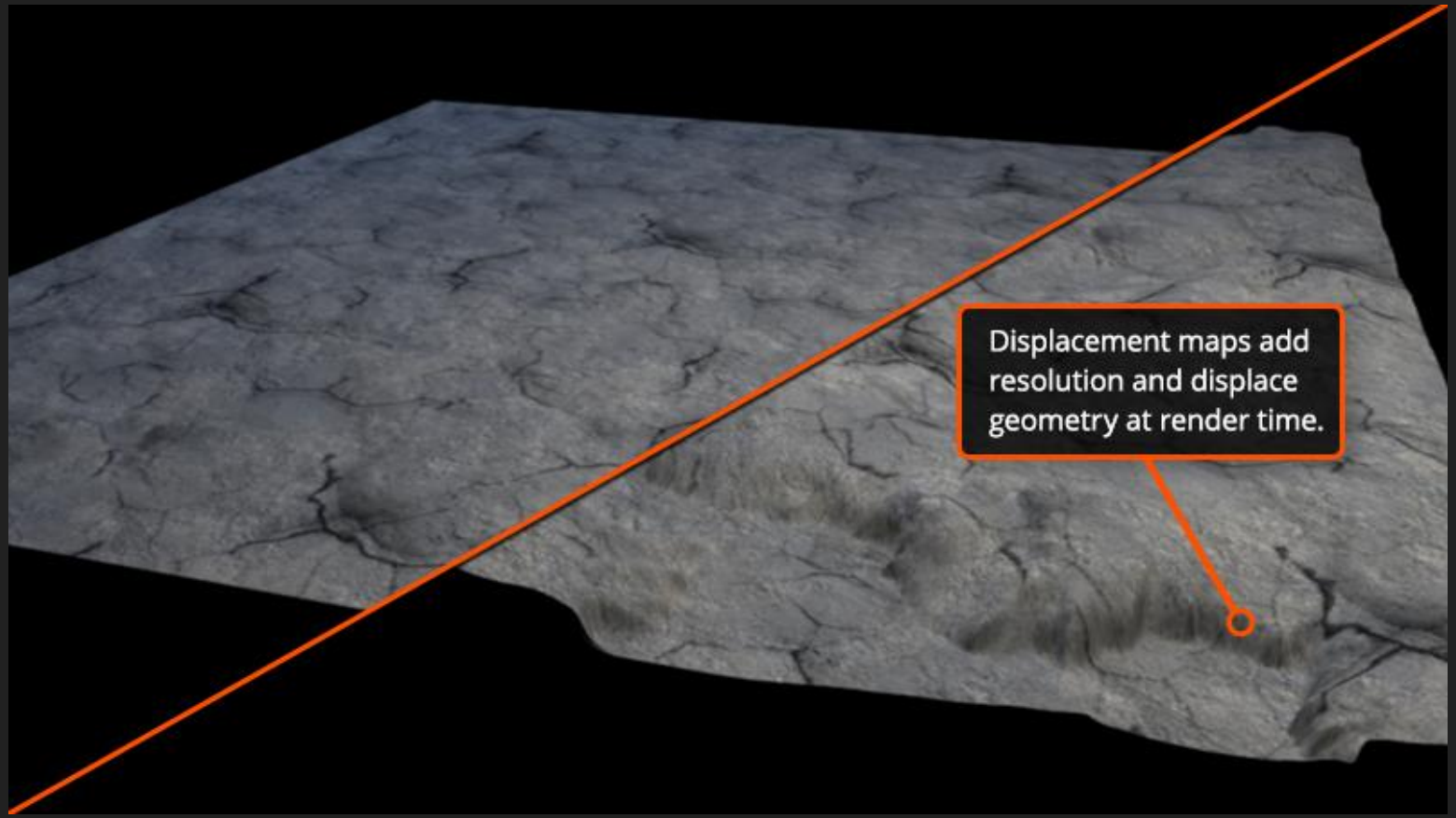
http://docs.unity3d.com/Manual/StandardShaderMaterialParameterHeightMap.html

Normal e Parallax Map



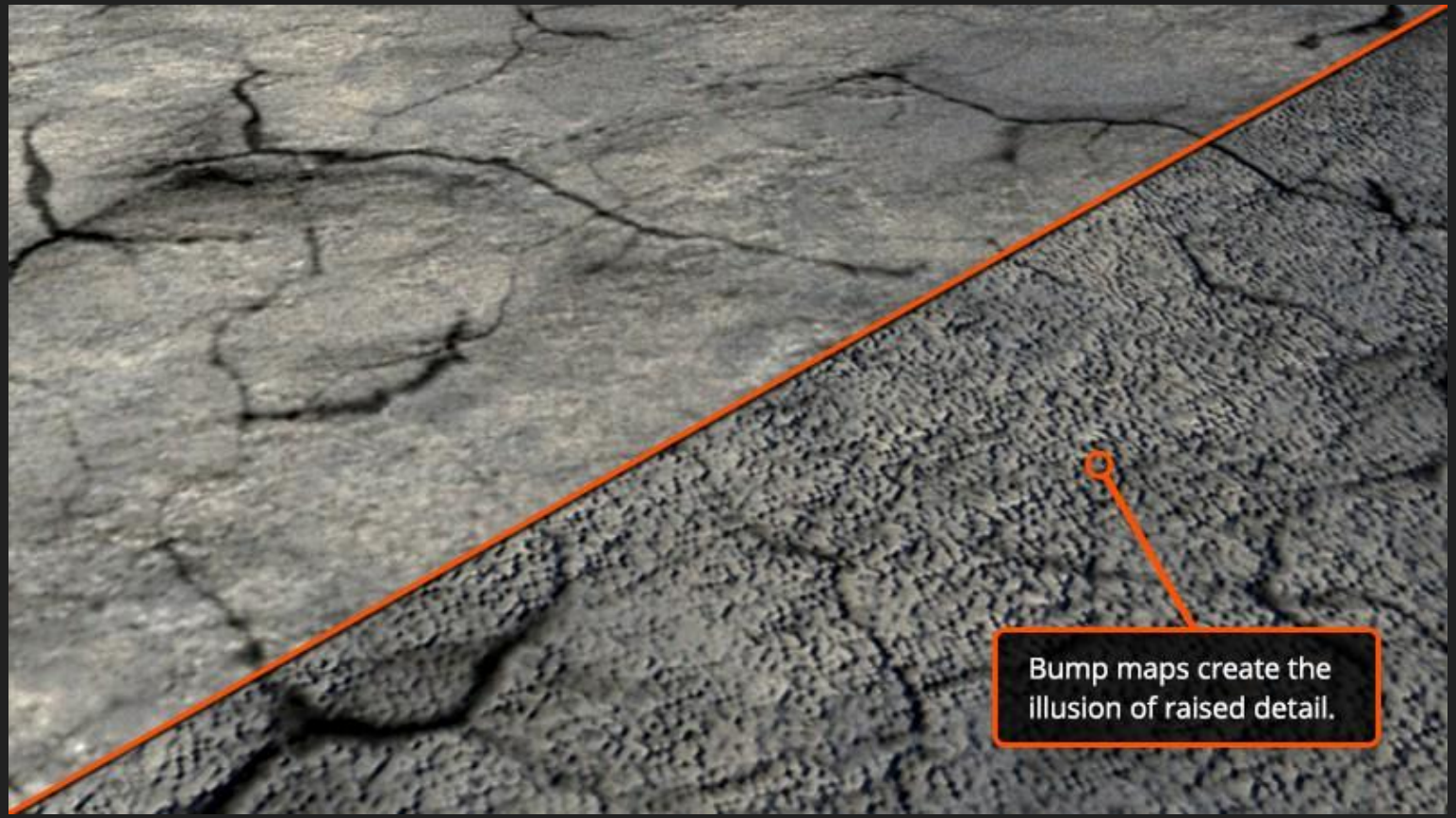
http://learnopengl.com/#!Advanced-Lighting/Parallax-Mapping

Displacement Map



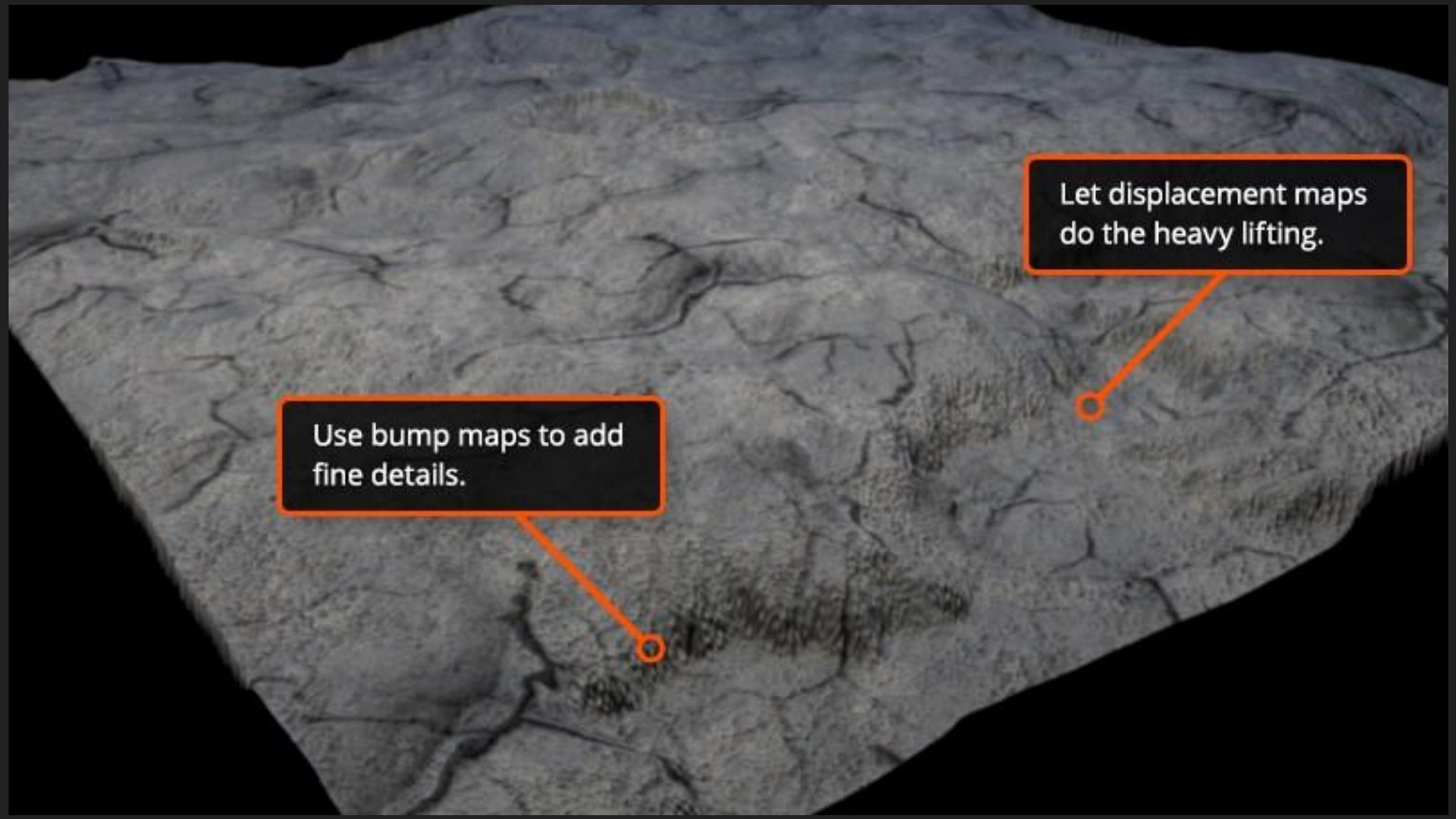
http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Bump Map



http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Bump e Displacement Maps



http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

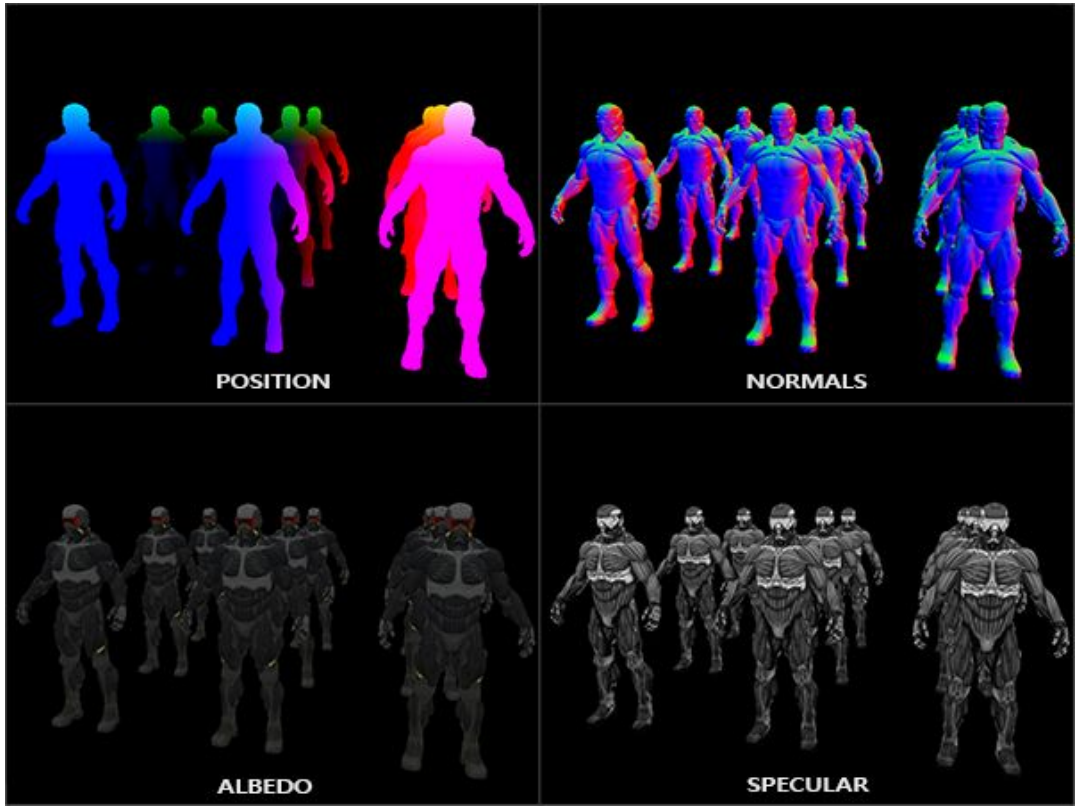
3. Renderização

- ➔ O problema dessa técnica é que precisamos fazer muitos cálculos de iluminação para cada fragmento, sendo que ele pode ser descartado no final (z-test)
- ➔ Até agora o que usamos foi “forward rendering”, existe outra técnica, chamada “deferred rendering” que se baseia em outros princípios

3. Renderização

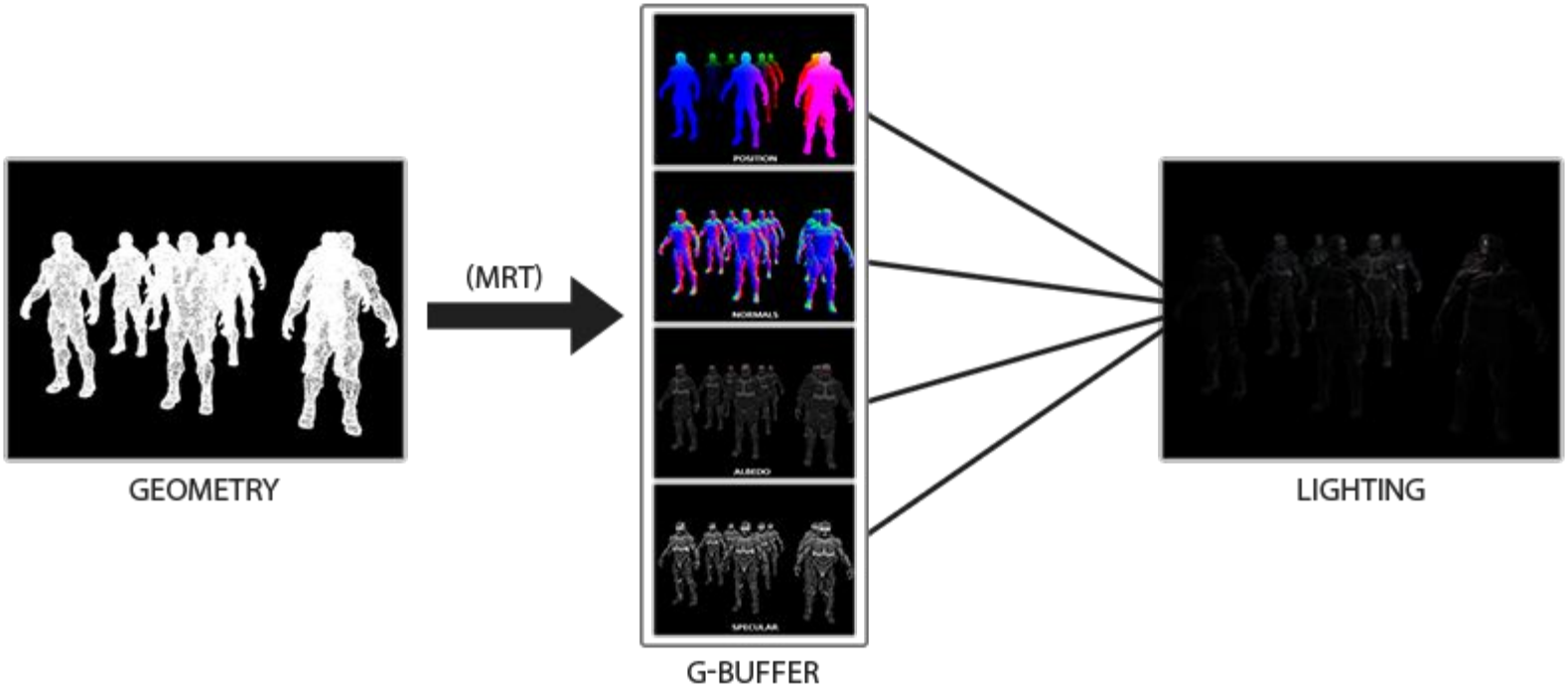
- Deferred Rendering
  - Invés de calcular a iluminação logo no começo, deixamos o trabalho pesado para o final (deferred)
  - Deixamos todas informações relevantes (posição, normal, cor e especular) em várias texturas
    - Multiple Render Targets (MRT)
  - Calculamos iluminação no passo final, juntando todos os buffers

G-buffer



https://learnopengl.com/Advanced-Lighting/Deferred-Shading

Deferred Rendering

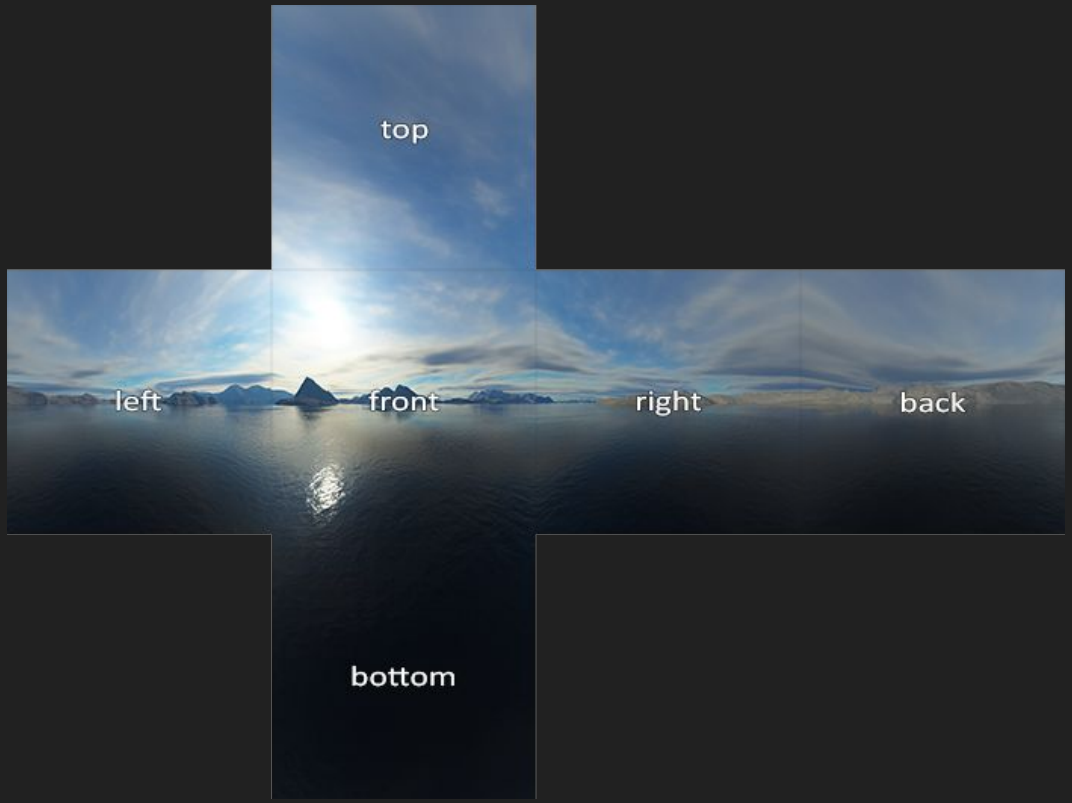


https://learnopengl.com/Advanced-Lighting/Deferred-Shading

3. Renderização

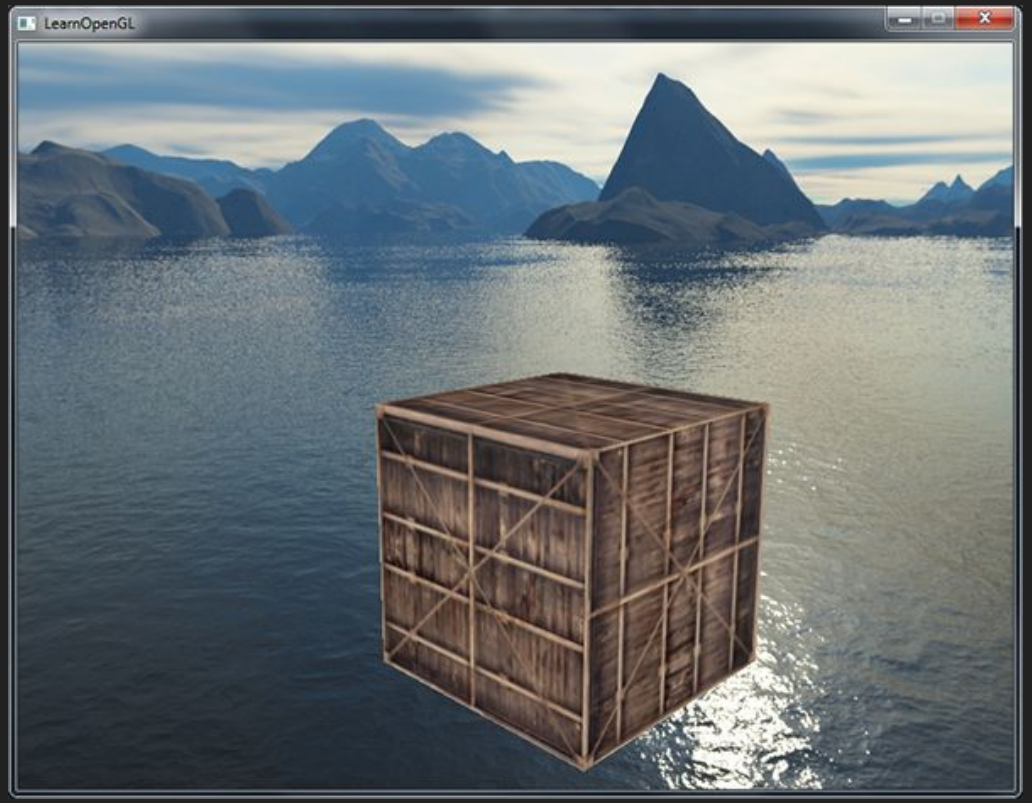
- Apesar de economizar processamento descartando o cálculo em fragmentos desnecessários, consumimos bastante memória
  - Não podemos usar algumas técnicas como MSAA
- Outros problemas incluem: GPU antigas, transparência, AA e sombras
- Funciona bem com iluminação dinâmica e escala bem com várias fontes, também podemos misturar com forward para resolver problemas de transparêcia

Cube Map



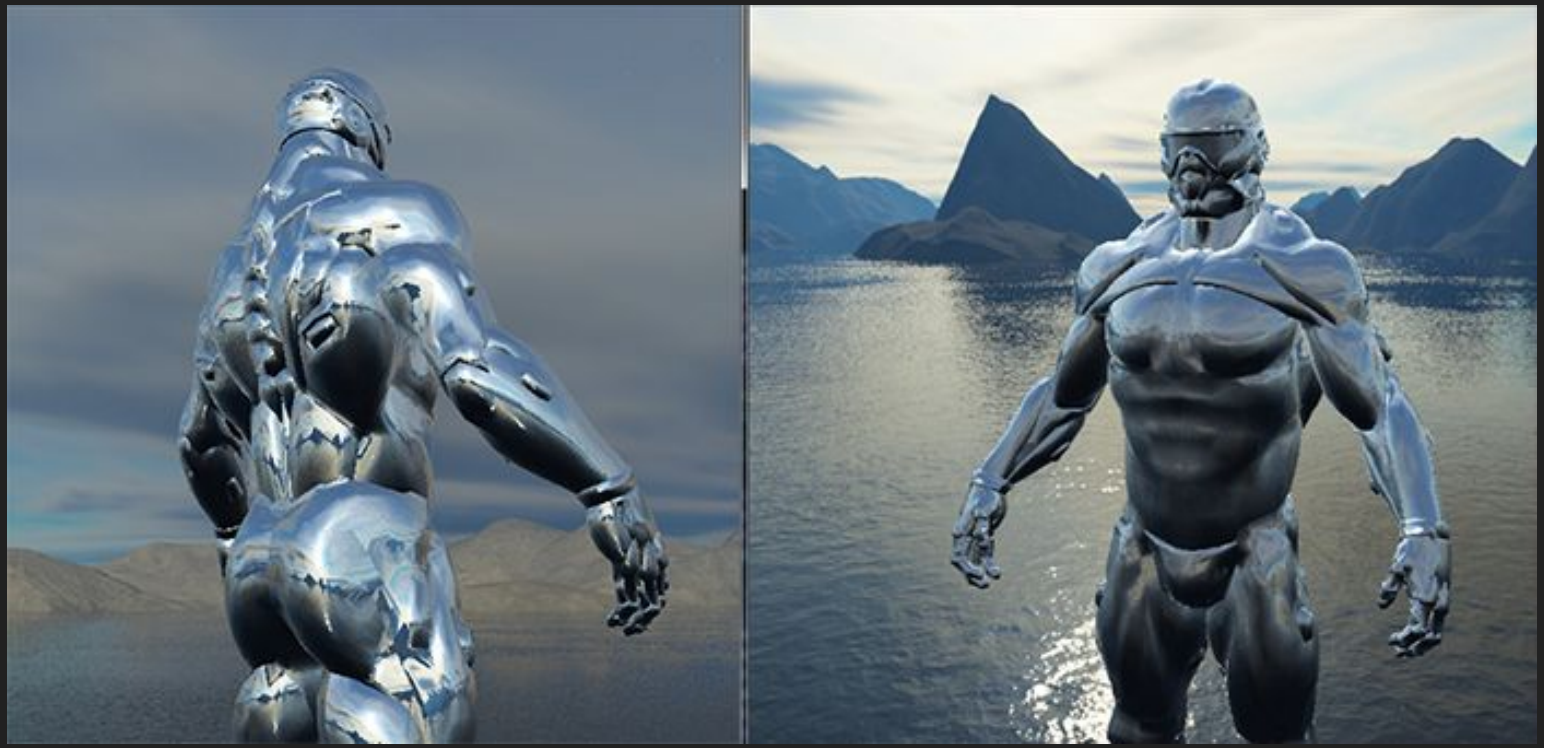
http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

Skybox



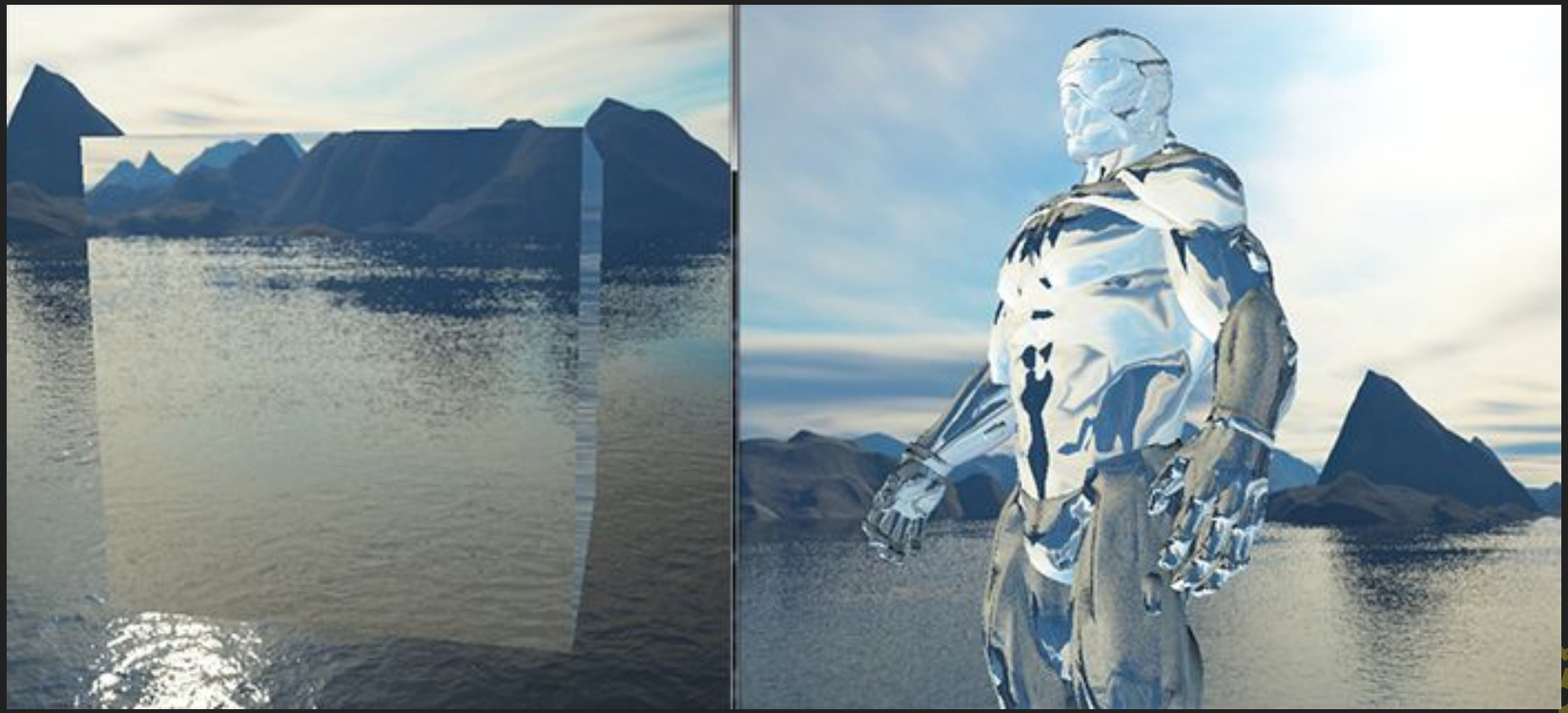
http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

Reflexão



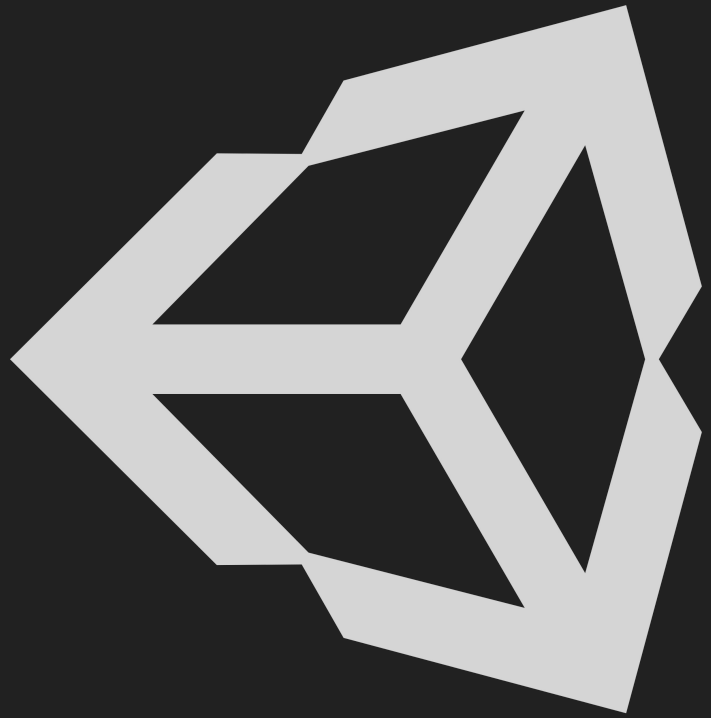
http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

Refração



http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

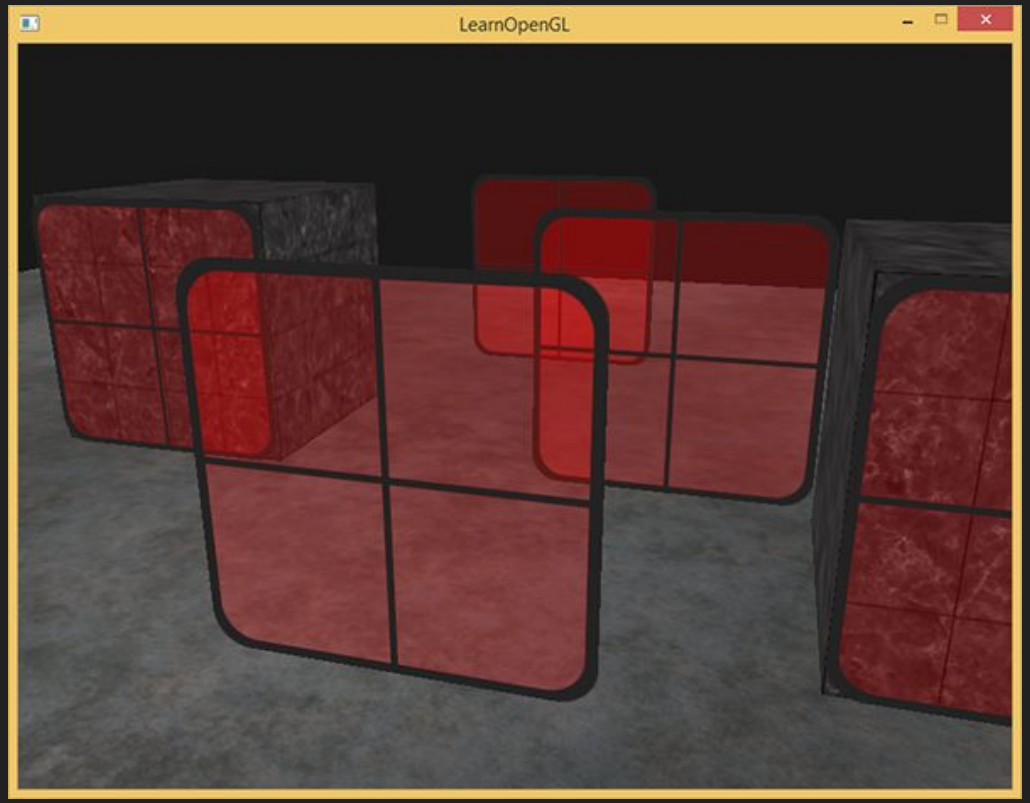
UNITY TIME !!!! - Skybox



3. Renderização

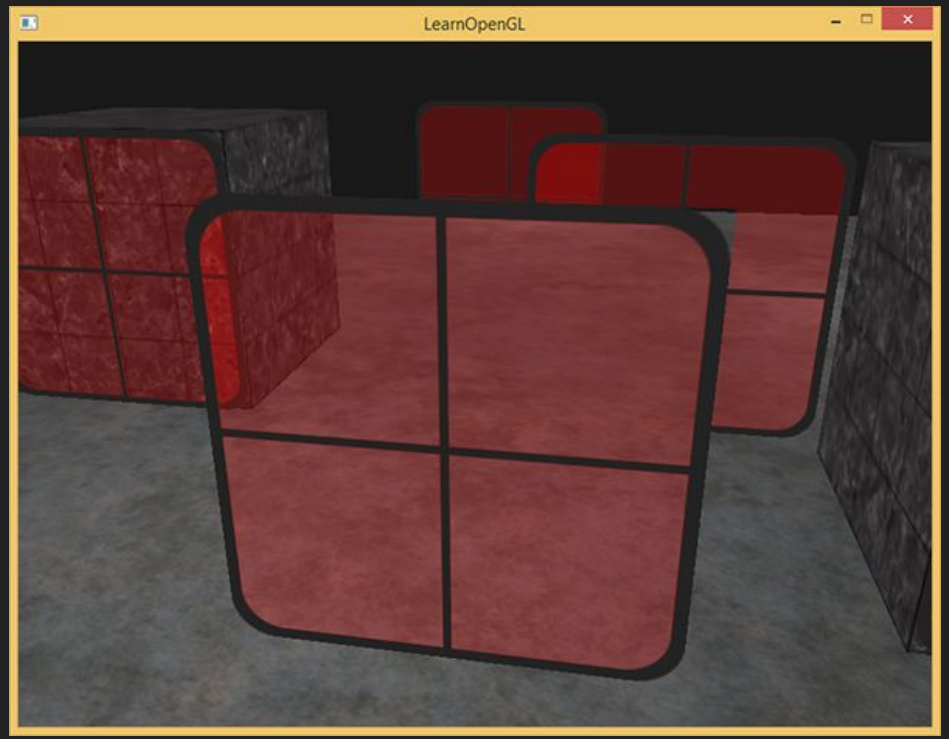
- Blending
  - Transparência e translucência
  - Aritmética entre canais alphas, então é preciso configurar a fórmula usada pelo shader e é preciso montar a ordem
  - A ordem sempre deve ser do último (atrás) para o primeiro (frente), tudo após os objetos opacos
    - Caso contrário, é possível que o pixel seja descartado no z-test

Blending



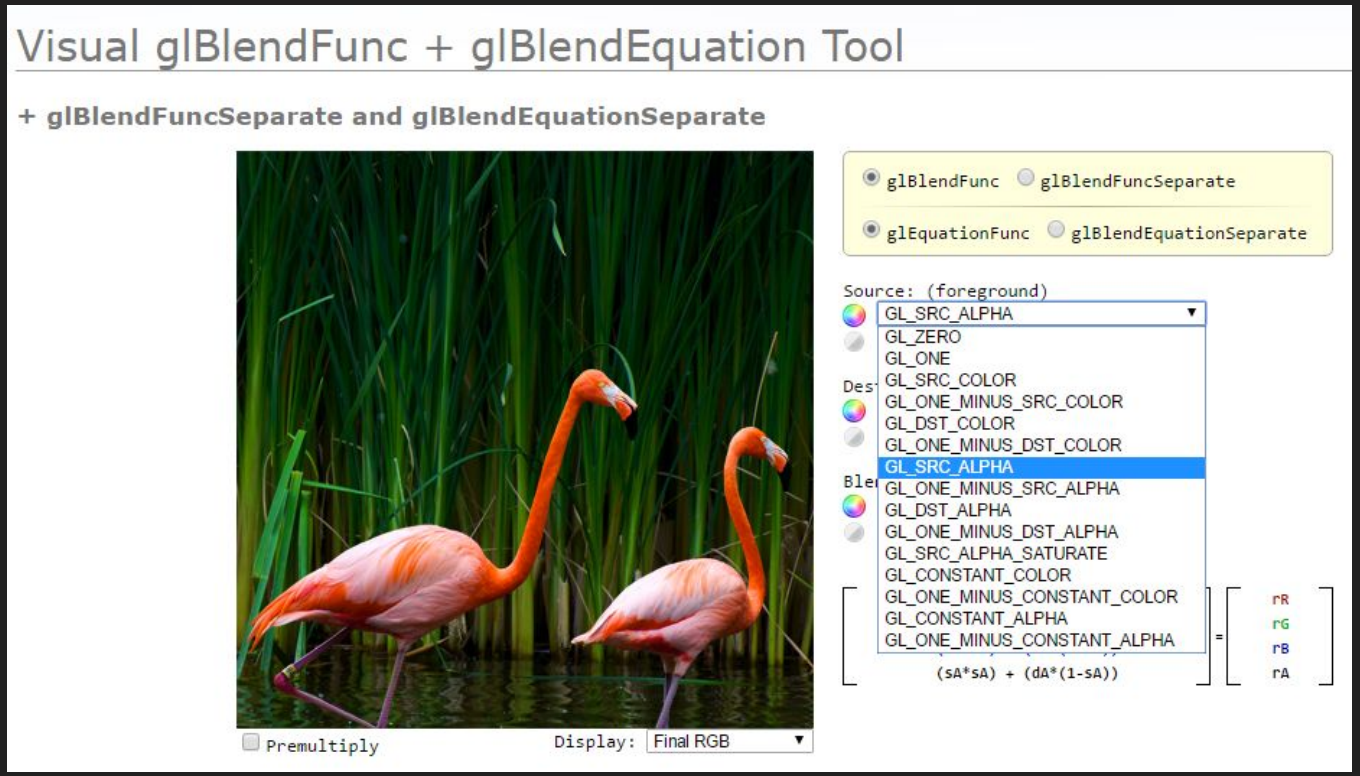
http://learnopengl.com/#!Advanced-OpenGL/Blending

Blending - Fora de ordem



http://learnopengl.com/#!Advanced-OpenGL/Blending

Blending

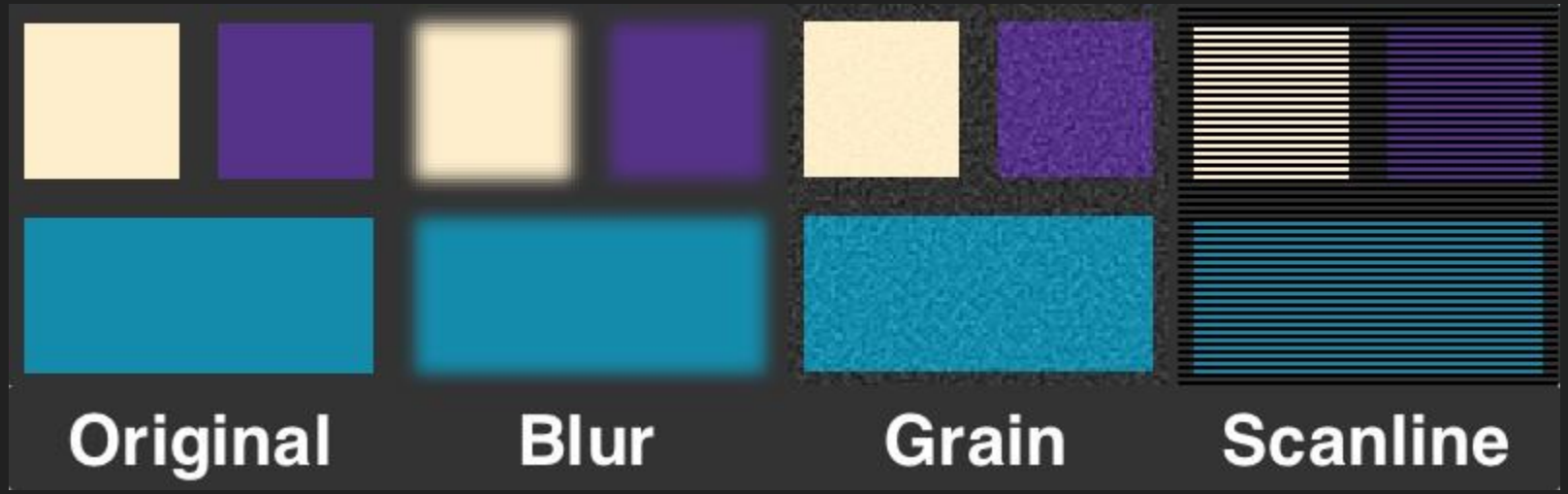


http://andersriggelsen.dk/glblendfunc.php

3. Renderização

- Post Processing
  - É capaz de gerar efeitos legais usando a imagem final
  - Efeitos bem similares a fotografia
  - A ideia é aplicar um shader no framebuffer (imagem final), tem médio custo, porém deixa a imagem mais realista

Post Processing



http://haxepunk.com/documentation/tutorials/post-process/

Bloom

http://haxepunk.com/documentation/tutorials/post-process/

HDR

http://gamesetwatch.com/

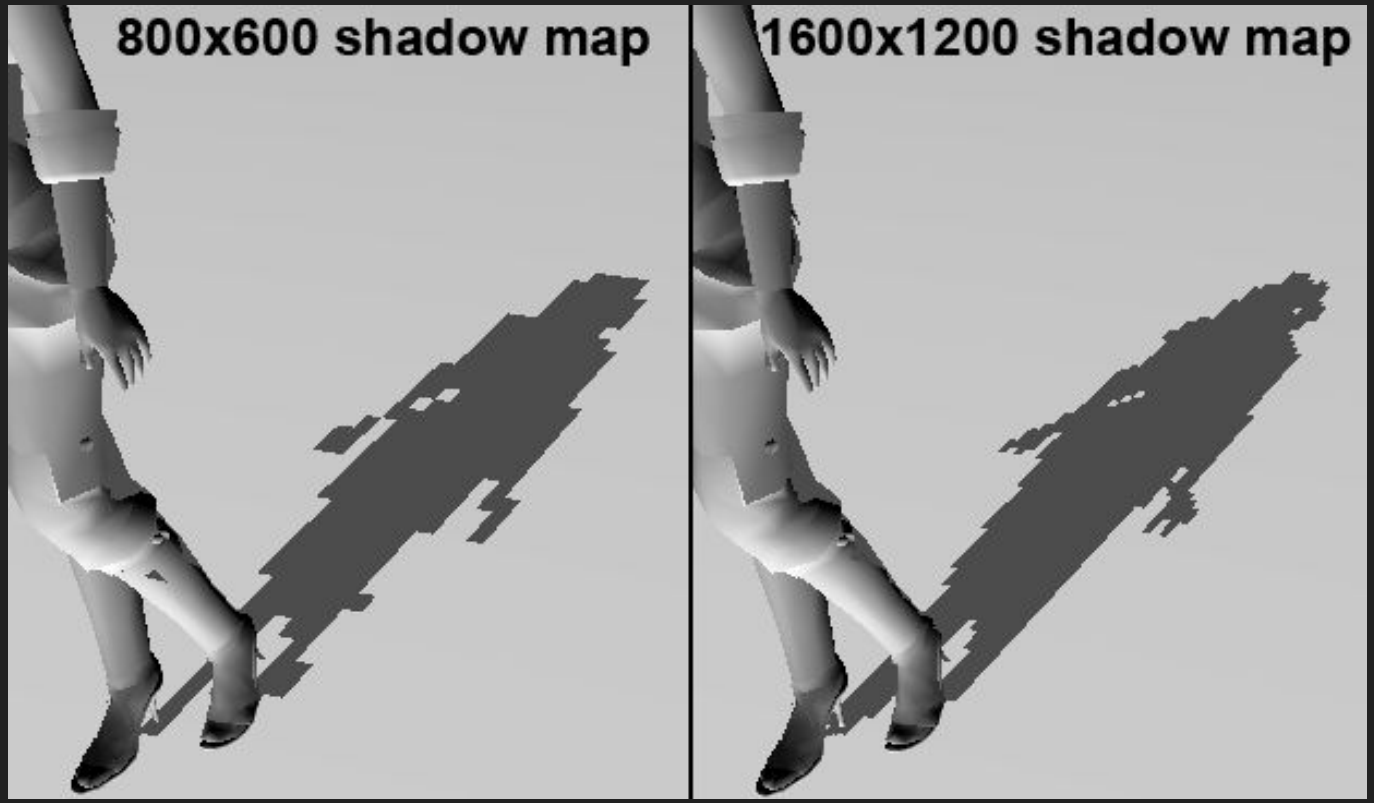
Depth of Field

https://steamcommunity.com/sharedfiles/filedetails/?id=134522361

3. Renderização

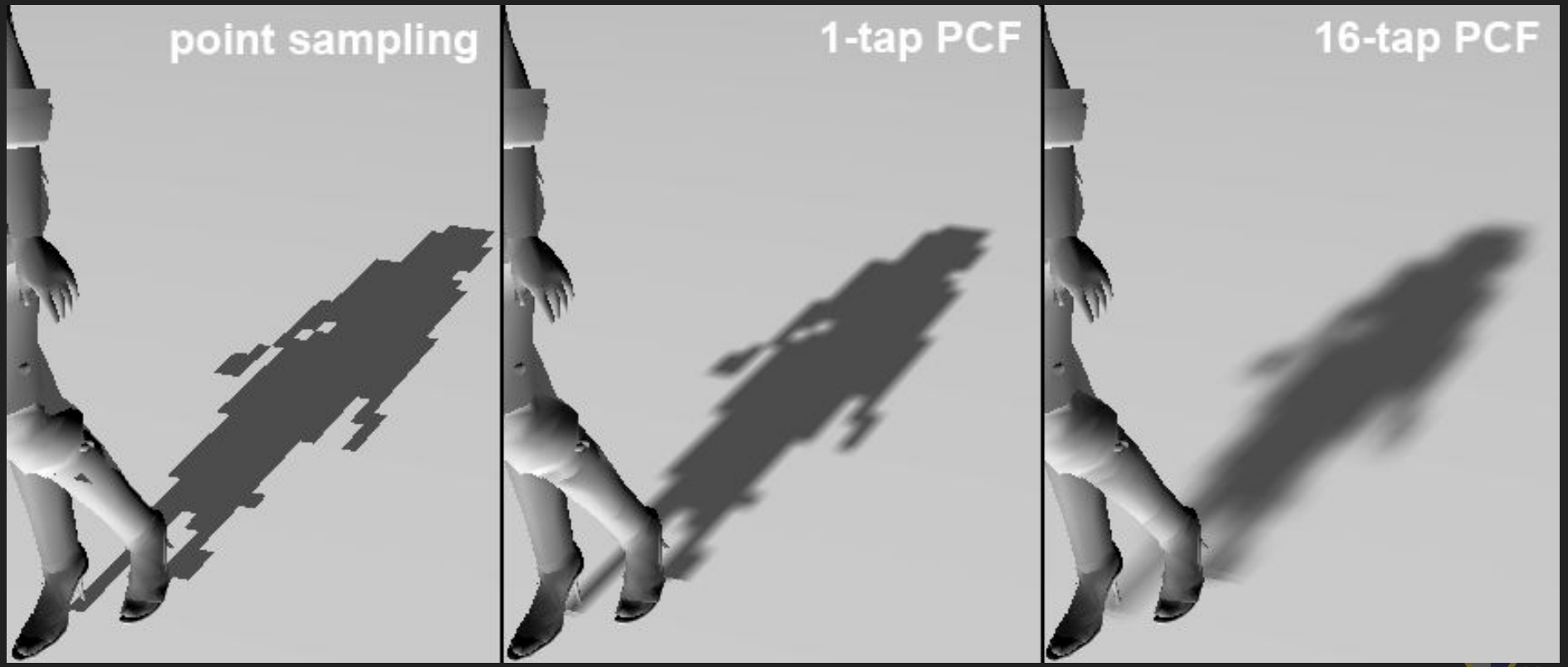
- Sombra (Shadow Map)
  - Tem alto custo na maioria dos casos e geralmente não serve para todos os tipos de fonte de luz
  - A ideia é fazer uma projeção dos objetos do ponto de vista da luz, mapear ele numa textura e usar a textura como sombra na imagem final (fazendo as transformações necessárias)
  - É uma aproximação, geralmente não tem resultados bons, por isso é feito algum filtro por cima (AA)

Shadow Map



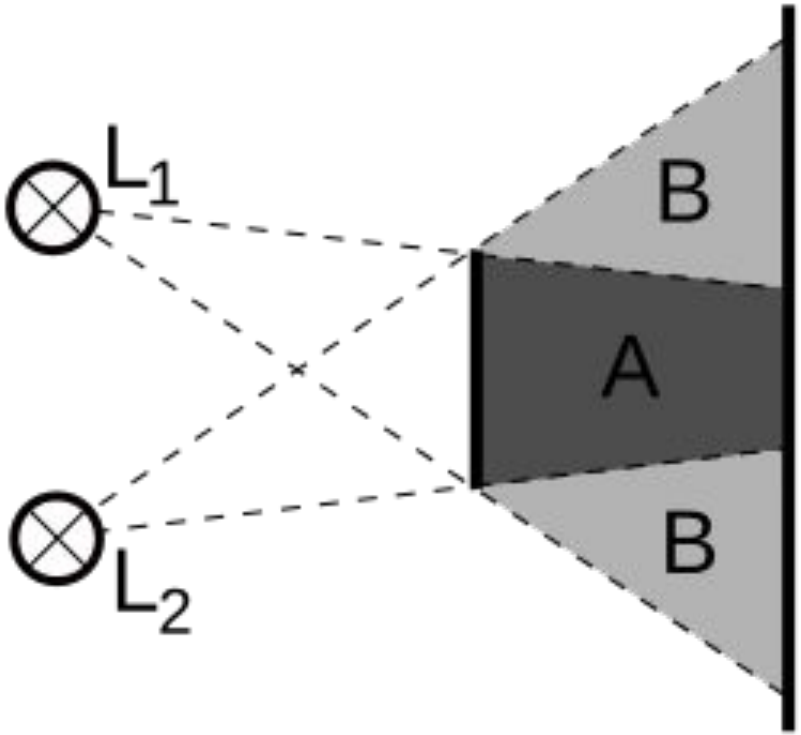
https://takinginitiative.wordpress.com/2011/05/25/directx10-tutorial-10-shadow-mapping-part-2/

Shadow Map



https://takinginitiative.wordpress.com/2011/05/25/directx10-tutorial-10-shadow-mapping-part-2/

Soft Shadow e Penumbra



https://en.wikipedia.org/wiki/Umbra,_penumbra_and_antumbra

3. Renderização

Exemplo com vários tópicos

3. Renderização

- Raytracing
  - Técnica de lançar vários raios saindo da câmera e dando “bounce” entre os objetos
  - Sombra e luz extremamente acurados, além de ser possível modelar transparência, reflexão e refração corretamente
  - Muito pesado, gera muito ruído (artefatos) e precisa de múltiplos raios por pixel

Ray tracing

https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

Ray tracing (RTX)

https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

Dúvidas?