2
Preliminares

2.1
Técnicas básicas de visibilidade

A determinação da visibilidade tem sido um problema fundamental em computação gráfica. Entre as técnicas de visibilidade mais conhecidas estão: back-face culling, view-frustum culling e occlusion culling. Faremos um breve comentário sobre cada uma dessas técnicas.

2.1.1
Back-face culling

Consiste em remover as faces ocultas de um objeto dada a posição do observador, evitando assim, o processamento desnecessário de faces que não são visíveis, como nos mostra a figura 2.1.

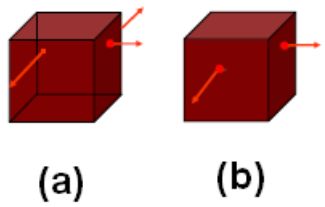

Figura 2.1: (a) imagem sem back-face culling (b) imagem utilizando o back face culling

A implementação do back-face culling consiste no cálculo do produto interno entre a normal da face em questão e a direção do observador. Seja N o vetor normal à face e V o vetor de visão (ver figura 2.2). Então, se:

- V.N = 0, os vetores são perpendiculares, e a face não é visível.
- V.N > 0, o ângulo entre os vetores é menor que 90 ° e a face está virada para trás e não é visível.
- V.N < 0, o ângulo entre os vetores é maior que 90 ° e a face está virada para frente.

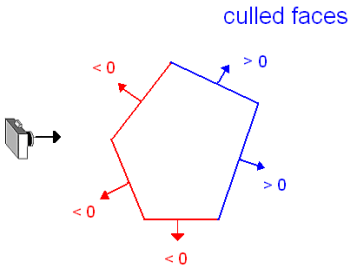

Figura 2.2: A face não é visível se o produto interno entre a normal da face e o vetor de visão for menor que zero.

A biblioteca do OpenGL possui uma opção para habilitar o back-face culling através do glEnable(GL_CULL_FACE).

2.1.2 View-frustum culling

Chamamos de viewing frustum o volume limitado entre a posição do observador e os vértices da janela em questão. Qualquer geometria que se encontrar fora deste volume estará fora do campo de visão do observador, logo, não será renderizada. Ver figura 2.3.

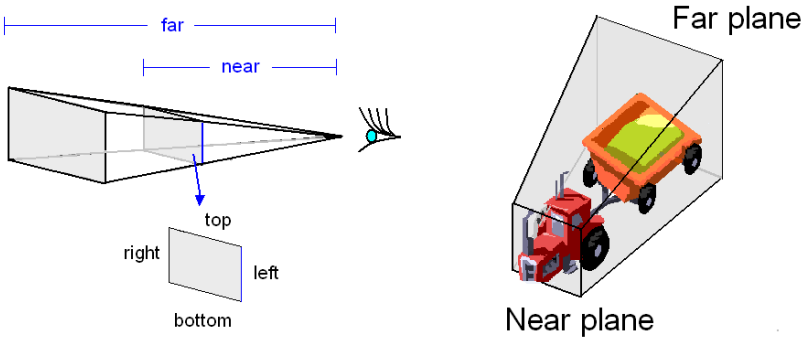

Figura 2.3: View frustum culling: qualquer parte do objeto fora do volume de visualização não é visível.

2.1.3 Occlusion culling

Outra técnica que evita cálculos desnecessários é o occlusion culling, cujo objetivo é não renderizar a geometria escondida por outro objeto. Na figura 2.4, o poliedro oculta parcialmente a visualização do cubo.

A figura 2.5 ilustra o back-face, view-frustum e occlusion culling.

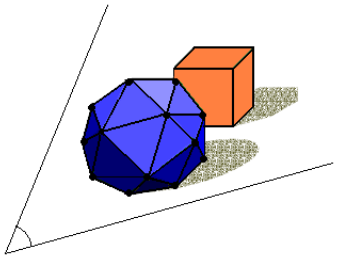

Figura 2.4: Occlusion culling: remove a geometria escondida por outro objeto.

2.2 Critérios para algoritmos de visibilidade

Nesta seção discutimos alguns critérios utilizados na classificação das principais técnicas de visibilidade.

2.2.1 Ponto versus Região

- *from point versus from region* – O algoritmo *from point* calcula a visibilidade considerando o observador posicionado em um único ponto, enquanto que o *from region* calcula a visibilidade que seja válida em qualquer ponto de uma certa região.

2.2.2 Espaço objeto versus Espaço imagem

- *object space versus image space* – Os algoritmos *object space* utilizam objetos em si para o cálculo da visibilidade. Por outro lado, os algoritmos *image space* operam na representação discreta de objetos quando fragmentados durante o processo de rasterização. Em geral, os algoritmos de imagem (*image space*) são mais eficientes e funcionam normalmente por aproximação, já os algoritmos de objeto (*object space*) possuem solução exata pois verificam a parte do objeto que está sendo encoberta por outro objeto qualquer.

2.2.3 Conservativo versus Aproximado

- *consevative versus aproximate* – o algoritmo *conservativo* é aquele que superestima o conjunto visível. Partes do objeto que não são visíveis são classificados como visíveis. O algoritmo *aproximado* calcula a visibilidade aproximada e não garante encontrar todas as primitivas visíveis possíveis.

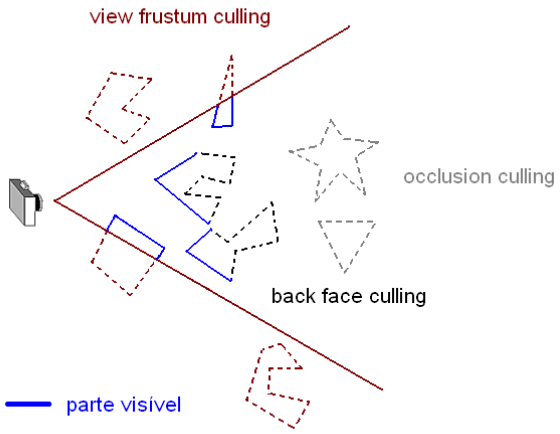

Figura 2.5: 3 técnicas de remoção de áreas escondidas: back face culling, view frustum culling e occlusion culling.

2.3 Recursos do OpenGL

O OpenGL (GL Graphics Library) é um sistema gráfico desenvolvido pela Silicon Graphics voltada para aplicações gráficas 3D.

Entre os vários recursos do OpenGL utilizados na implementação deste trabalho, destacamos dois que são fundamentais na geração do ocluder implícito: depth buffer e stencil buffer.

Depth buffer

O depth buffer é responsável pelo armazenamento da profundidade de cada pixel. Este valor varia de 0 a 1, onde o zero indica que o objeto está próximo do observador e 1 indica que o objeto está no infinito. É usado para o occlusion culling. O depth buffer também é conhecido como z-buffer.

A função glDepthFunc(GLenum func) especifica a função usada para comparar o valor z de cada pixel com o valor z presente no depth buffer. O default é GL_LESS, que significa que um fragmento passa pelo teste quando seu valor em z for menor do que o valor já armazenado no depth buffer. Outros parâmetros são GL_NEVER, GL_ALWAYS, GL_LEQUAL, GL_EQUAL, GL_NOTEQUAL e GL_GREATER. Tal comparação só é feita se o teste de profundidade estiver ativado pelo glEnable(GL_DEPTH_TEST).

Stencil buffer

Utilizamos o stencil buffer para restringir uma determinada porção da tela para que objetos não sejam desenhados, assim como um pintor utiliza um estêncil de papelão com spray para pintar uma superfície. Tal mascaramento

de regiões é realizado marcando os pixels que devem ou não ser desenhados por 1 ou 0.

O stencil buffer é ativado pela função glEnable(GL_STENCIL_TEST), e o teste do stencil é feito através da função glStencilFunc() que possui 3 parâmetros: (GLenum func, GLint ref, GLuint mask). O primeiro parâmetro pode ser: GL_NEVER, GL_ALWAYS, GL_LESS, GL_LEQUAL, GL_EQUAL, GL_GEQUAL, GL_GREATER, ou GL_NOTEQUAL. O teste glStencilFunc() realiza a comparação (func) do valor de referência (ref & mask) com o valor no stencil buffer (stencil & mask). Ver comentários na seção 4.5.2.

A função glStencilOp() especifica como o conteúdo da máscara do stencil buffer será modificado quando um fragmento passar ou falhar no teste do stencil.Tem como entrada de dados: (GLenum fail, GLenum zfail, GLenum zpass). O primeiro parâmetro indica o que acontece quando o teste do stencil falha. O segundo é se o teste do stencil não falhar, mas o z-buffer falhar, e por fim, o terceiro parâmetro indica se os 2 testes (do stencil e do z-buffer) não falharem. Os 3 parâmetros podem ser: GL_KEEP, GL_ZERO, GL_REPLACE GL_INCR, GL_DECR, ou GL_INVERT, onde:

- GL_KEEP - mantém o atual conteúdo do stencil para esse fragmento.
- GL_ZERO - zera o conteúdo para esse fragmento.
- GL_REPLACE - Troca o conteúdo do stencil para o valor indicado no campo ref do glStencilFunc.
- GL_DECR - decrementa o conteúdo.