Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação

IA725 – Computação Gráfica
Professora: Wu Shin Ting

Trabalho Final

VISIBILIDADE EM COMPUTAÇÃO GRÁFICA

Alunos: Elias Yauri RA: 100300

Natasha Nakashima RA: 108753

Campinas/2011

Sumário

Resumo

O objetivo da computação gráfica é criar imagens sintéticas de cenas virtuais através da simulação da propagação da luz. Na renderização de cenas um dos problemas cruciais a resolver é a visibilidade. Esta técnica permite visualizar cenas mais reais com reduzido tempo de processamento. A visibilidade é parte integral da interação da luz com o meio ambiente.

Os primeiros algoritmos procuravam determinar as linhas e superfícies visíveis em uma cena, eliminado as superfícies ocultas. Eles foram desenvolvidos para operar sobre os dispositivos vetoriais existentes naquela época. Com o nascimento dos dispositivos rasters, estes algoritmos foram substituídos por uma nova técnica: o algoritmo de Z-Buffer. Atualmente, existem duas abordagens algorítmicas de visibilidade. A primeira, o Z-Buffer, é utilizado para determinar as superfícies visíveis, sendo amplamente utilizado para renderizações em tempo real. A segunda, geralmente utilizada para o cálculo da iluminação global, é o traçado de raios, o qual calcula a visibilidade na direção de um raio projetado. Entretanto, além destas duas técnicas, existe uma ampla variedade de algoritmos de visibilidade para problemas específicos.

Neste trabalho será apresentado um histórico dos algoritmos de visibilidade, suas classificações, uma descrição dos principais algoritmos utilizados e a implementação de duas principais técnicas atuais: o Z-Buffer em OpenGL e o traçado de raio em PovRay.

Introdução

Evolução Histórica

Desde o surgimento da computação gráfica, o problema de visibilidade é um fator crucial na síntese de imagens. A visibilidade é um fenômeno que está diretamente ligado a interação das fontes de luz da cena com o meio ambiente que se deseja visualizar.

Os primeiros algoritmos que surgiram com esta finalidade, em meados dos anos 60, procuraram determinar as linhas e superfícies que seriam visíveis em uma cena 2D e 3D. Este problema foi categorizado como determinação de linhas ou superfícies visíveis (visible line/ visible surface) ou remoção de linhas ou superfícies escondidas (hidden line/ suface removal).

Em 1974, Sutherland et al. apresentou como os algoritmos de visibilidade poderiam tomar vantagens do conceito chamado de coerência (Foley 1996), a qual entende-se como sendo o grau de similaridade entre uma parte de um ambiente e sua projeção, aumentando, assim, a eficiência destas técnicas.

Até este período, grande parte dos algoritmos haviam sido desenvolvidos para dispositivos vetorias. Entretanto, nesta mesma época, houve um crescimento na disponibilidade de dispositivos raster, fazendo com que estes fossem substituidos pelo algoritmo de Z-Buffer (Catmull, A subdivision algorithm for computer display of curved surfaces. 1974).

A partir de então, o algoritmo Z-Buffer passou a ser a principal forma para determinar a visibilidade em computação gráfica, devido a facilidade de incorporação deste com outras técnicas de visibilidade como Binary Space-Partitioning Tree (BSP) e Scanline.

Atualmente, além do Z-Buffer, algorimos que determinam a visibilidade de um objeto a partir de um raio, Ray-tracing, também estão sendo utilizados. O primeiro trabalho nesta área, desenvolvido por Appel em 1968, utilizava raios imaginários que saem da câmera sintética para os objetos da cena 3D para determinar as superficies visíveis na mesma (Foley 1996).

Assim, os algoritmos de determinação de superfícies visíveis podem ser classificados, de acordo com suas abordagens do problema, em duas classes distintas, como pode ser visto a seguir.

Classificação dos algoritmos de visibilidade

De acordo com Foley (Foley 1996), os algoritmos de visibilidade podem ser classificados em dois grandes grupos: visibilidade orientada a imagem ou precisão de imagem e visibilidade orientada ao objeto ou precisão do objeto.

No primeiro caso, quer-se determinar quais os *n* objetos são visiveis em cada pixel da imagem final. O Custo computacional para este caso é de *np*, sendo *n* o número de objetos na cena e *p* o numero de pixels da imagem, pois, para cada pixel, deve-se verificar qual entre todos os objetos está mais próximo do observador ao longo do raio projector. Exemplo da utilização desta abordagem são os algoritmo Z-Buffer e o Ray-tracing. O pseudocódigo para esta abordagem pode ser visto a seguir.

```
for (each pixel in the image){

        Determine the object closest to the viewer that is pierced by the projector through the
pixel;

        Draw the pixel in the appropriate color;
}
```

Enquanto que no segundo caso, é feita uma comparação entre cada objeto da cena, eliminando todo ou parte do mesmo que não esteja visível. Neste caso, tem-se um custo computacional de n^2. Um exemplo desta abordagem é o algoritmo do Pintor. A seguir, pode-se ver o pseudo-código desta abordagem.

```
for (each object in the world){

        Determine those parts of the object whose view is unobstructed by other parts of it or
any other object;

        Draw the pixel in the appropriate color;
}
```

Pré-processamento

Em cada uma das abordagens para determinacão de superficies visíveis, tem-se um grande número de operações custosas para o computador, como o cálculo de intersecções entre os objetos da cena ou de suas projeções e onde elas acontecem e a deterinação de qual objeto está mais próximo do observador.

Para minimizar o tempo gasto para geração destas imagens, algumas técnicas podem ser utilizadas para aumentar a eficiência destas operações, como a utilização do conceito de coerência, a transformação de perspectiva, caixas limitantes (Bounding Volumes), Back-face culling, subdivisão especial e hierarquia.

Coerência

Como foi dito anteriormente, o conceito de coerência foi incorporado à determinacão de superficies visíveis em 1974, por Sutherland et al. Ele constatou que no ambiente, os objetos geralmente variam suas propriedades de forma suave. Isto permite que cálculos feitos para uma determinada região da imagem ou do ambiente sejam reaproveitados em áreas próximas. Vários tipos de coerência foram identificados:

- **Coerência de objetos:** se um objeto está inteiramente separado de outro objeto (objetos disjuntos), estes não podem apresentar intersecções a nível de faces e arestas.
- **Coerência de face:** as propriedades gráficas de uma face variam de forma suave.
- **Coerência de arestas:** a visibilidade de uma aresta só é alterada quando a mesma cruza atrás de uma aresta visível ou penetra uma face visível.
- **Coerência geométrica:** o segmento de intersecção entre duas faces planas pode ser determinado a partir de dois pontos de intersecção.
- **Coerência das linhas de varredura:** Entre duas linhas de varredura subsequentes, há uma pequena variação de suas propriedades geométricas e físicas.
- **Coerência na área de cobertura:** um grupo de pixels adjacentes geralmente é coberto pela mesma face visível.
- **Coerência na profundidade:** o valor de profundidade entre duas partes adjacentes de uma face apresenta pouca variação e diferentes superfícies que estão localizadas na mesma posição são separadas por valores diferentes de profundidade.
- **Coerência nos quadros:** dois quadros consecutivos de uma animação diferem muito pouco um do outro.

Transformação de perspectiva

Independentemente do tipo de projeção, paralela ou perspectiva, em que se escolha trabalhar, para saber se um ponto $P1(x1, y1, z1)$ sobrepoe-se em relação a um ponto $P2(x2, y2, z2)$, pode-se traçar um raio a partir do centro de projeção e verificar se estes

pertencem ao mesmo. Caso isto aconteça, basta fazer uma comparação entre os seus valores de z para saber qual ponto está mais próximo do observador.

Assim, em uma projeção paralela se x1 = x2 e y1 = y2, então os pontos estão no mesmo raio projetado. Entretanto, em uma projeção paralela, para que dois pontos pertençam ao mesmo raio, precisa-se fazer quarto divisões:

$$\frac{x1}{z1} = \frac{x2}{z2} \qquad\qquad \frac{y1}{z1} = \frac{y2}{z2}$$

Estas divisões podem ser evitadas transformando as coordenadas do objeto 3D para o sistema de coordenadas da câmera, fazendo com que o teste para verificar se um objeto sobrepõe-se a outro seja igual ao teste feito para a projeção paralela.

A matriz M transforma o volume de visão normalizado da projeção em perspectiva em um paralelepípedo limitado por $-1 \leq x \leq 1, -1 \leq y \leq 1\ e\ -1 \leq z \leq 1$.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+zmin} & \frac{-zmin}{1+zmin} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Caixa limitante (Bounding Volumes)

A caixa limitante tem como objetivo simplificar os testes de sobreposição de objetos na cena. Ele consistem em determinar um volume limitante justo ao objeto, sendo este volume geralmente o menor cubo que envolva o objeto, com planos paralelos aos eixos do sistema de referência, como mostra a figura abaixo:

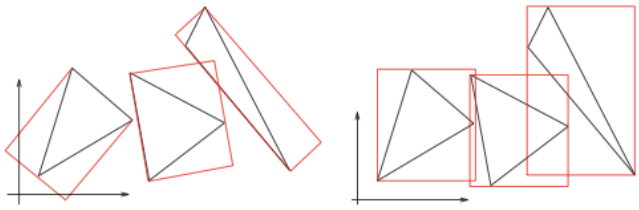

Figure 1: Caixas limitantes não paralelas e paralelas aos eixos de referência, respectivamente.

Back-face Culling

A técnica de back-face culling tem como objetivo remover as faces do objeto que estão "de costas" para o observador. Isto é feito através da verificação do ângulo formado entre o vetor normal da superfície do objeto $n = (nu, nv, nn, 0)$ e um raio de visao $V = (0,0,-1,0)$ no espaço normalizado. O ângulo é dado pela seguinte equação:

$$Bf = n\,.V = -nn$$

Se $Bf > 0\ ou\ nn < 0$, a face está “de costas“ para o observador, portanto não estando visível. Caso contrario, ela é potencialmente visível, como mostra a figura abaixo:

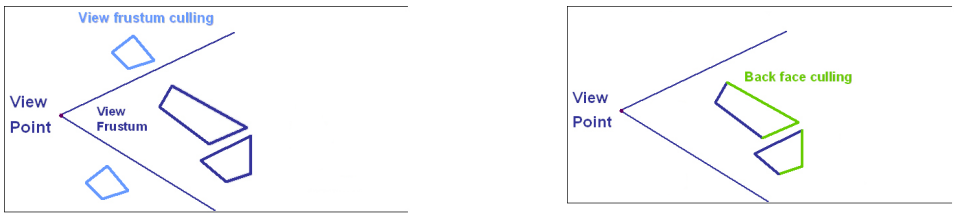

Figure 2: Exemplo de Back-face Culling

Particionamento espacial

Esta técnica de pré-processamento tem como abordagem básica a separação coerente dos objetos da cena em grupos. Uma forma de aplicá-la e dividir o plano de projeção em uma grade regular e realizar a verificação de sobreposição apenas nos objetos que estiverem dentro de cada pedaço da grade, ou seja, primeiro determina-se em qual parte da grade o raio de projeção intercepta e depois testar apenas os objetos desta partição.

Hierarquia

Uma estrutura hierarquica aninhada, onde cada nó filho é considerado como parte do nó pai, pode ser utilizada para diminuir o número de comparações necessárias para determinar superficies visíveis em uma cena. Neste caso, se não houver intersecção entre dois objetos da hierarquia, seus nós filhos não precisarão ser testados.

Principais algoritmos de visibilidade

Algoritmo do Buffer de Profundidade (Z-Buffer)

O algoritmo de Z-Buffer, desenvolvido por Catmull (Catmull, A subdivision algorithm for computer display of curved surfaces. 1974) é um algoritmo para determinação de superfícies visíveis centrado na resolução da janela de visualização.

Na implementação do algoritmo, cria-se dois espaços de memória: o frame buffer (F), onde esta armazenada as cores computadas a partir da cena, e o buffer de profundidade (Z-Buffer, Z), da mesma resolução que o frame buffer, contendo a profundidade (valor de Z) para cada pixel.

O Z-Buffer (que trabalha nas coordenadas da janela do dispositivo) é inicializado com o valor de 1 (sua variação é de zero até um), o qual representa o valor de Z no plano atrás do volume de visão, enquanto que, o Frame Buffer será inicializado com a cor do fundo da cena. A figura 3 apresenta uma cena e seu correspondente mapa de profundidade.

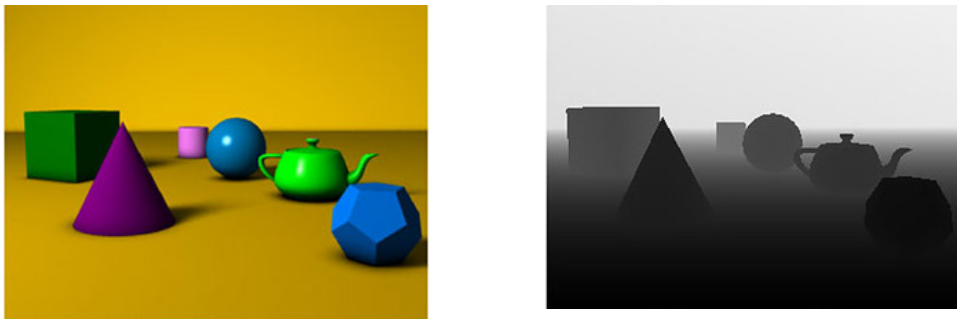

Figure 3: Cena 3D a) Imagem colorida. b) Mapa de profundidade da imagem a.

Na versão mais simples do algoritmo, os objetos são processados sobre o frame buffer em qualquer ordem, um a um. Durante este processo de conversão de um ponto do polígono para a coordenada (x,y), se o valor de sua profundidade é mais próxima para o observador, então a cor daquele ponto e sua profundidade substituirão o último valor do frame buffer e buffer de profundidade, respectivamente.

O pseudocódigo do algoritmo de z-buffer apresenta-se na figura 4.

```
void zBuffer() {
  for (int y = 0; y < YMAX; y++)//clear the frame buffer
        for (int x = 0; x < XMAX; x++) {
              writePixel(x, y, BACKGROUND_VALUE);
              writeZ(x, y, 1);
        }
  for (int i = 1; i <= NUM_POLYGON; i++) {//for all polygon in any order
        pz = z_value_of_polygon_at_pixel_coord(x, y);
        if (pz >= readZ(x, y)) { //new points is near
              writeZ(x, y, pz);
              writePixel(x, y, color_of_polygon_at_pixel_coord(x, y));
        }
  }
}
```

Figure 4: Pseudocódigo do algoritmo de z-buffer

Algumas das características básicas do algoritmo são:

- Não é necessário nenhum pré-ordenamento ou comparação objeto a objeto. Assim, os polígonos serão apresentados na tela segundo a ordem em que foram processados.
- O processo é simplesmente encontrar o menor valor Z_i sobre um conjunto de pares $\{Z_i(x,y), F_i(x,y)\}$ para valores fixos de x e y. A figura 5 exemplifica a adição de dois polígonos a imagem.

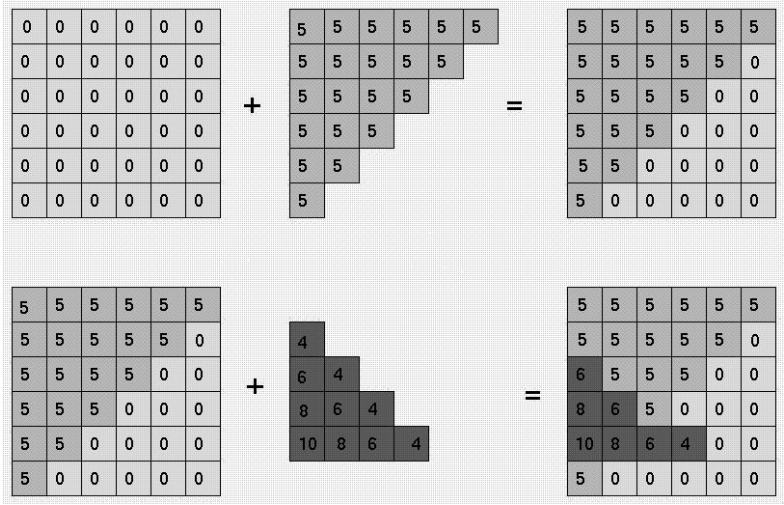

Figure 5: O Z-Buffer. A cor do pixel é apresentada em cores e seu valor Z é representado pelo número. a) Adição ao z-buffer de um polígono com valor constante de profundidade. b) Adicionando outro polígono que intersecta o primeiro.

O algoritmo de Z-Buffer não exige que os objetos sejam polígonos, seu maior atrativo é que pode ser utilizado para renderizar qualquer objeto se sua cor e valor z podem ser determinados para cada ponto na sua projeção. Nenhum algoritmo de interseção é necessário.

Baseado na **coerência de profundidade**, o cálculo da profundidade z para cada ponto pode ser simplificado numa linha de varredura aproveitando o fato de que o polígono é planar. Normalmente, calcular o valor z equivale a resolver a equação do plano: Ax + By + Cz + D=0 para z:

$$z = \frac{-D - Ax - By}{C}$$

Se, avaliamos a equação anterior para z_i, então para (x+dx) o novo valor de z é

$$z = z_i - (A/C)(dx)$$

É necessária somente uma subtração para calcular z(x+1,y), dado o valor de z(x,y), o cociente A/C for constante e dx = 1. Da mesa forma pode ser realizado o cálculo do z na próxima linha de varredura, decrementando B/C para cada dy. Se o polígono não for planar, o valor de z(x,y) pode ser calculado interpolando os valores z dos vértices do polígono ao longo de um par de arestas, e depois ao longo de cada linha de varredura, como se apresenta na figura 6.

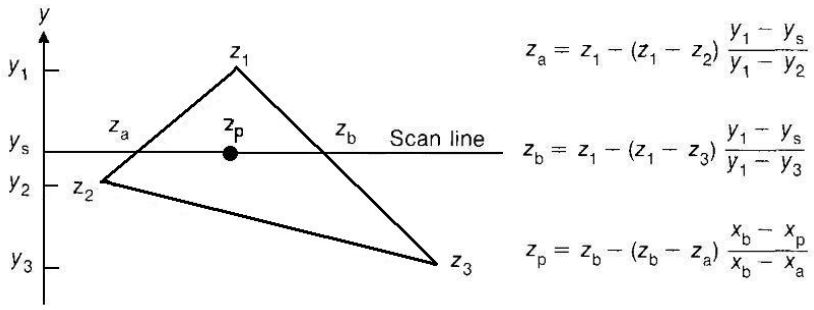

Figure 6: Interpolação de z ao longo das arestas de um polígono e uma linha de varredura. z_a é interpolado entre z_1 e z_2; z_b, entre z_1 e z_3, z_p entre z_a e z_b.

Embora o algoritmo de Z-Buffer requeira grandes quantidades de memória, sua simplicidade e carência de estruturas de dados adicionais, facilitam sua implementação em software e hardware (firmware).

Entretanto, devido a sua natureza centrada na imagem, ele pode sofrer o fenômeno de a aliasing, no caso em que o Z-Buffer possuir precisão limitada. O Z-Buffer é frequentemente implementado em hardware com inteiros de 16 ou 32 bits; em software, se utiliza valores de ponto flutuante.

Depois que a imagem foi renderizada, o z-buffer ainda não perdeu sua utilidade. Seus valores podem ser salvos e posteriormente utilizados. Um exemplo é o cursor 3D que se movimenta ao redor de uma imagem em x, y e z para selecionar um objeto o região da cena (o clássico “picking”).

Algoritmos de lista de prioridade

Os algoritmos baseados em listas de prioridades determinam a visibilidade ordenando os objetos de uma cena, garantindo que uma imagem correta será gerada se os objetos forem renderizados em tal ordem. Por exemplo, para objetos não sobreposicionados bastará somente ordenar seus valores de z de forma decrescente e renderizar os objetos em tal ordem. Objetos afastados serão obscurecidos pelos objetos mais próximos, os pixels dos objetos próximos sobrescrevem os pixels dos objetos afastados.

Se os objetos estão sobreposicionados em z, ainda é possível ordená-los, como no caso da figura 7-a. Se os objetos estão sobreposicionados ciclicamente um sobre outro, figuras 7b e 7c, ou penetram um deles, então não há forma de determinar a ordem certa. Nesse casso, será necessário partir um ou mais objetos para fazer um ordenamento linear de ser possível.

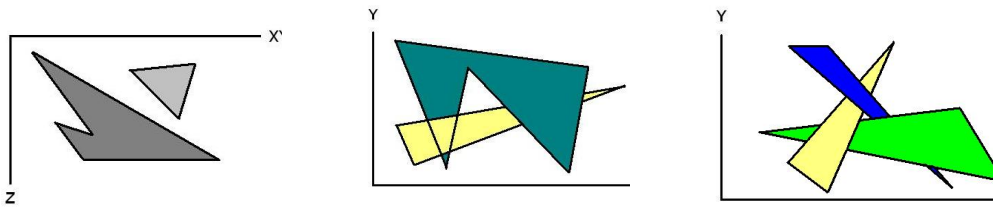

Figure 7: Alguns casos onde o valor de z se estende nos polígonos sobrepostos.

Os algoritmos baseados em lista de prioridades são algoritmos híbridos que combinam operações de precisão de objetos e precisão de imagem. A comparação da profundidade z é feita sobre os objetos, entretanto, a varredura de conversão realizada pelo dispositivo gráfico para sobrescrever os pixels dos objetos anteriores é feita com precisão de imagem. Como o ordenamento é realizado com precisão de objeto, a renderização pode ser feita com qualquer resolução de imagem. Temos dois algoritmos de lista de prioridades: o Algoritmo de Ordenamento de Profundidade (Newell, Newell and Sancha 1972) e os Arvore de partição binária do espaço (Fuchs, Kedem and Naylor 1980). A principal diferencia nestes algoritmos está na forma determinar a lista de prioridade, como são divididos os objetos e quando acontece tal divisão.

Em seguida, será apresentado brevemente o Algoritmo do Pintor, o qual é a versão mais simples do Algoritmo de Ordenamento de Profundidade.

Algoritmo do Pintor.

O Algoritmo do Pintor é uma versão simplificada do Algoritmo de Ordenamento de Profundidade, desenvolvido por Newell (Newell, Newell and Sancha 1972).

A ideia fundamental neste algoritmo é pintar os polígonos no frame buffer em ordem decrescente, o seja, do objeto mais afastado terminando no objeto mais próximo.

Dois passos conceituais são realizados:

1. Ordenar o polígono segundo seu valor da profundidade z.
2. Fazer a conversão para o frame buffer em ordem descendente (de atrás para frente).

Entretanto, realizando estes passos não se garante que o uma cena seja a desejada, pois nem sempre o algoritmo produz um ordenamento correto.

A figura 8 apresenta uma sequencia de objetos renderizados segundo o Algoritmo do Pintor.

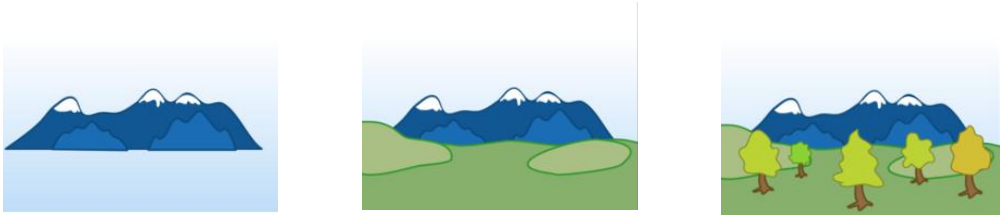

Figure 8: Algoritmo do Pintor. a) Primeiro se pinta as montanhas distantes. b) Depois, pinta-se o prado. c) Finalmente, pintam-se as árvores, as quais são os objetos mais próximos.

Algoritmos de subdivisão de área

Os algoritmos de subdivisão de área utilizam a estratégia “divide e vencerás” realizando uma divisão espacial do plano de projeção. Como a região da imagem é examinada, se fosse possível conhecer quais polígonos são visíveis na área, então eles

simplesmente seriam visualizados. Caso contrário, a área é novamente dividida recursivamente em pequenas subáreas, sobre as quais continuamos aplicando a decisão de visualização. Esta abordagem explora a coerência de área, o seja, pequenas áreas da projeção tendem a ser cobertas por apenas um polígono visível.

Dentro da bibliografia, se mencionam três algoritmos representativos desta categoria: o Algoritmo de Warnock (Warnock 1969), o Algoritmo de Weiler-Athernton (Weiler and Atherton 1977) e o Algoritmo de subdivisão de sub-pixels (Catmull, A hidden-surface algorithm with anti-aliasing 1978) (Carpenter 1984). A seguir, fazemos uma exposição do Algoritmo de Warnock.

Algoritmo de Warnock

O Algoritmo desenvolvido por Warnock (Warnock 1969) subdivide cada área de projeção em quatro quadrados iguais. Em cada etapa do processo de subdivisão recursiva, a projeção de cada polígono tem um de quatro tipos de relacionamento como a área de interesse.

1. ***Polígono envolvente***, aquele que contém completamente a área de interesse, figura 9-a.
2. ***Polígono de intersecção***, quando cruza a área, Fig. figura 9-b.
3. ***Polígono contido***, esta completamente dentro da área, figura 9-c.
4. ***Polígono disjunto***, esta complemente fora da área, Fig. figura 9-d.

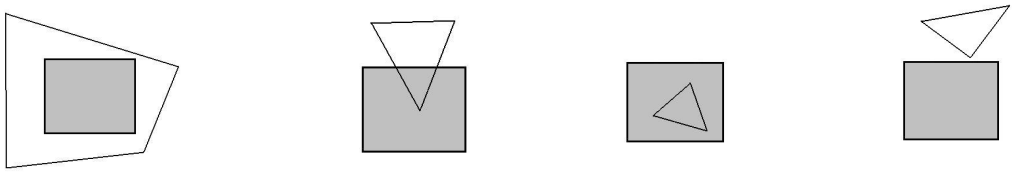

Figure 9: Quatro tipos de relacionamento da projeção de um polígono com uma área de interesse. a) Envolvente. b) Intersecção. c) Contido. d) Disjunto.

Da figura anterior, deduzimos que os ***polígonos disjuntos*** não tem nenhuma influencia sobre a área de interesse. Igualmente, a parte de um polígono de interseção que esta fora da área é irrelevante. Para o algoritmo somente são importantes à parte interior do ***polígono de intersecção*** que recaiu sobre a área de interesse, os ***polígonos contidos e os polígonos envolventes***.

Existem quatro casos triviais para determinar o que esta na frente de uma área, assim que a área não precisa dividir ainda mais para ser conquistada:

1. Todos os polígonos são disjuntos. A cor de fundo pode ser exibida na área.
2. Há somente um polígono contido ou um polígono de interseção. A área é preenchida primeiramente pela cor de fundo, logo depois, a parte de polígono contida na área é exibida.
3. Há um polígono envolvente, mas nenhum polígono de interseção e nenhum polígono contido. A área é preenchida pela cor do polígono envolvente.

4. Mais de um polígono é intersecante, contido, ou envolvente na área, mas um deles é um polígono envolvente mais próximo ao observador, então toda a área é preenchida com a cor do polígono envolvente.

O pseudocódigo do Algoritmo de Warnock apresenta-se na figura 10.

```
void Warnock(int viewport[2]) {
  int *area_to_be_subdivided = viewport;
  int number_of_areas = 1;
  //form the list of visible polygons
  sort_all_polygons_in_ascending_order();
  while (number_of_areas > 0) {
        select_an_area_as area_to_be_subdivided;
        if (all polygon forming an object projected on the
                    screen fall outside the area_to_be_subdivided)  {
              set   it with the background color;
        }
        if (only one polygon in the list of visible poligons cover
                    the area_to_be_subdivided) {
                    draw the polygon area inside the area_to_be_subdivided;
        }

        if (area_to_be_subdivided is a pixel) {
                    set the pixel with the color of polygon
                    with the minimum z_value;
        }

        if (none of the above is true) {
                    divide area into four subareas;
                    modify number of subareas;
        }
  }
}
```

Figure 10: Pseudocódigo do algoritmo de Warnock.

Para uma resolução de 1024x1024, são necessários, pelo menos, 10 níveis de subdivisão. Se ao alcançar o número máximo de subdivisões nenhumas dos quatro casos acontecer, então se calcula a profundidade dos polígonos no centro de pixel indivisível, pintando aquele pixel com a cor de polígono mais próximo.

A figura 11 apresenta uma cena simples com as subdivisões necessárias para exibição. O número em cada área corresponde aos casos acima indicados. Nas áreas sem nenhum rótulo, nenhuns dos enunciados são verdadeiros.

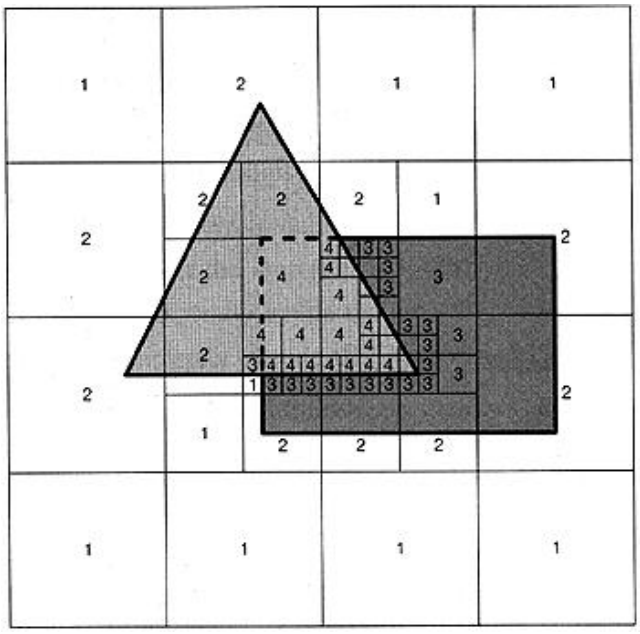

Figure 11: Algoritmo de Warnock. Subdivisão de área em quadrados iguais.

O principal inconveniente do Algoritmo de Warnock é que seus testes são muito complexos e lentos. As cenas complexas têm muitos polígonos pequenos e complicada distribuição de profundidade (z), assim varias regiões da tela terminam sobre um simples pixel.

Ray-Tracing

De acordo com a classificação feita no início do trabalho, o algoritmo de ray-tracing faz parte do grupo que aborda a visibilidade orientada a imagem, pois este método determina as superficies visíveis traçando raios de luz imaginários do observador até os objetos da cena, testando se há intersecção entre eles.

A idéia básica deste algoritmo consiste em traçar, para cada pixel na janela de visualização, um raio a partir do centro de projecão até o centro do pixel da cena. Assim, a cor do pixel sera definida como a cor do ponto de intersecção mais próximo encontrado, como pode ser visto na figura 12.

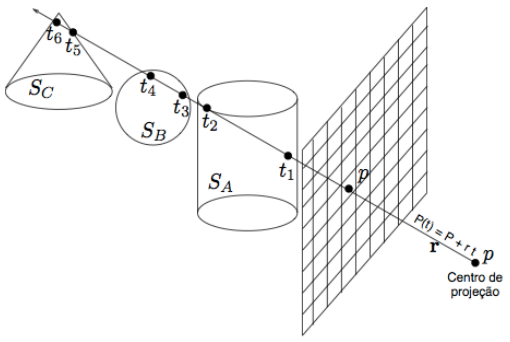

Figure 12: Raio projetado saindo do centro de projeção em direção aos objetos da cena.

Assim, este algoritmo, além da determinação de superficies visíveis, pode englobar em sua implementação, a tonalização de cada pixel visível. Entretanto, neste trabalho, será abordado apenas o primeiro caso, o qual se resume em encontrar a intersecção entre os raios traçados e os objetos da cena.

Para solucionar este problema, utiliza-se a representação paramétrica de um vetor. Assim, cada ponto $P(x, y, z)$ que pertença a um raio que parte do ponto $P0(x0, y0, z0)$ até o ponto $P1(x1, y1, z1)$ pode ser definido por uma variável *t*, de tal forma que:

$$x = x0 + t(x1 - x0)\ ou\ x = x0 + t\Delta x$$

$$y = y0 + t(y1 - y0)\ ou\ x = x0 + t\Delta y$$

$$z = z0 + t(z1 - z0)\ ou\ x = x0 + t\Delta z$$

Sendo:

t = 0 o ponto *P0* (COP – Centro de projeção)

t = 1 o ponto *P1* (pixel)

t < 0 a representação dos pontos atrás do COP

t > 1 a representação dos pontos mais distantes do centro de projeção

Assim, caso seja detectada a ocorrência de uma intersecção entre o raio e um objeto, calcula-se a distância entre eles e verifica-se se esta e a menor distância em relação a outros objetos que o mesmo raio ultrapassa.

Experimentos realizados

Visibilidade em OpenGL

Foram desenvolvidos alguns códigos, utilizando o OpenGL, para testar a renderização de cenas com a ativação e desativação do cáculo de profundidade e do back-face culling.

- Projeção em perspectiva, sem a ativação da função **gl_depth_test**

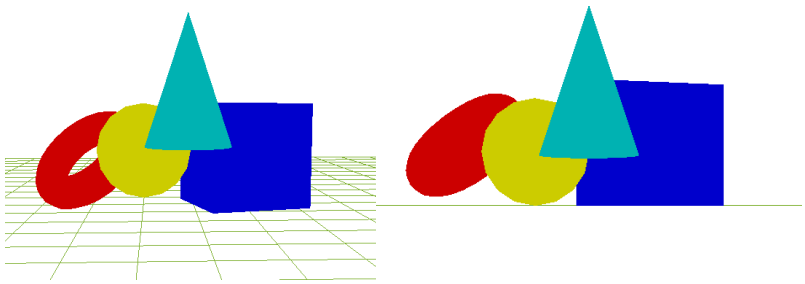

Figure 13: Imagens geradas sem o buffer de profundidade.

- Projeção em perspectiva, com a ativação da função **gl_depth_test**

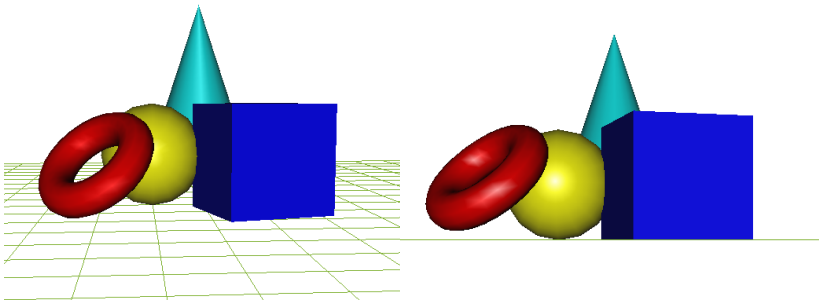

Figure 14: Imagens geradas com o buffer de profundidade ativo.

- Rasterização de polígonos com linhas.

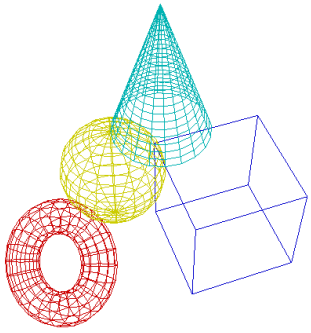

Figure 15: Cena gerada sem a ativação do Back-face Culling.

- Rasterização de polígonos com linhas e eliminação de linhas escondidas utilizando a função glEnable(**GL_CULL_FACE**) e **glCullFace(GL_FRONT)**, a qual não renderiza as faces frontais dos objetos.

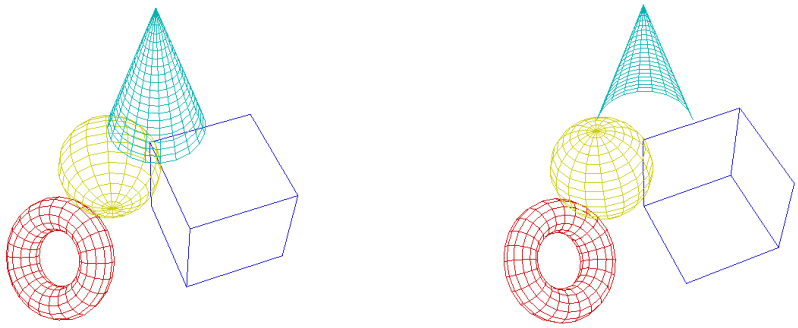

Figure 16: Cenas geradas utilizando Back-face Culling.

Visbilidade em PovRay

O PovRay é um programa de ray-trancing aplamente utilizado para geração de cenas virtuais.

Com este programa, gerou-se uma imagem com dois objetos e uma fonte de iluminação, como pode ser visto na figura 17.

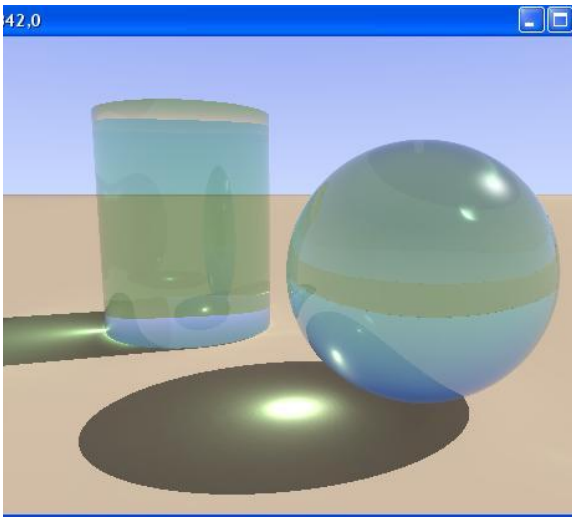

Figure 17: Cena 3D gerada utilizando o PovRay.

Durante a renderização da cena, pode-se perceber que os cáculos de ray-tracing eram realizados pixel a pixel na janela de visualização, reforçando a idéia de precisão de imagem deste algoritmo. Na figura 18 é mostrada a renderização parcial da cena.

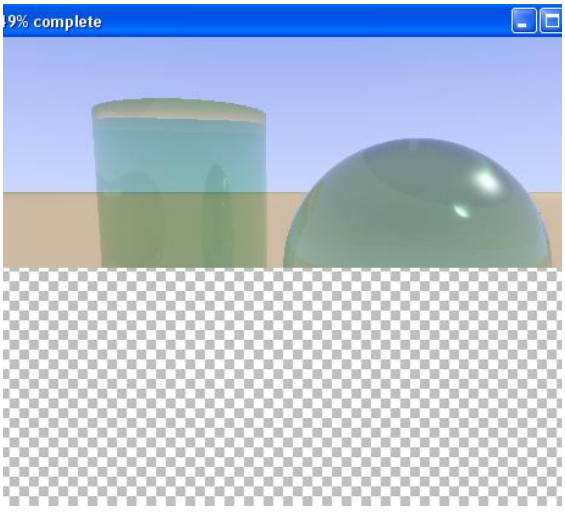

Figure 18: Renderização parcial de uma cena 3D no PovRay.

Conclusão

Neste trabalho, conseguiu-se fazer um estudo das principais técnicas de visibilidades existentes, como o algoritmo do Pintor, Warnock, Z-Buffer e Ray-Tracing, sendo estes últimos, juntamente com suas combinações com outros algoritmos como Scanline e BSP, os mais utilizados atualmente.

Percebeu-se também, durante o trabalho, que além de resolver o problema de determinação de linhas e superficies visíveis para um observador, estas técnicas podem ser utilizadas para determinar quais superficies são visíveis para uma fonte de luz, determinando, assim, quais destas estão localizadas em locais de sombra.

Além do estudo das técnicas, conseguiu-se validar estes algoritmos utilizando OpenGL, através de suas funções como **gl_depth_test**, a qual ativa o teste de profundidade; e a função **glEnable(GL_CULL_FACE)**, a qual ativa o Back-face Culling. Uma outra função utilizada do OpenGL foi .

A validação do algoritmode Ray-tracing foi realizada utilizando-se o programa PovRay, com o qual percebeu-se a geração da imagem pixel a pixel.

Bibliografia

Warnock, John Edward. *A hidden surface algorithm for computer generated halftone pictures.* The University of Utah, 1969.

Watt, A.H. *3D computer graphics.* Vol. 1. Addison-Wesley, 2000.

Weiler, Kevin, and Peter Atherton. "Hidden surface removal using polygon area sorting." *SIGGRAPH Comput. Graph.* (ACM) 11, no. 2 (july 1977): 214-222.

Carpenter, Loren. "The A -buffer, an antialiased hidden surface method." *SIGGRAPH Comput. Graph.* (ACM) 18, no. 3 (january 1984): 103-108.

Catmull, Edwin Earl. "A hidden-surface algorithm with anti-aliasing." *SIGGRAPH Comput. Graph.* (ACM) 12, no. 3 (august 1978): 6-11.

—. *A subdivision algorithm for computer display of curved surfaces.* The University of Utah, 1974.

Fuchs, Henry, Zvi M. Kedem, and Bruce F. Naylor. "On visible surface generation by a priori tree structures." *SIGGRAPH Comput. Graph.* (ACM) 14, no. 3 (july 1980): 124-133.

Foley, J. D. *Computer graphics: principles and practice.* Addison-Wesley, 1996.

Newell, M. E., R. G. Newell, and T. L. Sancha. "A solution to the hidden surface problem." *Proceedings of the ACM annual conference - Volume 1.* Boston, Massachusetts, United States: ACM, 1972. 443-450.

Ting, Wu Shin. "Sistemas de informações gráficas: Síntese de Imagens: Uma Introdução ao Mundo de Desenho e Pintura dos Sistemas Digitais." Campinas: Universidade Estadual de Campinas, 2009.