PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

DEPARTAMENTO DE MATEMÁTICA

Visualização e Reconstrução para Nuvem de Pontos

Alexandre Marangoni Costa[1]
Sinésio Pesco[2]

CNPq
Conselho Nacional de Desenvolvimento
Científico e Tecnológico

[1]Aluno de Graduação do Departamento de Engenharia da Computação da PUC-Rio.
[2]Doutor em Matemática, Professor Assistente do Departamento de Matemática da PUC-Rio.

SUMÁRIO

1. INTRODUÇÃO

Nuvens de pontos (point-clouds) surgem principalmente como dados de saída de scanners 3D. Uma estratégia para visualizar nuvens de pontos é determinar a visibilidade dos pontos sem aplicar técnicas de reconstrução.

A visualização direta de uma nuvem de pontos resulta em ambiguidades, visto que um ponto p1, em geral, não oculta outro ponto p2 situado atrás dele (a menos que p1, p2 e o observador sejam colineares). A figura 1 ilustra a dificuldade de se determinar o conjunto de pontos visíveis a um determinado observador.

Katz et al.[1] propõe uma técnica para determinar a visibilidade com a aplicação de um fecho convexo e uma transformação de reflexão. Assim é possível determinar quais pontos estão mais próximos do observador, e se possa constatar que, neste caso, o observador esta visualizando a parte de trás do objeto mencionado (ver figura 2). Esse método será descrito em detalhes nas seções seguintes.

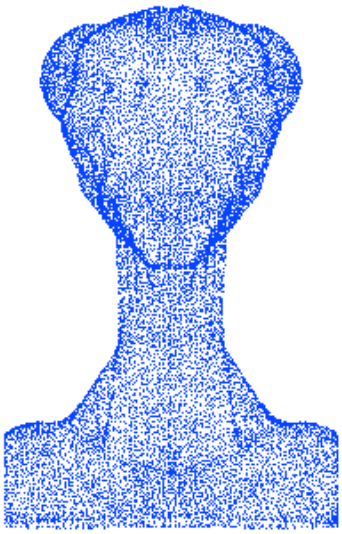


fig. 1: alien.ply.

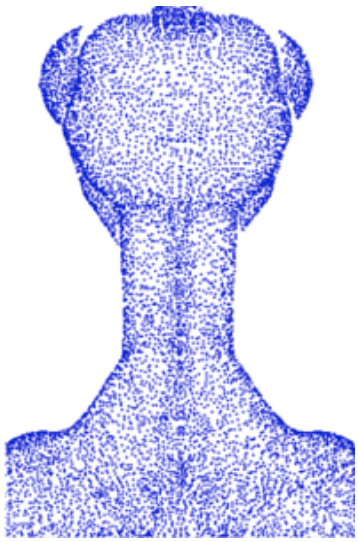


fig. 2: alien.ply – Somente os pontos das costas são visíveis para o observador definido.

2. OBJETIVO

O presente trabalho tem como objetivo o desenvolvimento de um método computacional, através da utilização da linguagem de programação C/C++ e da biblioteca gráfica OpenGL, capaz de visualizar uma nuvem de pontos e de, posteriormente, otimizar esta visualização utilizando a proposta de Katz et al. [1].

A implementação utilizada neste trabalho mostrou-se eficiente no aspecto de determinar a visibilidade para vários tipos de point-clouds. Essa eficiência depende, porém, da manipulação de parâmetros que devem variar com o espaçamento entre os pontos utilizados, bem como com o tamanho virtual do objeto (maior distância calculada entre dois pontos).

3. TRABALHOS ANTERIORES

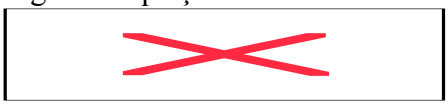
A visualização de nuvens de pontos remete a um estudo direcionado à “point-set surfaces” [7]. Fugindo um pouco desta abordagem, este trabalho visa basicamente determinar quais pontos são visíveis para um observador fixo.

Um tipo de algoritmo bastante utilizado em geometria computacional é o algoritmo de fecho convexo (Convex-Hull). Ele é responsável por determinar o menor polígono (ou poliedro) convexo que contenha um conjunto de pontos dados. Isto é muito útil quando se deseja selecionar determinados pontos em um conjunto C1 e formar, a partir destes, um outro conjunto menor C2, ainda mais útil ao estudo pretendido. Neste trabalho em particular, foi utilizado o algoritmo de Convex Hull elaborado por Joseph O'Rourke e disponível em seu livro "Computational Geometry in C" (Segunda edição), capítulo 4 [2].

Um outro problema significativo relacionado à computação gráfica é a reconstrução de superfícies por meio de um conjunto de pontos [8], [9], [10], [7], [11], [12] e [13]. Apesar deste trabalho não ter como foco principal este tema, ele irá apresentar indiretamente uma alternativa que irá facilitar tal reconstrução.

4. CONCEITOS BÁSICOS

Reflexão sob a superfície de uma esfera. Considere uma esfera de raio R, centrada em C, que contenha todos os pontos de um conjunto P. Pode-se refletir um ponto $p_i \in P$ sob a superfície desta esfera a partir da seguinte equação:



O objetivo da reflexão é suavizar as concavidades do objeto quando visto por um observador localizado no centro da esfera. As figuras 3 e 4 ilustram o resultado da reflexão:

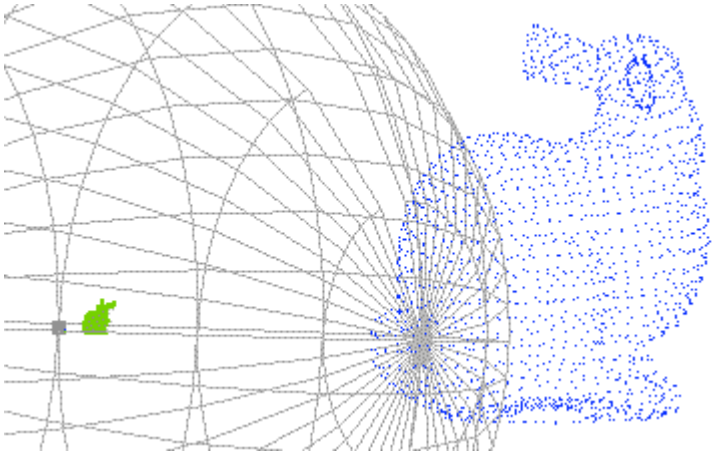


fig. 3: bunny.ply - Em verde temos o conjunto original de pontos. Em azul temos este conjunto refletido sob a superfície da esfera cinza, com centro em cinza.

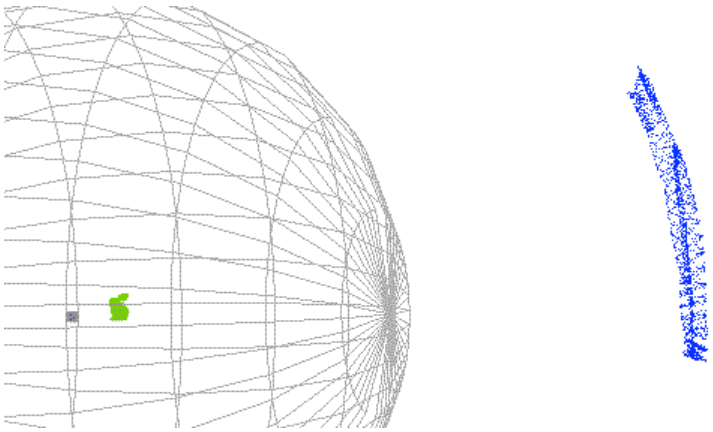


fig. 4: bunny.ply – Reflexão sob a esfera.

A partir destas imagens, pode-se observar que o resultado da reflexão é um conjunto de pontos mais delgado. Além disso, as concavidades existentes na sua superfície são claramente suavizadas.

Em C, podemos utilizar um laço para, a partir de uma matriz M com as coordenadas dos pontos do coelho em verde, obter uma matriz G dos pontos do coelho refletido em azul. A variável R é o raio da esfera.

```
for(i=0;i<n;i++){
        norma= sqrt(M[i][0]*M[i][0]+M[i][1]*M[i][1]+M[i][2]*M[i][2]);
        G[i][0] = M[i][0] + 2*(R-norma)*M[i][0]/norma;
        G[i][1] = M[i][1] + 2*(R-norma)*M[i][1]/norma;
        G[i][2] = M[i][2] + 2*(R-norma)*M[i][2]/norma;
}
```

Utilizando OpenGL, podemos plotar a matriz M e a matriz G do seguinte modo:

```
glColor3f(0.4, 0.8, 0.0);
for(i=0; i<n; i++){
        glBegin(GL_POINTS);
glVertex3f(M[i][0], M[i][1], M[i][2]);
        glEnd();
}
glColor3f(0.0, 0.0, 1.0);
for(i=0; i<n; i++){
        glBegin(GL_POINTS);
              glVertex3f(G[i][0],G[i][1],G[i][2]);
        glEnd();
}
```

Fecho convexo. Outro conceito importante para o entendimento do método mostrado neste trabalho é o conceito de fecho convexo. Dado um conjunto de pontos P, um fecho convexo é uma combinação finita de elementos desse conjunto de modo a formar uma figura convexa. Pode-se generalizar este conceito imaginando, no caso de pontos coplanares, um elástico esticado envolvendo tais pontos, ou, no caso de não-coplanares, um balão sob tensão (ver figura 5).

Neste trabalho, o algoritmo para a aplicação do fecho convexo foi extraído do livro "Computational Geometry in C" (segunda edição) de Joseph O'Rourke [2], com algumas modificações para melhor se adequar ao nosso propósito.

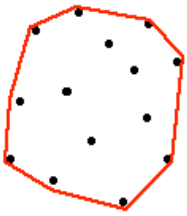


fig. 5

5. DESCRIÇÃO DO MÉTODO

Este método foi desenvolvido primariamente por Katz et al. [1], e, posteriormente, foi implementado em OpenGL usando C, e explorado neste trabalho. O algoritmo utiliza uma combinação dos conceitos básicos mostrados anteriormente e será descrito em três etapas.

Inicialmente temos a situação mostrada na fig.6-a. Aplicando uma reflexão sob uma esfera de centro no observador, obtêm-se como resultado a fig.6-b. Observa-se nesta figura a suavização da região mais próxima ao observador, que reduz as concavidades.

Para selecionar os pontos desejados, calcula-se o fecho convexo sobre o conjunto de pontos da nuvem refletida incluindo o ponto que representa a posição do observador. Verifica-se nesta etapa a importância da etapa anterior, pois se a reflexão não fosse feita, a superfície desejada seria côncava, o que impediria o fecho convexo de selecionar todos os seus pontos. O resultado é mostrado na fig.6-c, onde os pontos selecionados pelo fecho estão em verde.

Como resultado, os pontos que pertencem ao fecho são os pontos visíveis ao observador, antes da reflexão. Para localizar esses pontos, basta observar os índices dos elementos capturados pelo fecho na matriz de pontos inicial. Esta informação é usada para localizar e plotar os pontos de mesmo índice, porém antes da reflexão. Assim não se faz necessária a inversão da reflexão para a obtenção dos pontos desejados. Pode-se observar o resultado na figura fig.6-d, onde os pontos obtidos estão em verde. Todo o resto do objeto que não está em verde pode agora ser desprezado, visto que estes pontos não são visíveis ao observador O.

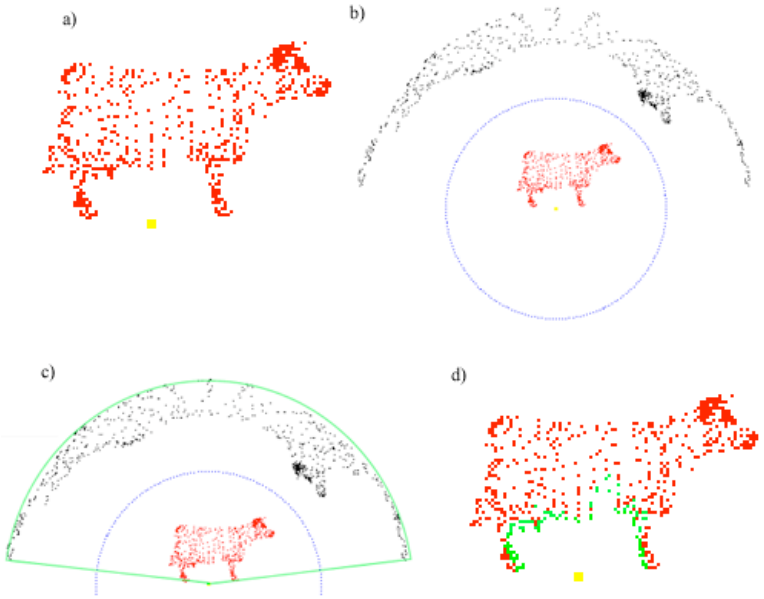


fig. 6

6. DESCRIÇÃO DO CÓDIGO

Para uma visualização prática de toda a teoria abordada neste estudo foram usadas as linguagens de programação C e C++ juntas da biblioteca para modelagem gráfica OpenGL.

Uma nuvem de pontos, geralmente, é guardada num arquivo de texto. Este arquivo, inicialmente, guarda algumas informações a respeito da point-set, como a informação de quantos pontos compõem certa nuvem de pontos, e, posteriormente, fornece as coordenadas x, y e z de cada ponto.

Visto isso, o código inicia-se com a leitura do arquivo de texto, utilizando os comandos fopen e fscanf, guardando, primeiramente, o número total de pontos da point-cloud. Esta informação é utilizada para fazer a alocação dinâmica de uma matriz M[n][3], que irá guardar as coordenadas dos pontos da nuvem, onde n é o número de pontos e 3 são as coordenadas x, y e z. De posse desta matriz, os dados são colhidos e armazenados nela.

A partir daí, os métodos anteriormente explicados são aplicados da seguinte forma: a matriz criada é utilizada para gerar outra matriz G[n][3], que armazenará os pontos da matriz M refletidos sob a superfície de uma esfera e, em seguida, a matriz G e a posição do observador são submetidas a aplicação de um fecho convexo.

```
for(i=0;i<n;i++)
{
        norma = sqrt(M[i][0]*M[i][0]+M[i][1]*M[i][1]+M[i][2]*M[i][2]);
        G[i][0] = M[i][0] + 2*(R-norma)*M[i][0]/norma;
        G[i][1] = M[i][1] + 2*(R-norma)*M[i][1]/norma;
        G[i][2] = M[i][2] + 2*(R-norma)*M[i][2]/norma;
}
convex_hull();
recupera_indices();
reconstroi_objeto();
```

Na fórmula utilizada para obter a reflexão esférica deve-se tomar bastante cuidado com o tamanho do raio R definido. Caso este seja muito pequeno, as concavidades presentes na superfície não serão totalmente suavizadas, e o fecho convexo não irá capturar todos os pontos que desejamos. Caso o raio R seja definido exageradamente grande, o fecho convexo irá capturar pontos da face oposta, que não gostaríamos de plotar.

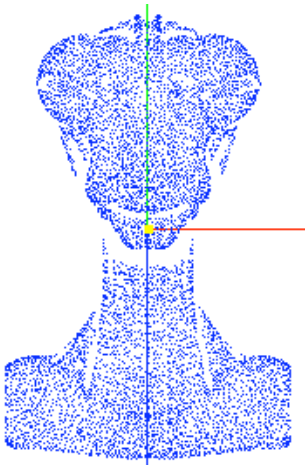


fig. 7: Raio pequeno – faltam pontos

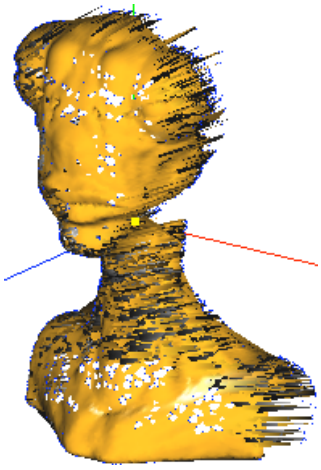


fig. 8: Raio grande – sobram pontos

O algoritmo de fecho convexo dará ao programa, como saída, o índice dos vértices dos pontos capturados pelo fecho. De posse destes índices, utiliza-se a biblioteca gráfica OpenGL para plotar o objeto desejado na tela. Porém, após todo este método, não é mais necessário plotar o objeto como um todo, e sim somente os pontos vistos pelo observador, que foram obtidos a partir do fecho convexo.

7. RESULTADOS

A implementação utilizada neste trabalho mostra-se eficiente para vários tipos de point-clouds. Essa eficiência depende, porém, da manipulação de parâmetros que devem variar com o espaçamento entre os pontos utilizados, bem como com o tamanho virtual do objeto (maior distância calculada entre dois pontos deste). Ainda assim, os resultados são bastante satisfatórios (como pode ser conferido nas figuras abaixo), deixando a visualização da nuvem de pontos bem mais nítida e com menor esforço computacional.

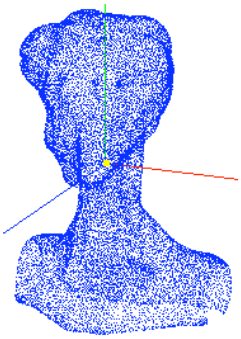


fig. 9: Original

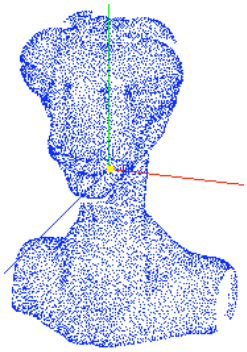


fig. 10: Após o método

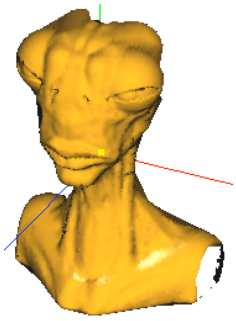


fig. 11: Aplicação de iluminação

Neste trabalho, o exemplo utilizado para descrição do método foi o de um alienígena. Abaixo seguem dois outros exemplos testados: bunny e skull.

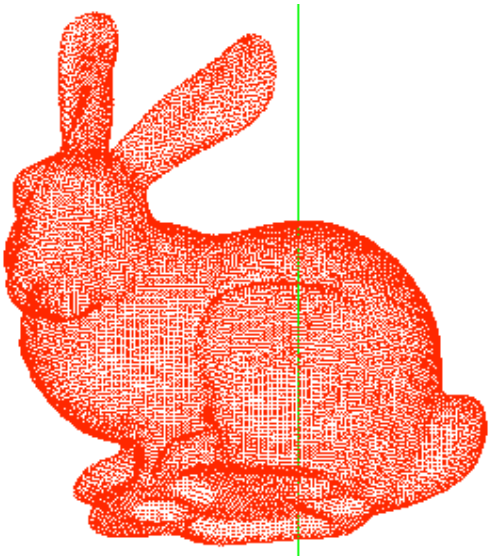


fig. 12: bunny.ply - Original

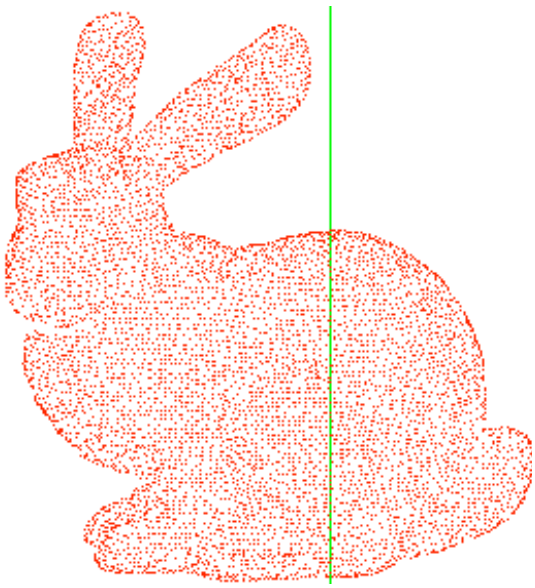


fig. 13: bunny.ply – Após aplicação do método

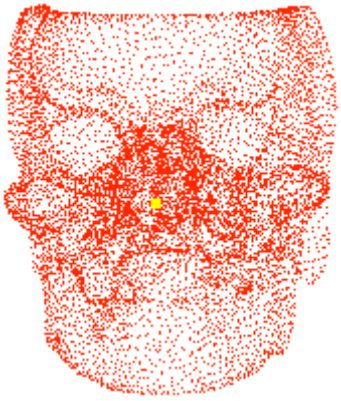


Fig. 14: skull.ply - Original

Fig. 15: skull.ply – Após aplicação do método

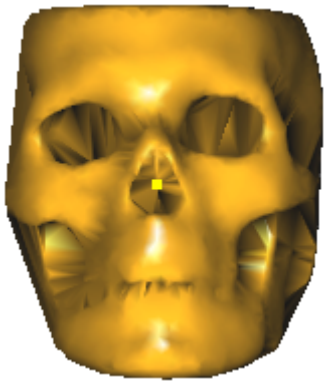


Fig. 16: skull.ply – Usando Iluminação e reconstrução das faces

8. REFERÊNCIAS

[1] Katz S., Tal A., and Basri R., Direct Visibility of Point Sets, *SIGGRAPH 2007, ACM Transactions on Graphics*, Volume 26, Issue 3, August 2007

[2] J. O'Rourke, Computational Geometry in C, Cambridge University Press, 1994. Segunda edição, 1998.

[3] Jack E. Bresenham, "Algorithm for computer control of a digital plotter", *IBM Systems Journal*, Vol. 4, No.1, January 1965, pp. 25–30

[4] APPEL, A. 1968. Some techniques for shading machine renderings of solids. In *AFIPS Spring Joint Computer Conf.*, vol. 32, 37.45.

[5] SUTHERLAND, E., SPROULL, R., AND SCHUMACKER, R. 1974. A characterization of ten hidden-surface algorithms. *ACM Comput. Surv. 6*, 1, 1.55.

[6] SAINZ, M., AND PAJAROLA, R. 2004. Point-based rendering techniques. *Computers & Graphics 28*, 6, 869.879.

[7] ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. 2003. Computing and rendering point set surfaces. *IEEE Trans. on Vis. and Computer Graphics 9*, 1, 3.15.

[8] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. *Computer Graphics 26*, 2, 71.78.

[9] CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH*, 67.76.

[10] OHTAKE, Y., BELYAEV, A., ALEXA, M., TURK, G., AND SEIDEL, H.-P. 2003. Multi-level partition of unity implicits. *ACM Trans. Graph. 22*, 3, 463.470.

[11] FLEISHMAN, S., COHEN-OR, D., AND SILVA, C. 2005. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph. 24*, 3, 544.552.

[12] AMENTA, N., CHOI, S., AND KOLLURI, R. 2001. The power crust, unions of balls, and the medial axis transform. *Int. J. of Computational Geometry and its Applications 19*, 2-3, 127.153.

[13] MEDEROS, B., AMENTA, N., VEHLO, L., AND DE FIGUEIREDO, L. 2005. Surface reconstruction from noisy point clouds. In *Eurographics Symposium on Geometry Processing*, 53.62.