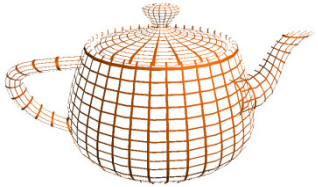
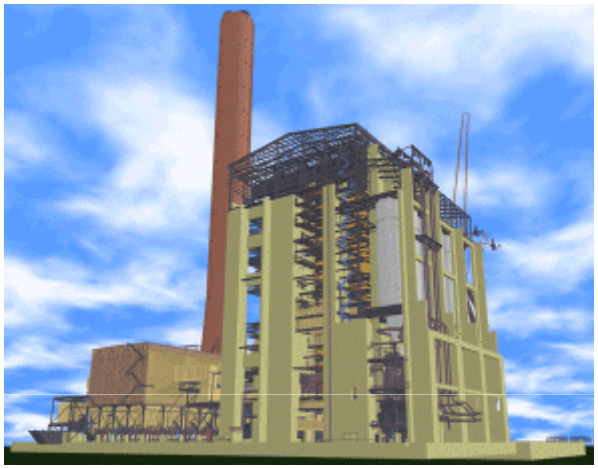
Computação Gráfica

Visualização e Navegação em Tempo Real

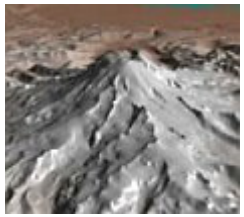


O Problema: Triângulos!

buda: 1 milhão

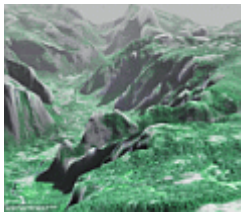


central: 13 milhões

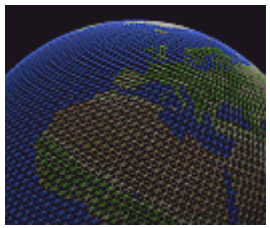


Terreno: 1.3 milhões

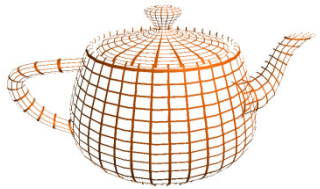
Terreno: 512 milhões



Terreno: 16 milhões



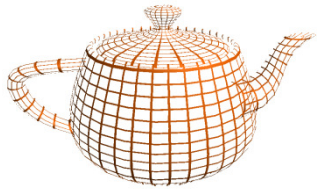
Terra: 1 bilião de pontos



O Problema

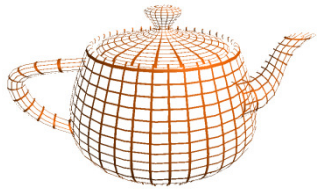
- Não existe hardware "actualmente" capaz de produzir interacção em tempo real com cenas de grande dimensão.

- Amanha teremos cenas ainda maiores, ou modelos de iluminação (real-time) mais complexos.



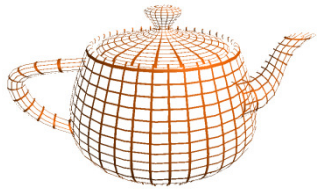
Questões

- Dada uma posição de câmara, quantos polígonos é que realmente contribuem para a imagem final?
- Para modelos distantes precisamos de tanto detalhe?
- Como interactuar com modelos de dimensão tão elevada?



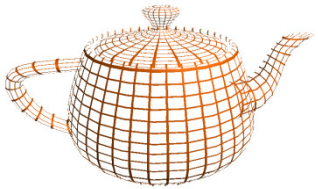
Soluções

- Eliminação das partes que não contribuem para a cena final (culling)
  - Back Face Culling
  - View Frustum Culling
    - Bounding Volumes
    - Partição Espacial – Quad e Octrees
    - *Partição Espacial – BSP*
    - *Partição Espacial – Portais*
  - *Occlusion Culling*



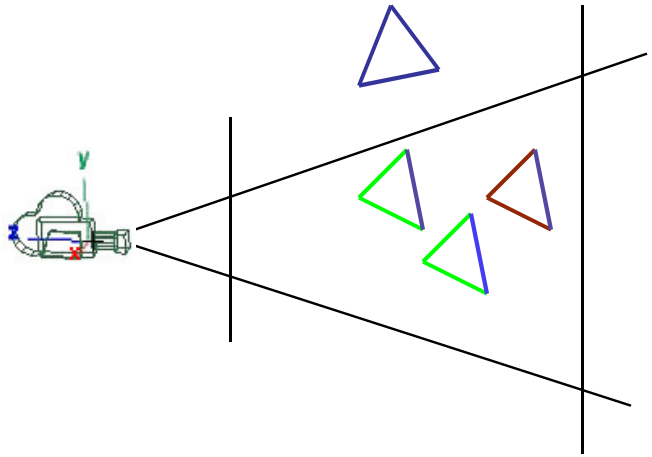
Soluções

- Eliminação de detalhes dificilmente perceptíveis:
  - *Level of Detail*
  - *Impostores*



Culling

- Eliminação de polígonos que não contribuem para a imagem final

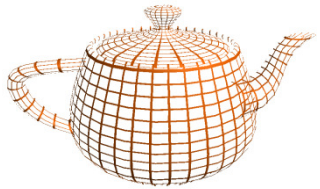


Polígonos Visíveis

Back Face Culling

View Frustum Culling

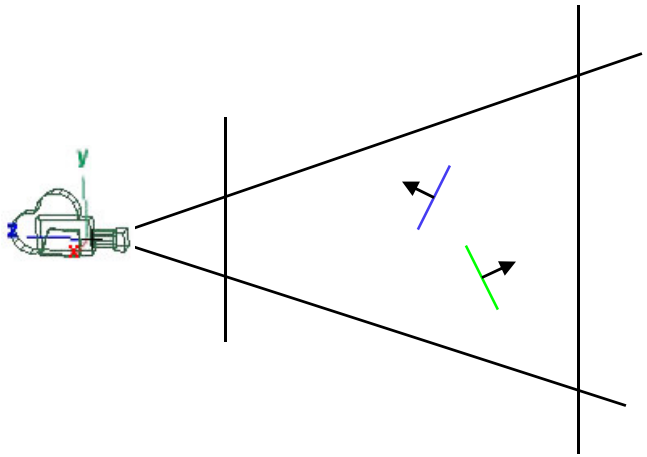
Occlusion Culling



Back Face Culling

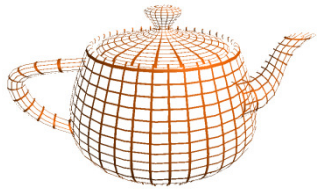
- Eliminação dos polígonos cuja face não está virada para a câmara.

$$o = v.n$$



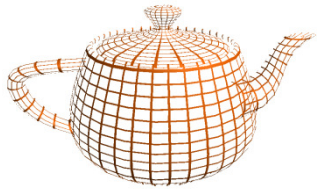

```
if(o > 0)
    render
else
    cull
```

v - vector do polígono à câmara
n – normal do polígono



Back Face Culling

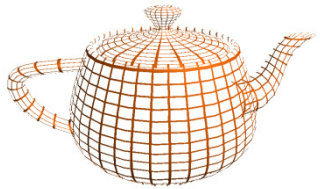
- Em OpenGL
  - Activar/Desactivar
    - `glEnable(GL_CULL_FACE);`
    - `glDisable(GL_CULL_FACE);`
  - Definir a face visível
    - `glCullFace(GL_BACK); // ou GL_FRONT`
  - Definir a orientação dos polígonos
    - `glFrontFace(GL_CCW); // ou GL_CW`



Back Face Culling

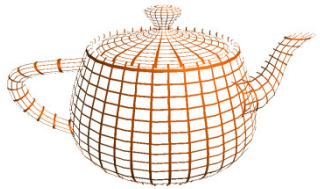
- Permite eliminar grande número de triângulos

- Realizado em hardware polígono a polígono



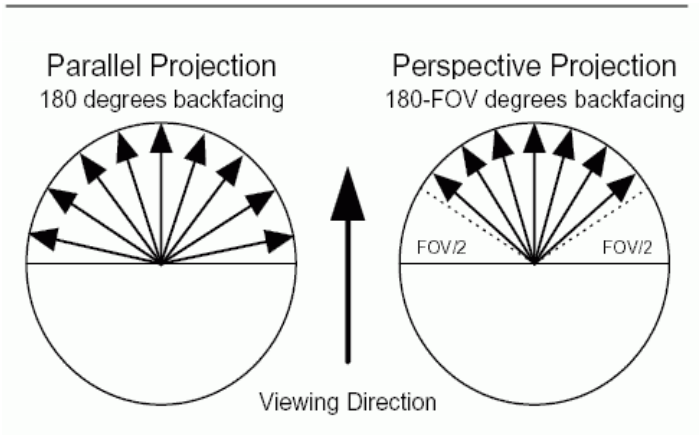
Back Face Culling

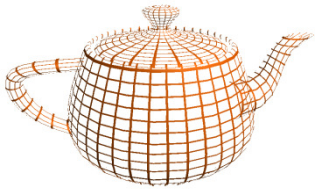
- O processo por hardware implica que os polígonos ainda têm de ser enviados para a placa gráfica.
- A eliminação só ocorre algures no pipeline gráfico
- O ideal seria evitar esta comunicação desnecessária...
- ...mas o teste individual em software seria demasiado demorado



Back Face Culling

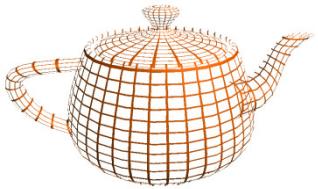
- Zhang and Hoff propõem:
  - Agrupar os polígonos de acordo com a direcção das suas normais
  - Eliminar grupos de polígonos de uma só vez





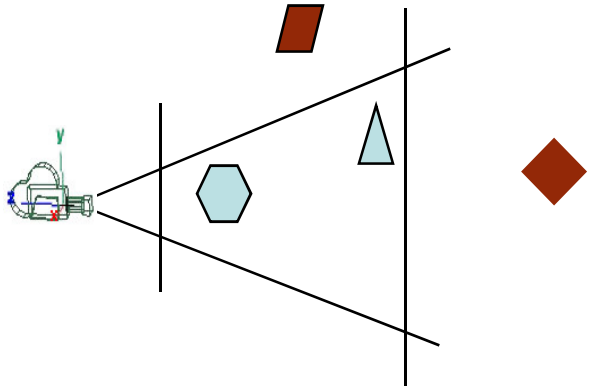
Culling

- Back Face Culling
- **View Frustum Culling**

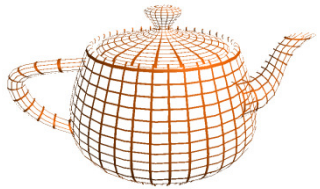


View Frustum Culling

- Eliminar Polígonos fora do volume de visualização

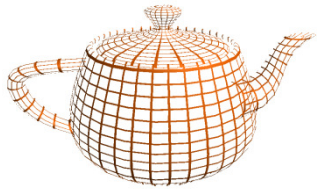


Método: testar a posição relativa aos planos do volume de visualização



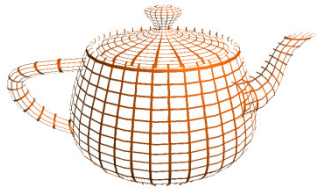
View Frustum Culling

- Fases:
  - Setup: Extracção dos planos do view frustum (1 vez por frame)
  - Teste: Para cada vértice determinar se este se encontra dentro ou fora do volume de visualização (1 vez por vértice)



View Frustum Culling

- Operação pode ser realizada em:
  - Clip Space
  - Espaço global (World Space)



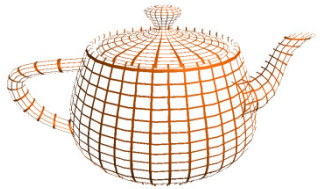
View Frustum Culling

- Clip Space: Setup
  - Seja M a matriz modelview , P a matriz de projecção, e p um ponto em World Space

$$A = M * P$$

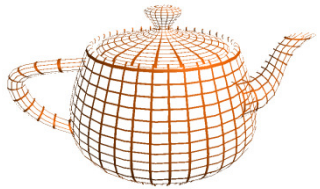
$$p' = pA$$

  - então p' é o ponto em clip space,
  - ou seja, A é a matriz que permite converter pontos de World Space para Clip Space



View Frustum Culling

- Clip Space - Setup
  - Obtenção das matrizes em OpenGL
    - `float m[16],p[16];`
    - `glGetFloatv(GL_MODELVIEW_MATRIX,m);`
    - `glGetFloatv(GL_PROJECTION_MATRIX,p);`

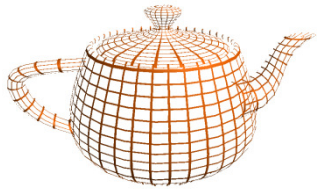


View Frustum Culling

- Multiplicação de Matrizes com OpenGL

- Código para fazer matA = m * p

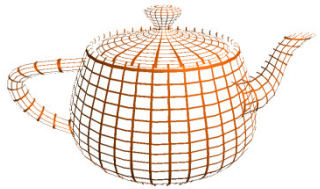
```
glPushMatrix();

glLoadMatrixf(p);
glMultMatrixf(m);
glGetFloatv(GL_MODELVIEW_MATRIX, matA);

glPopMatrix();
```



View Frustum Culling

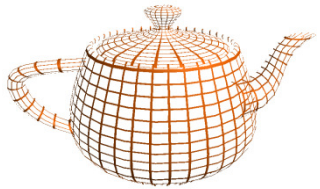
- Clip Space: Test
  - Os pontos visiveis em Clip Space estão no cubo centrado na origem com dimensão 2, ou seja, de -1 a 1 (coordenadas homógeneas) em todas as dimensões.
  - Seja p um ponto em World Space,
  - então $p'=(x',y',z',w') = pA$ é o ponto em Clip Space.
  -
  - p' está dentro do volume de visualização se:

$$\begin{aligned} -w' < x' < w' \\ -w' < y' < w' \\ -w' < z' < w' \end{aligned}$$



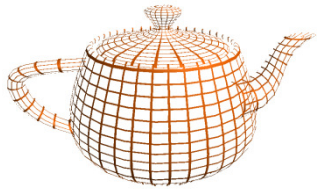
View Frustum Culling

- Clip Space: Test
  - Número de operações:
    - 16 multiplicações + 12 adições para obter o ponto em clip space
    - até 6 testes (<,>) para determinar se o ponto se encontra dentro ou fora do cubo a visualizar.



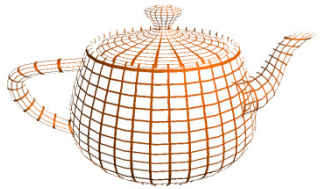
View Frustum Culling

- World Space: Setup
  - Seja $p=(x,y,z,w)$ e $p'=pA=(x',y',z',w')$.
  - Sabemos que
    - $-w' < x' < w'$



View Frustum Culling

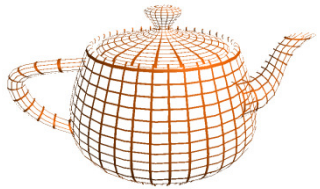
- World Space: Setup
  - Seja
    - $A = [col_1, col_2, col_3, col_4]$
  - então (em Clip Space) se
    - $-w' < x' < w'$
  - temos (em World Space)
    - $-p*col_4 < p*col_1 < p*col_4$



View Frustum Culling

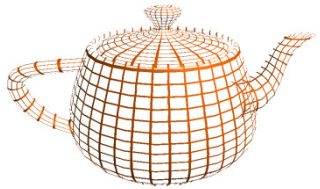
- World Space: Setup
  - $-p*col_4 < p*col_1 < p*col_4$
  - Se x está à direita do plano esquerdo então
    - $-p*col_4 < p*col_1$
    - $0 < p*col_1 + p*col_4$
    - $0 < p * (col_1 + col_4)$
    - $0 < x(a_{11}+a_{41}) + y(a_{12}+a_{42}) + z(a_{13}+a_{43}) + w(a_{14}+a_{44})$

Nota: os índices são (coluna,linha)



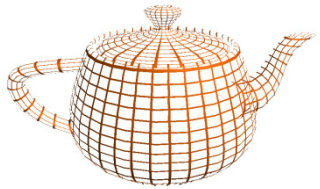
View Frustum Culling

- World Space: Setup
  - O plano esquerdo do view frustum é portanto
    - $x(a_{11}+a_{41}) + y(a_{12}+a_{42}) + z(a_{13}+a_{43}) + w(a_{14}+a_{44}) = 0$
  - Raciocínio idêntico permite retirar os restantes planos
  - Os planos do view frustum podem portanto ser extraídos directamente da matriz $A = MP$.
  - Setup é ligeiramente mais demorado que no caso do Clip Space.



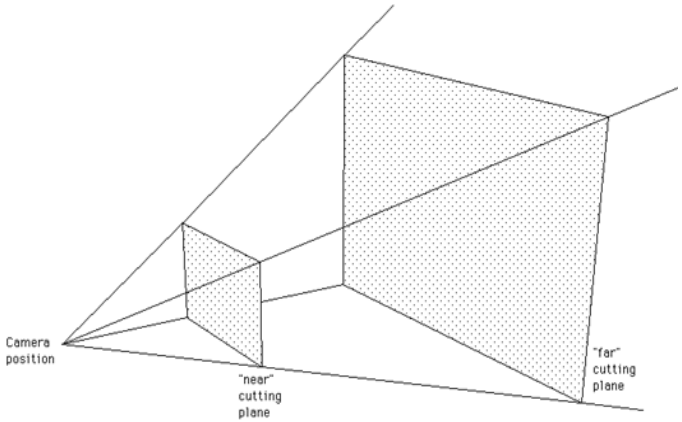
View Frustum Culling

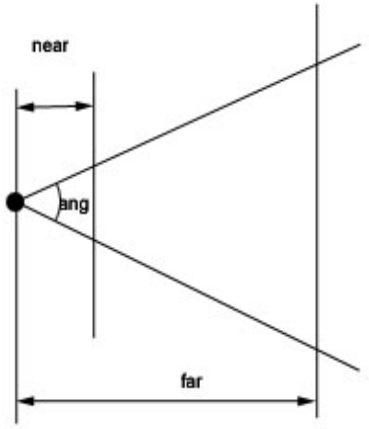
- World Space – Setup Alternativo:
  - Cálculo directo dos planos em World Space sem necessitar de obter as matrizes do OpenGL

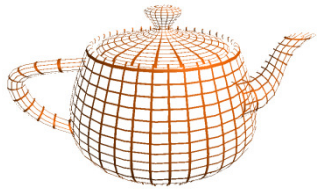


View Frustum Culling

- World Space – Setup Alternativo:
  - definição da área de visualização
    - `gluPerspective(ang,ratio,near,far);`
    - `gluLookAt(pos, lookAt, up);`







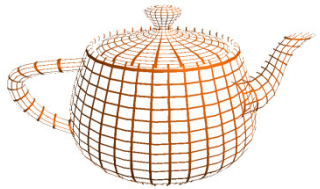
View Frustum Culling

- World Space – Setup alternativo
  - Cálculo dos 8 pontos do frustum em World Space
    - Para cada plano, *near* e *far*, calcular metade da sua altura, e largura

$$a = dist * \tan\left(\frac{ang}{2}\right)$$

$$l = dist * \tan\left(\frac{ang}{2}\right) * ratio$$

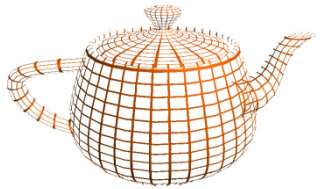
sendo *dist* = distância aos planos *near* ou *far*



View Frustum Culling

- World Space – Setup alternativo
  - Calcular os cantos de cada plano
    - Sendo a e l metade altura e largura do plano respectivamente temos:

```
se = cam.pos + cam.dir * dist + cam.up * a - cam.left * l
sd = cam.pos + cam.dir * dist + cam.up * a + cam.left * l
ie = cam.pos + cam.dir * dist - cam.up * a - cam.left * l
id = cam.pos + cam.dir * dist - cam.up * a + cam.left * l
```



View Frustum Culling

- World Space: Setup Alternativo
  - Três pontos definem um plano

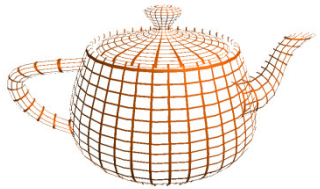
$$v_1 = (p_1 - p_0), v_2 = (p_2 - p_0)$$

$$n = \frac{v_1 \times v_2}{\| v_1 \times v_2 \|}$$

$$d = p_1 \bullet n$$

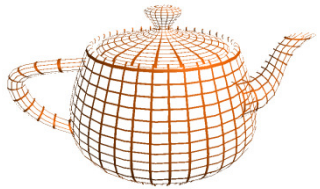
$$xn_1 + yn_2 + zn_3 + d = 0$$

  - Nota: atenção para obter consistência nas equações dos planos (sinais para dentro e fora)



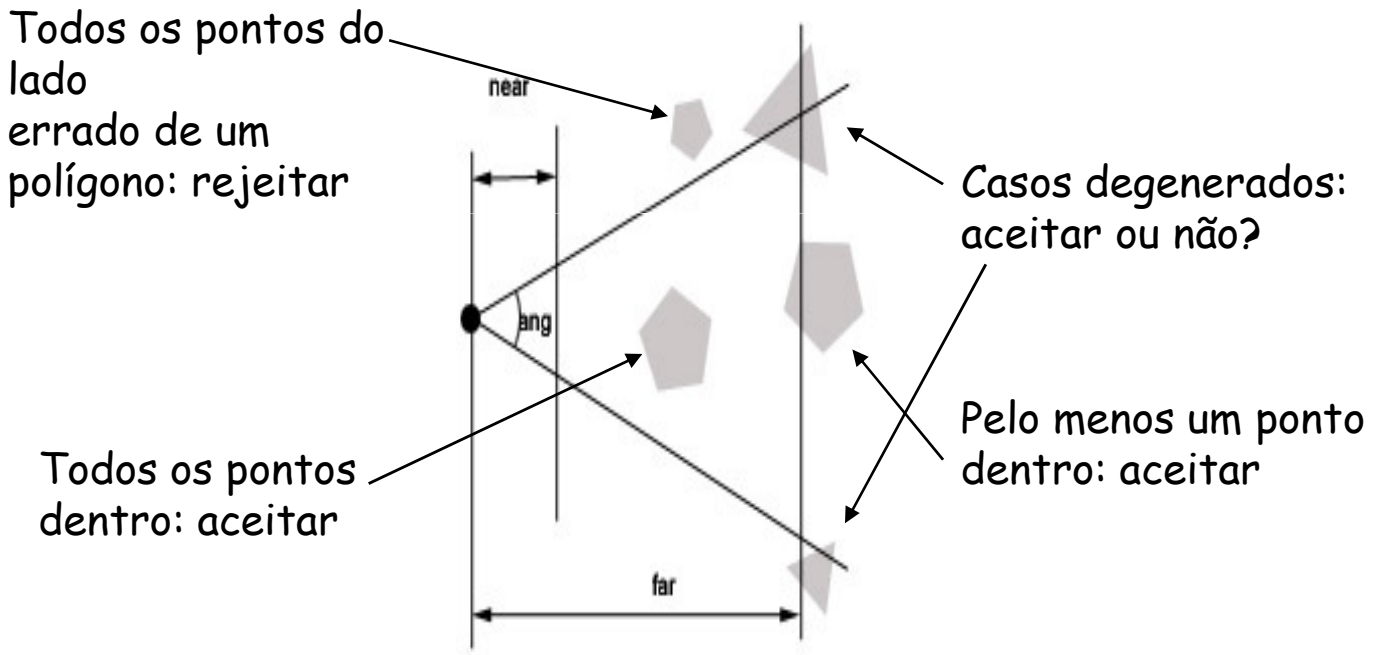
View Frustum Culling

- World Space: Test
  - Para verificar se um ponto está dentro do View Frustum é preciso testar o ponto contra os 6 planos.
  - Número de operações:
    - 1 plano => 3 adições e 3 multiplicações
    - 6 planos => 18 adições e 18 multiplicações
  - Para rejeitar um ponto basta que este se encontre do lado "errado" de um plano

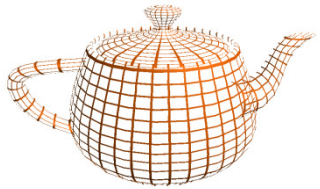


View Frustum Culling

- O teste a um polígono/volume tem de considerar alguns casos especiais:

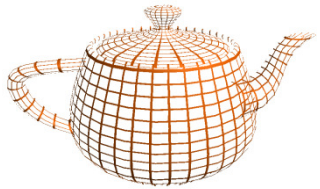


Rejeitar se todos os pontos do polígono/volume estiverem do lado "errado" de um plano



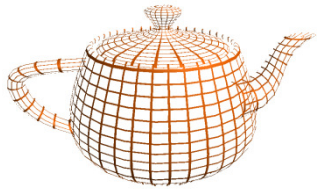
View Frustum Culling

- Coerência Translação-Rotação (Assarsson and Möller)
  - ex: se um objecto é rejeitado pelo plano esquerdo e o VF realiza uma rotação para a direita então o objecto continua fora do VF.
  - ex: se um objecto é rejeitado pelo *near plane*, i.e. encontra-se atrás do VC e o movimento da câmara for uma translação para a "frente", então o objecto pode imediatamente ser rejeitado.



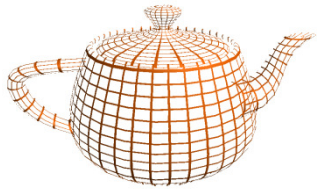
View Frustum Culling

- Coerência Temporal (Assarsson and Möller)
  - Guardar, para os objectos rejeitados, o plano utilizado na ultima rejeição.
  - Este plano deverá ser o primeiro a ser testado.



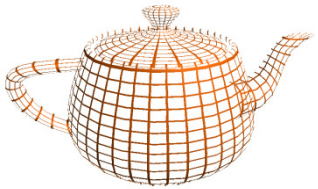
View Frustum Culling

Demo Frustum Culling



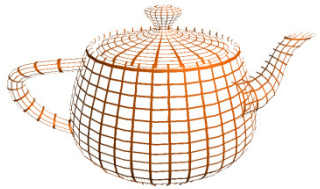
View Frustum Culling

- Mas...
- podemos ter milhões de triângulos =>
- milhões de testes =>
- provavelmente custo do teste é superior ao custo do desenho!



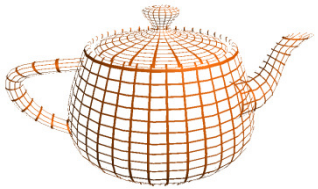
Partições Hierárquicas

- **Bounding Volumes**
- Quadtrees e Octrees
- BSP



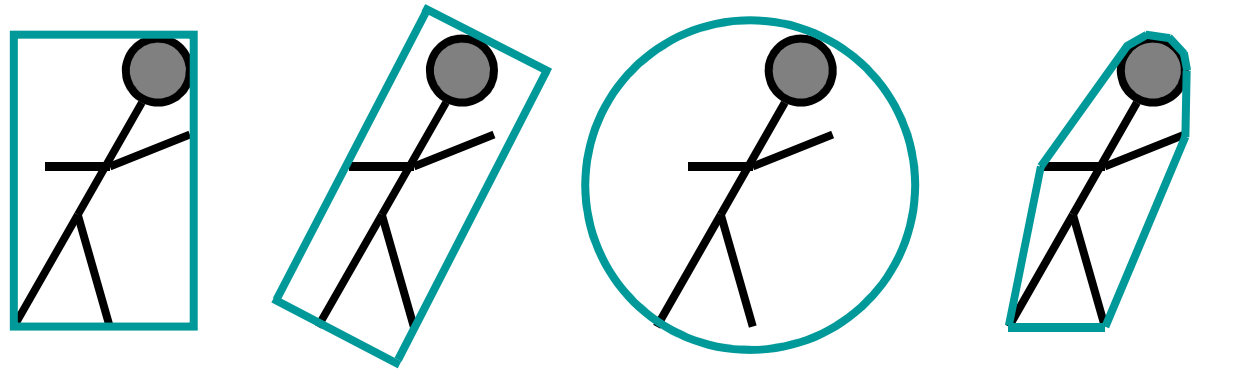
Bounding Volumes

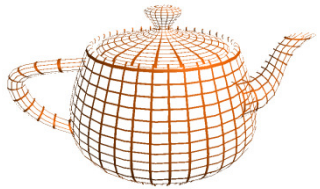
- Bounding Volumes:
  - Definição de Bounding Volumes para objectos: uma representação simplificada e totalmente envolvente do objecto.
  - O teste ao Bounding Volume permite eliminar de uma só vez todo o objecto.
  - Caso o Bounding Volume esteja parcialmente incluído, o que fazer?



Bounding Volumes

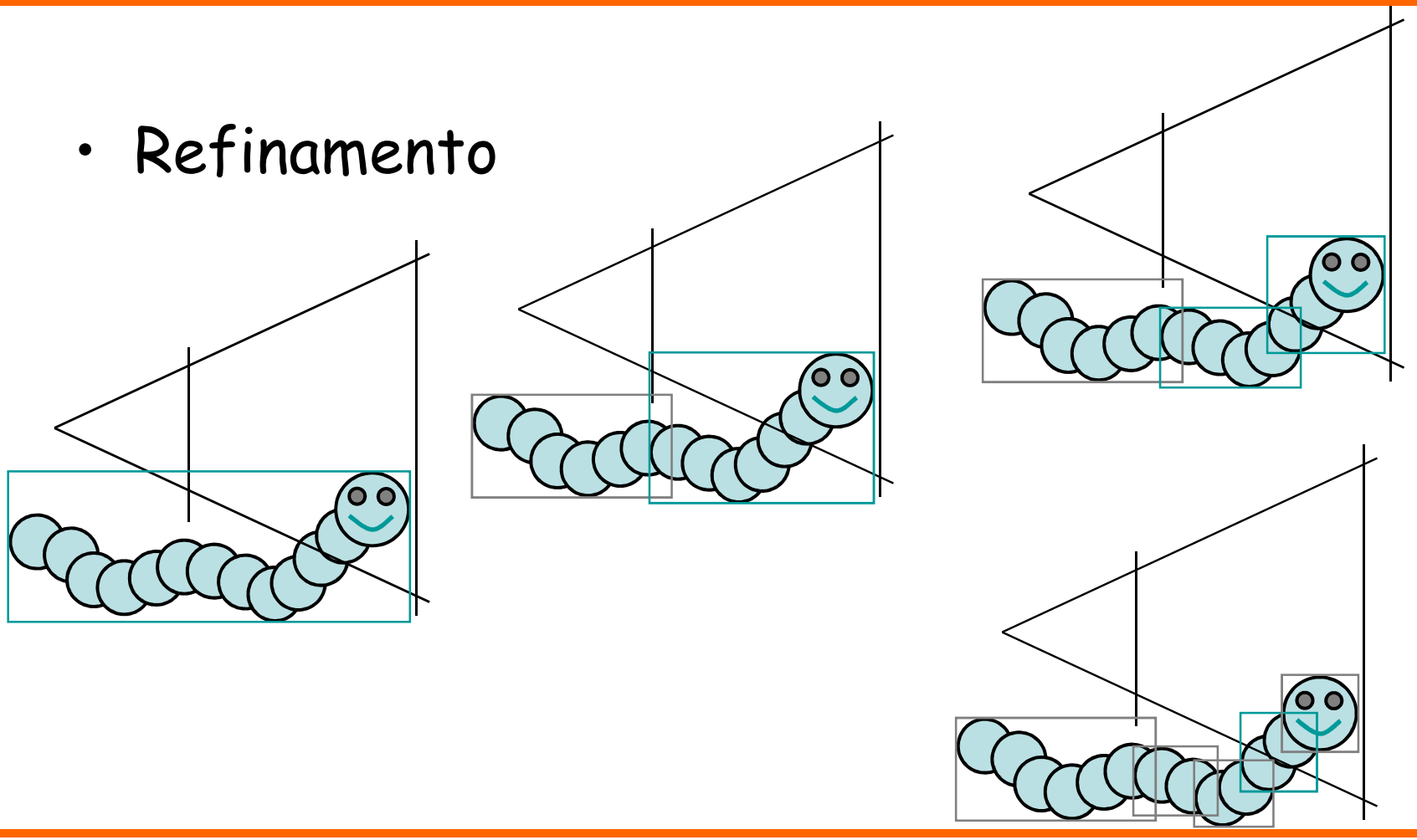
- Alguns Tipos de Volumes

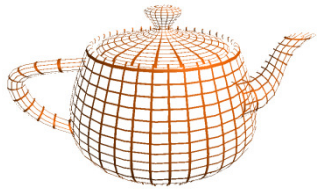




Bounding Volumes

- Refinamento

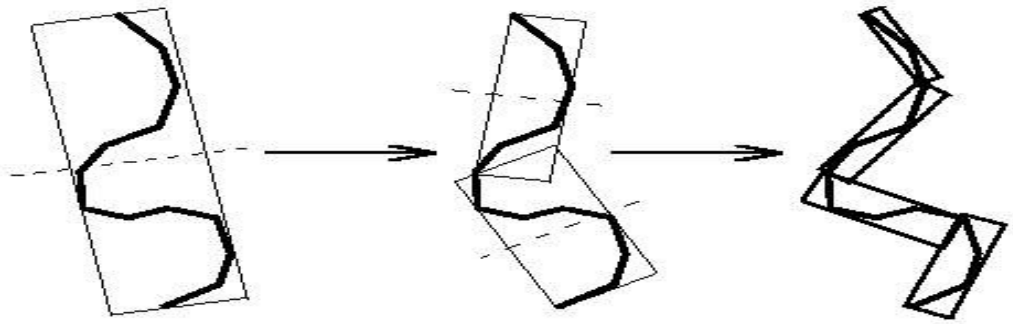


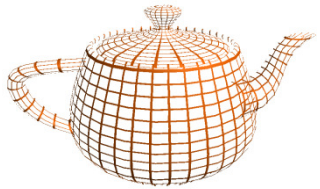


Bounding Volumes

- BV mais refinados => maior a probabilidade de rejeição num teste de visibilidade

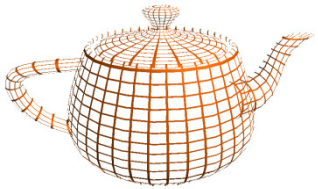
- mais testes a realizar, potencialmente menos polígonos a desenhar



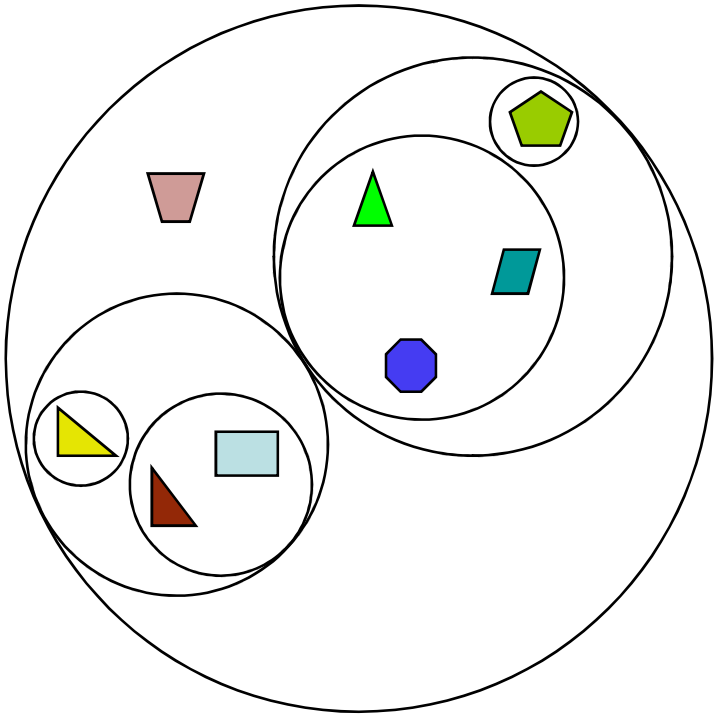


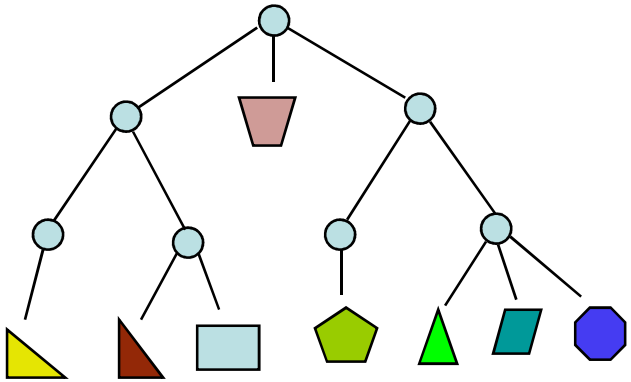
Bounding Volumes Hierárquicos

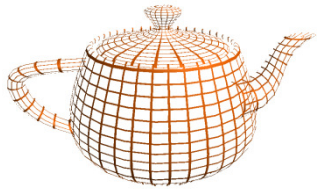
- Definição de Hierarquia em que um BV contêm um conjunto de objectos.
- Cada objecto tem o seu BV, contido no BV original.
- Cada objecto pode por sua vez ser dividido em sub objectos, cada sub objecto com um BV.



Bounding Volumes Hierárquicos

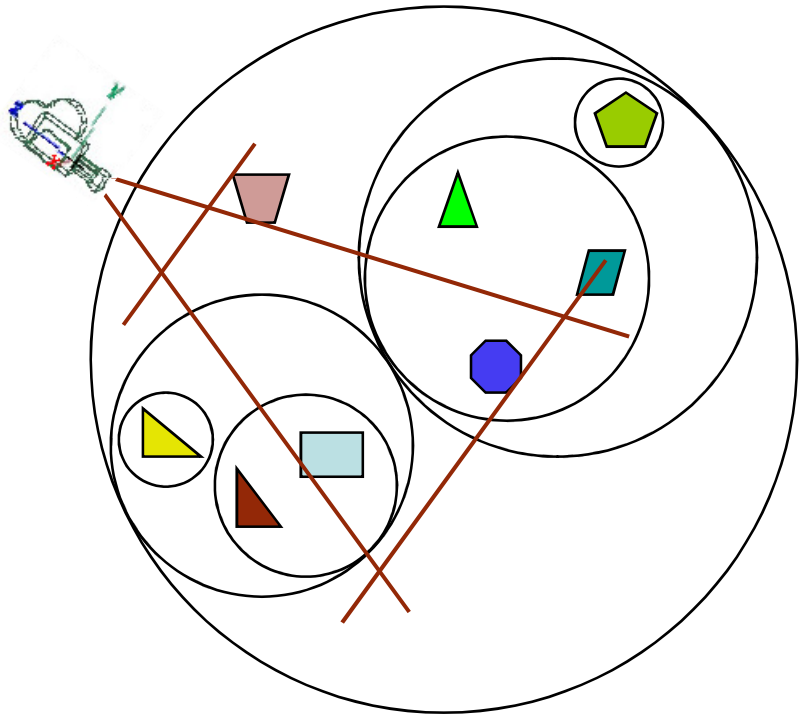


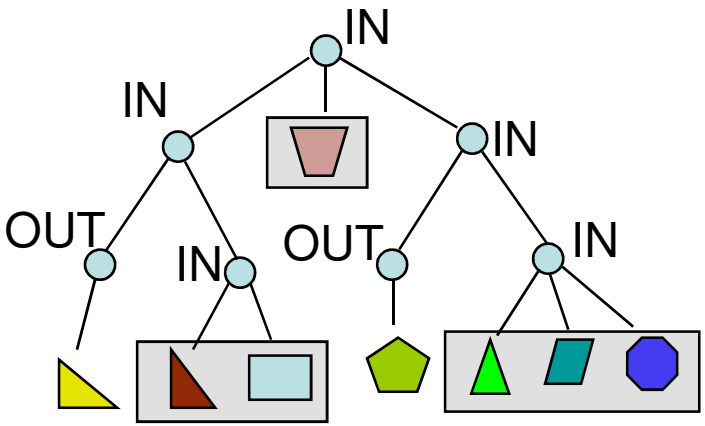


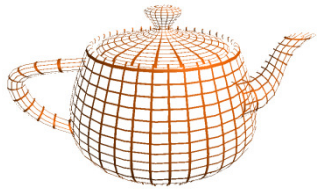


Bounding Volumes Hierárquicos

- View Frustum Culling c/ BVH

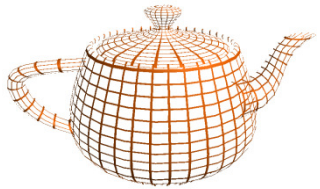






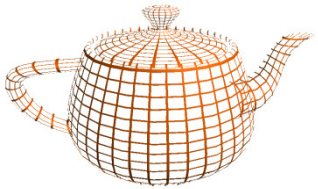
Bounding Volumes

- A solução baseada em Bounding Volumes assume a existência explicita de objectos na cena (objecto = { polígonos}).
- O que fazer nos casos em que não há este tipo de informação, ou seja temos uma "sopa" de polígonos?
- Solução: Partição Espacial



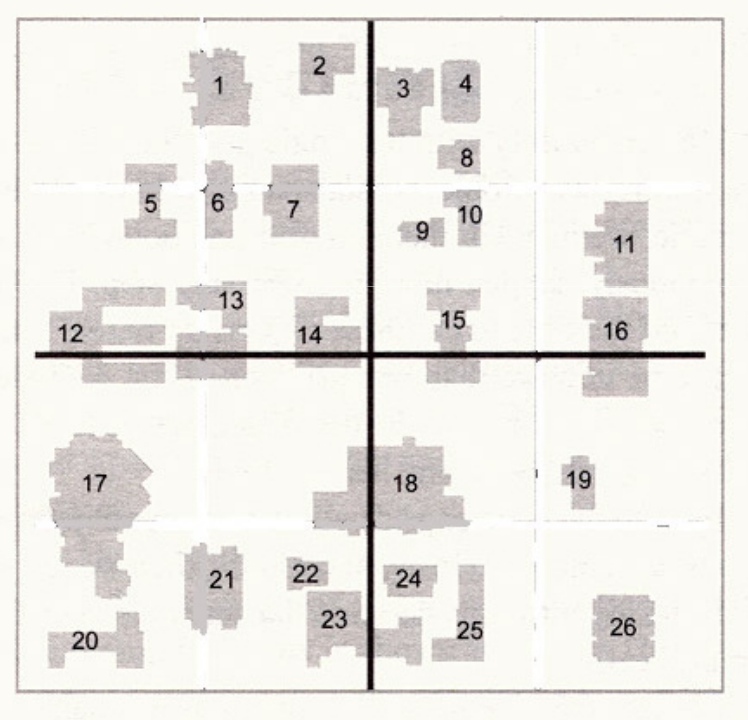
Partições Hierárquicas

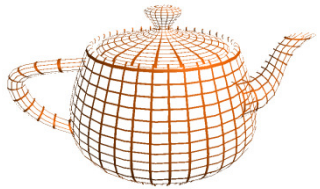
- Bounding Volumes
- **Quadtrees e Octrees**
- BSP



Partição Espacial - Quadtrees

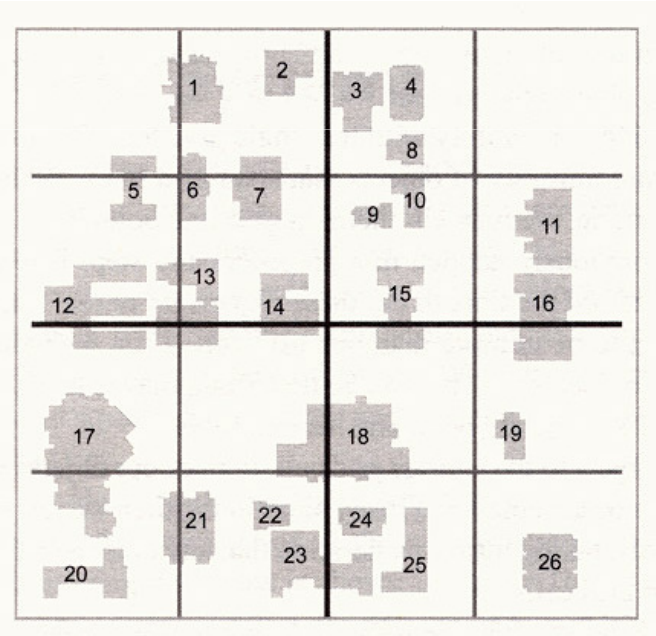
- Aplicável a objectos e "sopas" de polígonos.
- Inicialmente o mundo é dividido em 4 blocos



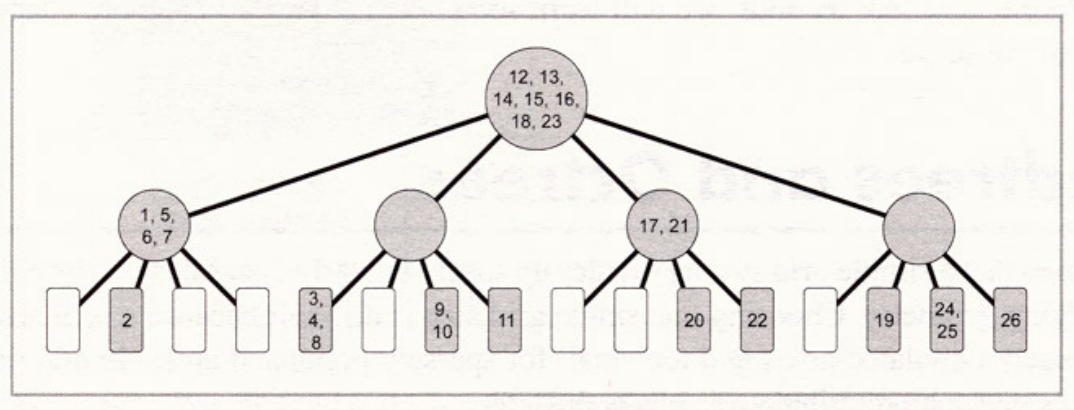


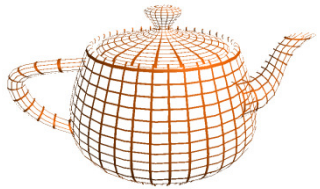
Partição Espacial - Quadtrees

- Cada bloco é recursivamente dividido em quatro blocos



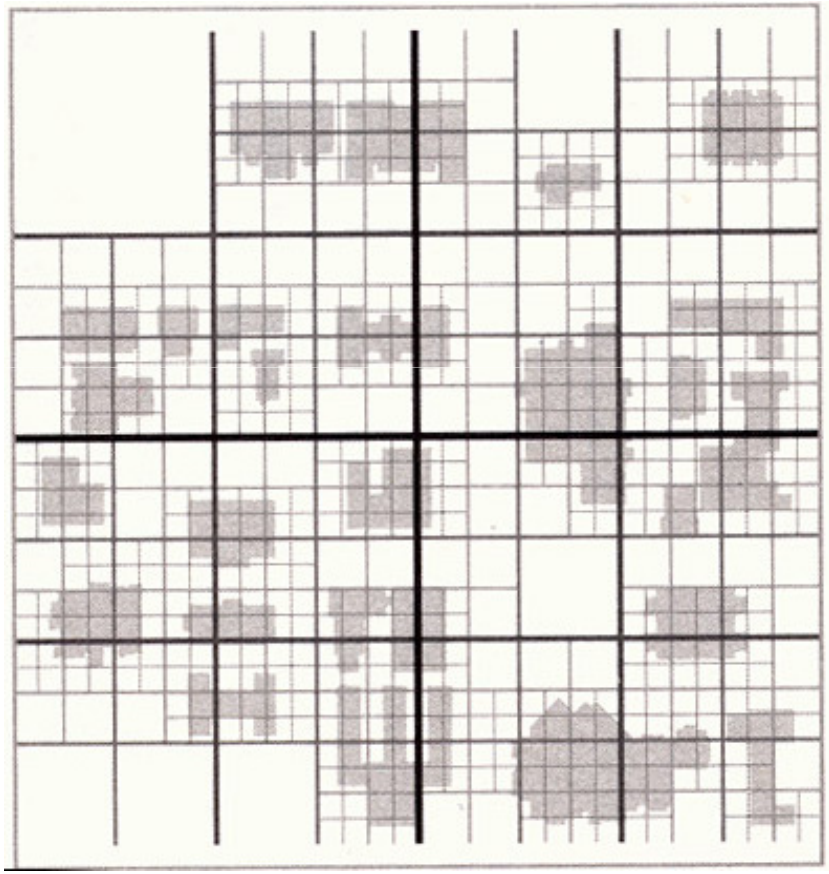
Representação da árvore quaternária

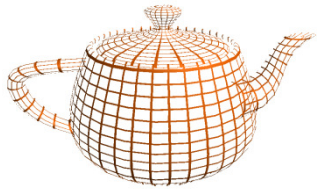




Partição Espacial - Quadtrees

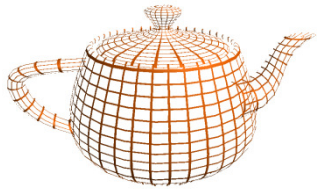
- A divisão não é necessariamente homogénea





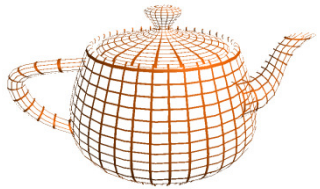
Partição Espacial - Quadtrees

- Critérios de paragem da subdivisão:
  - Número de polígonos na célula é menor que um determinado limite;
  - A profundidade da árvore atingiu um determinado limite;
  - A dimensão da célula é demasiado pequena.



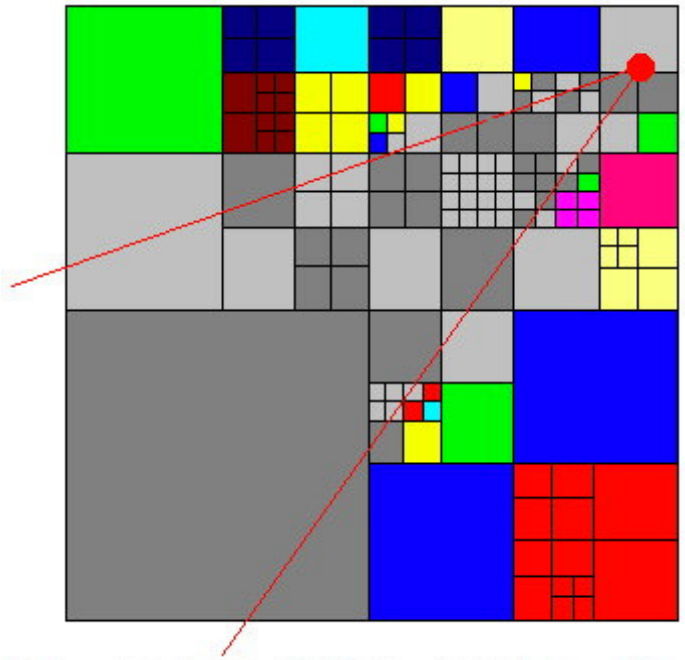
Partição Espacial - Quadtrees

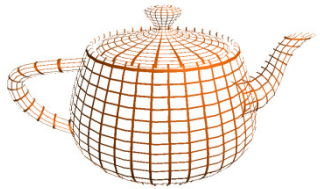
- Que fazer com os objectos/polígonos que ocupam duas ou mais células?
  - Incluí-los na célula mãe;
  - Incluí-los em cada uma das células;
  - Dividi-los de forma a que cada parte só ocupe uma célula



Partição Espacial - Quadtrees

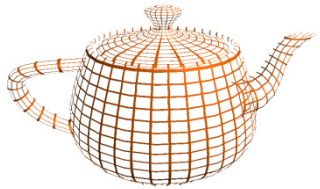
- View Frustum Culling com Quadtrees





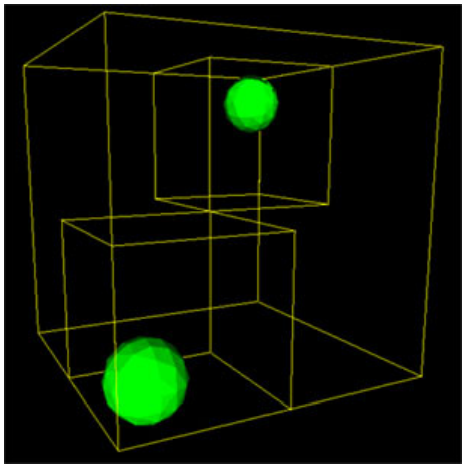
Partição Espacial - Quadtrees

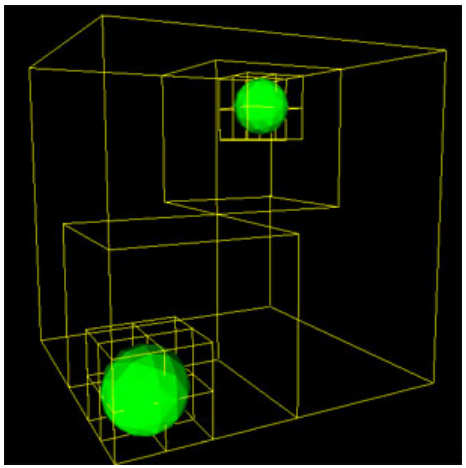
- As Quadtrees são indicadas para mundos "planos".
- Um exemplo são os terrenos.
- E os mundos em que a complexidade se reflecte nas 3 dimensões?
  - Octrees: generalização para 3D!

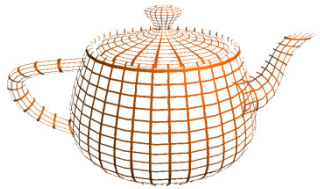


Partição Espacial - Octrees

- Generalização para 3D:
  - O mundo é dividido em 8 cubos.
  - Cada cubo pode ser por sua vez sub dividido.

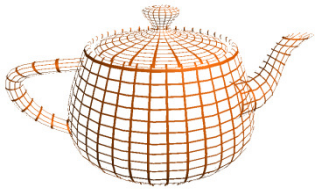






Partição Espacial - Octrees

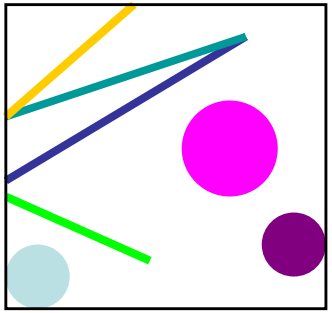
Demo - Construção de uma Octree



BVH vs. Spatial Partitioning (Kenneth E. Hoff III)

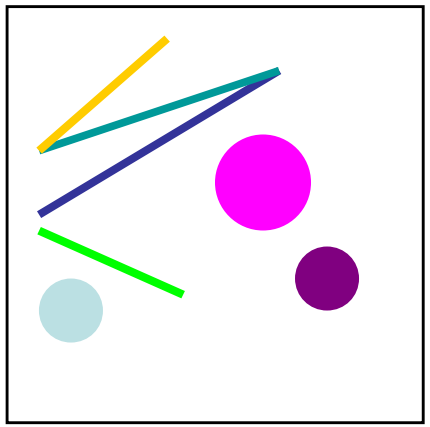
Bounding Volume Hierarchies

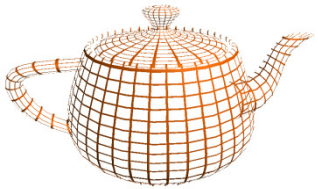
- Tightly fits objects
- Redundant spatial representation



Spatial Partitioning

- Tightly fills space
- Redundant object representation

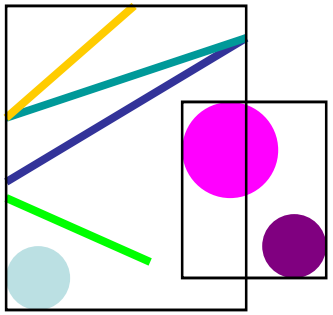




BVH vs. Spatial Partitioning (Kenneth E. Hoff III)

Bounding Volume Hierarchies

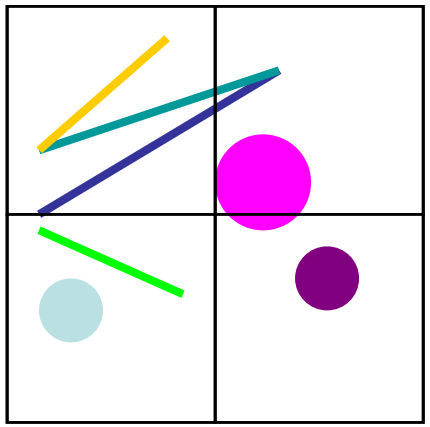
- Tightly fits objects
- Redundant spatial representation



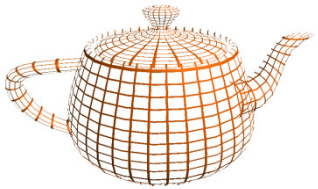
Volumes overlap multiple objects

Spatial Partitioning

- Tightly fills space
- Redundant object representation



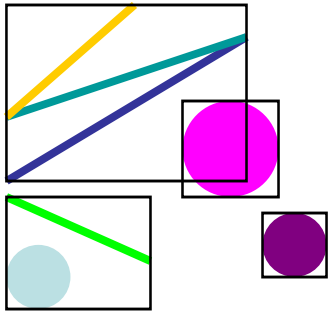
Objects overlap multiple volumes



BVH vs. Spatial Partitioning (Kenneth E. Hoff III)

Bounding Volume Hierarchies

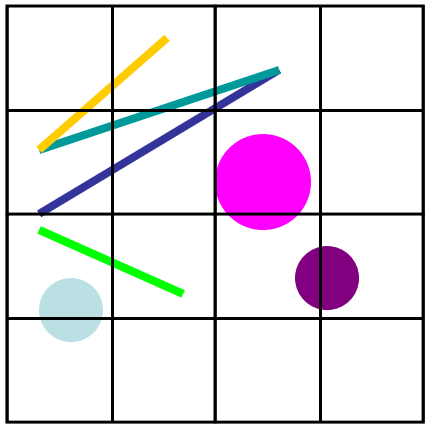
- Tightly fits objects
- Redundant spatial representation



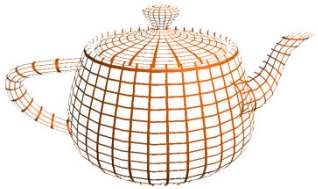
Volumes overlap multiple objects

Spatial Partitioning

- Tightly fills space
- Redundant object representation



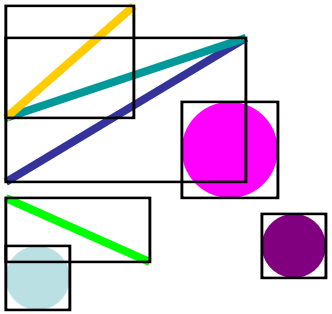
Objects overlap multiple volumes



BVH vs. Spatial Partitioning (Kenneth E. Hoff III)

Bounding Volume Hierarchies

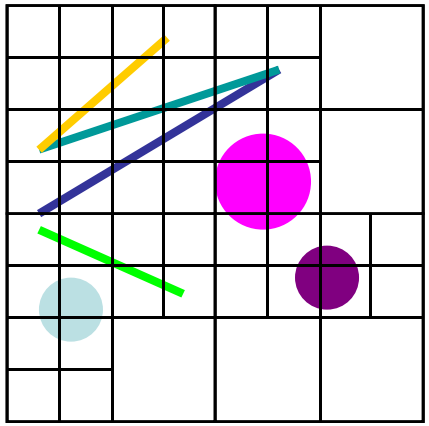
- Tightly fits objects
- Redundant spatial representation



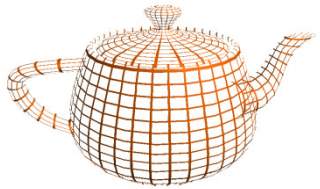
Volumes overlap multiple objects

Spatial Partitioning

- Tightly fills space
- Redundant object representation

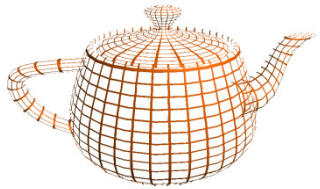


Objects overlap multiple volumes



Partições Hierárquicas

- *Masking* (Assarsson and Möller)
  - Se um objecto, parcialmente dentro do VF (i.e. é necessário testar os filhos) está completamente do lado correcto de um plano, os seus filhos também estão, logo...
  - => este plano não precisa de ser testado nos respectivos sub-objectos.



Referências

- **OpenGL Programming Guide**, OpenGLARB
- **Fast Backface Culling using Normal Masks**, Zhang and Hoff
- **Fast Extraction of Viewing Frustum Planes from the World-View-Projection Matrix**, Gil Gribb and Klaus Hartmann, http://www2.ravensoft.com/users/ggribb/plane%20extraction.pdf
- "Optimized View Frustum Culling Algorithms" Ulf Assarsson and Tomas Möller