LEI / LCC
Departamento de Informática
Universidade do Minho

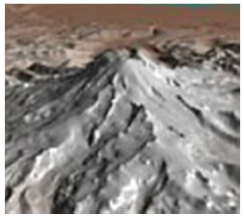
Visualização e Navegação em Tempo Real

Back Face e View Frustum Culling
Particação Espacial

O Problema: Triângulos!

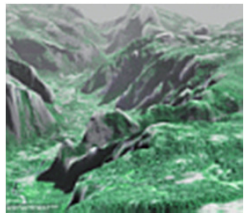
buda: 1 milhão

central: 13 milhões

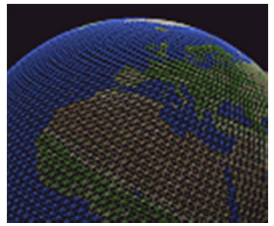


Terreno: 1.3 milhões

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

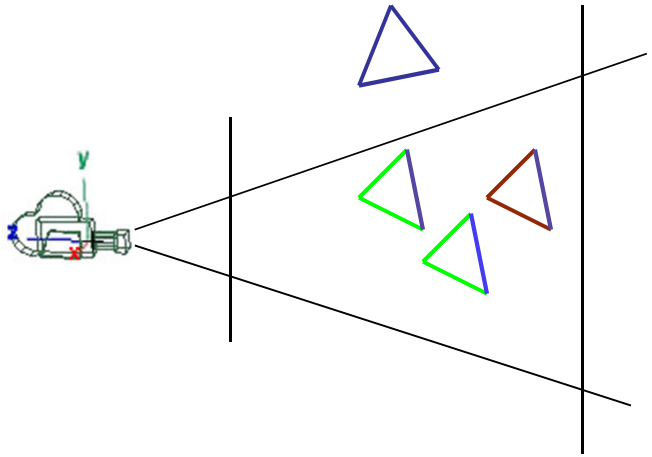
- Eliminação das partes que não contribuem para a cena final (culling)
  - *Back Face Culling*
  - *View Frustum Culling*
    - Bounding Volumes
    - Partição Espacial – Quad e Octrees
    - *Partição Espacial – BSP*
    - *Partição Espacial – Portais*
  - *Occlusion Culling*

Soluções

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

Culling

- Eliminação de polígonos que não contribuem para a imagem final



Polígonos Visíveis

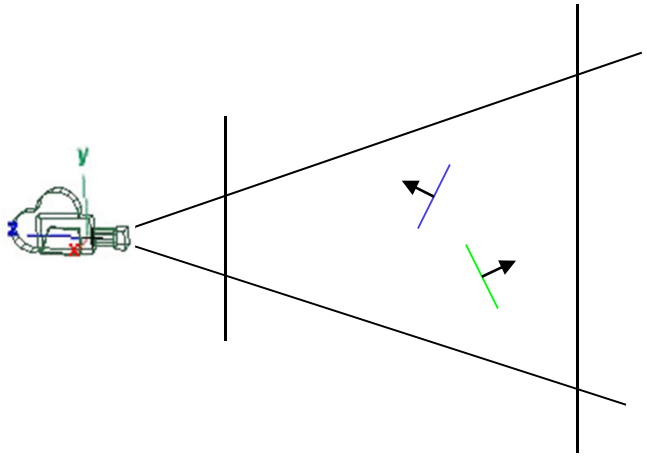
Back Face Culling

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

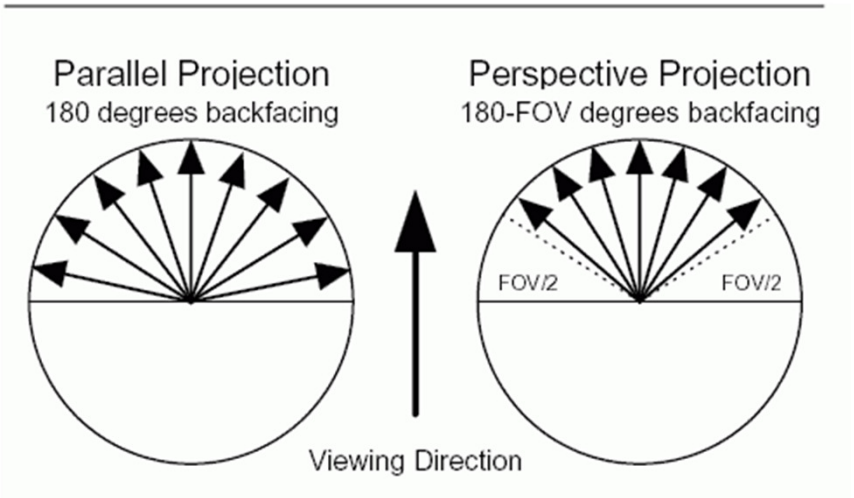
Back Face Culling

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

Back Face Culling

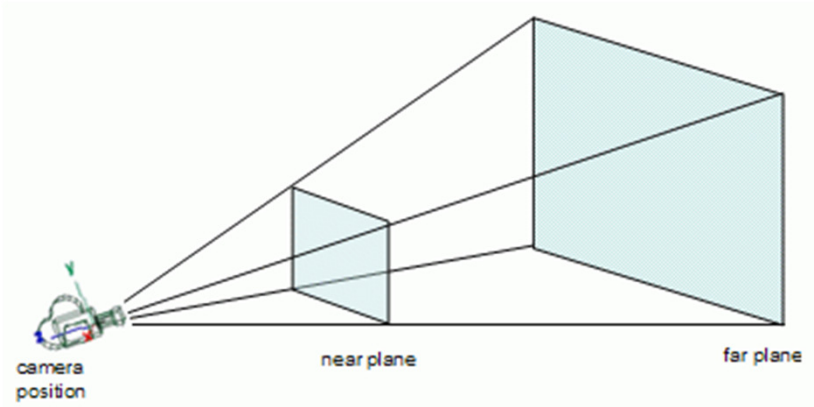
- Zhang and Hoff propõem:

  - Agrupar os polígonos de acordo com a direcção das suas normais
  - Eliminar grupos de polígonos de uma só vez



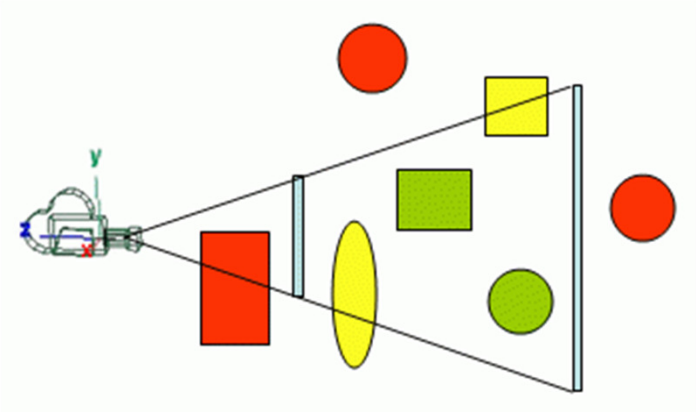
Culling

- 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

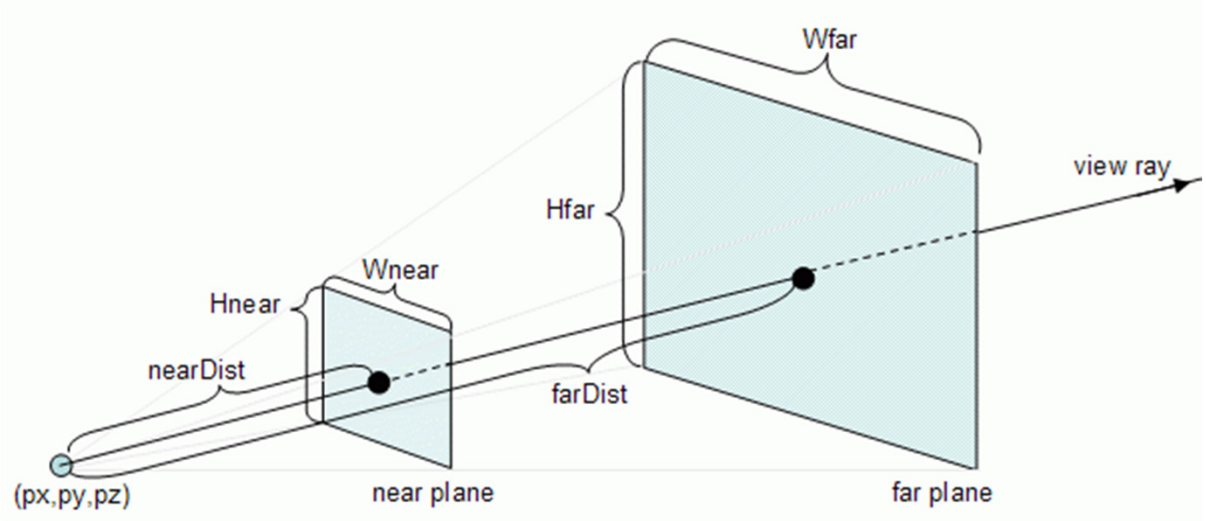
View Frustum Culling

- Etapas:
  - 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

- Propriedades e Medidas do View Frustum

```
gluPerspective(fov, ratio, nearDist, farDist);
gluLookAt(px,py,pz, lx,ly,lz, ux,uy,uz)
```



$$Hnear = 2 \times \tan(\frac{fov}{2}) \times nearDist$$

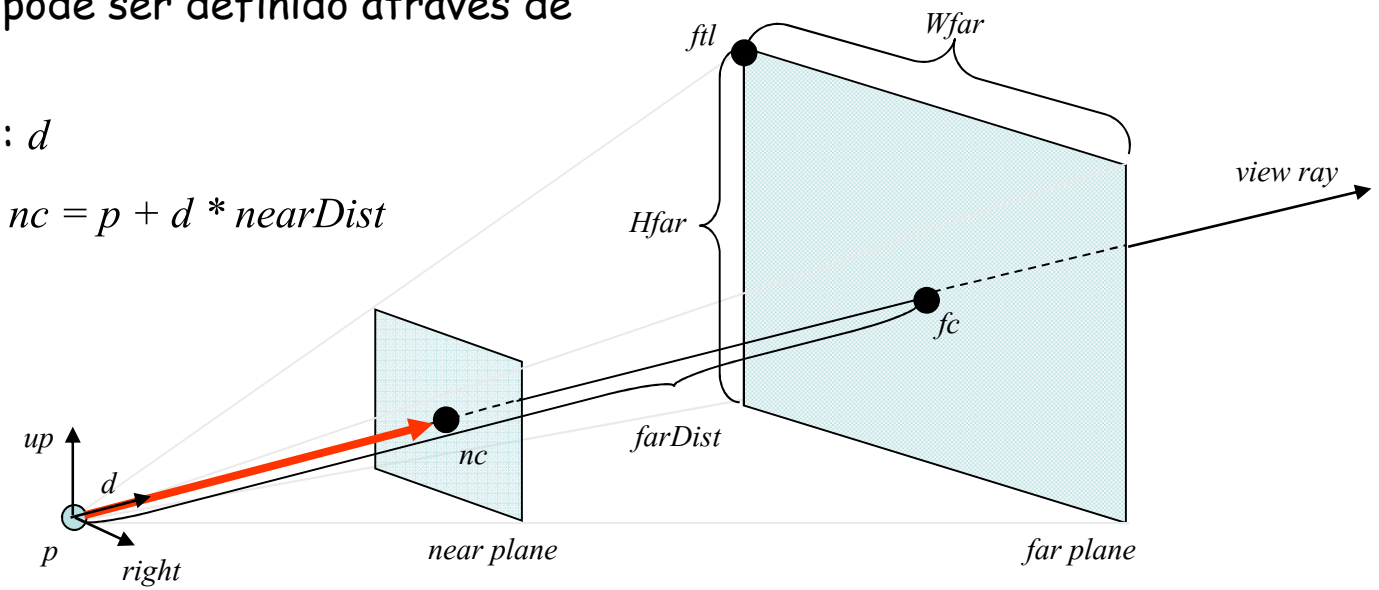
$$Wnear = Hnear \times ratio$$

$$Hfar = 2 \times \tan(\frac{fov}{2}) \times farDist$$

$$Wfar = Hfar \times ratio$$

View Frustum Culling

- Visão Geométrica
  - Uma normal e um ponto permitem definir um plano
  - *near plane* pode ser definido através de
    - normal: d
    - ponto: $nc = p + d * nearDist$

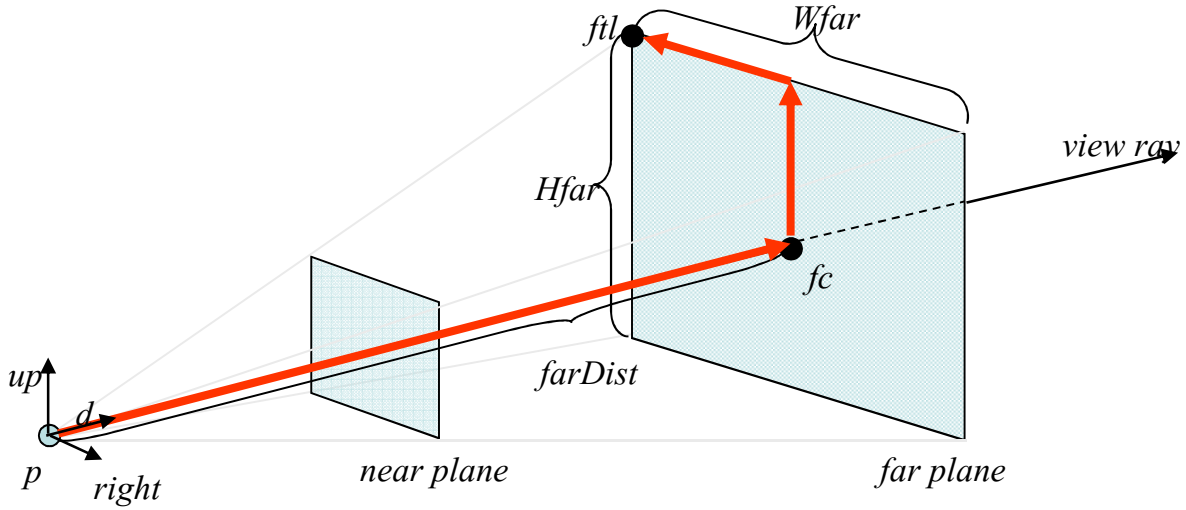


View Frustum Culling

- Visão Geométrica
  - três pontos permitem definir um plano

$$fc = p + d \times farDist$$

$$ftl = fc + (up \times \frac{Hfar}{2}) - (right \times \frac{Wfar}{2})$$



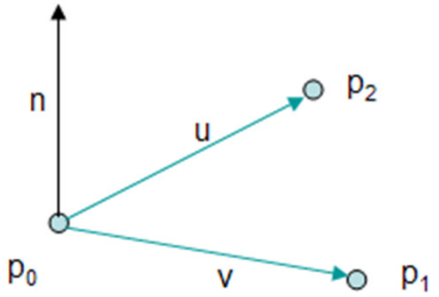
View Frustum Culling

- Definição de um plano $Ax + By + Cz + D = 0$

  - Dados: um ponto e uma normal

    - normalizar $n = (n_x, n_y, n_z)$

$$A = n_x$$
$$B = n_y$$
$$C = n_z$$



Sabemos que p0 é um ponto do plano, logo

$$Ap_{0x} + Bp_{0y} + Cp_{0z} + D = 0 \Leftrightarrow$$
$$D = -Ap_{0x} - Bp_{0y} - Cp_{0z} = -n \cdot p_0$$

View Frustum Culling

- Distância de um ponto ao plano

  - Assumindo que o plano foi definido como apresentado no slide anterior dizemos que a equação do plano é normalizada

  - Neste caso a distância de um ponto

    $p = (px, py, pz)$

  - ao plano

    $Ax + By + Cz + D = 0$

  - é definida por

$$dist = Ap_x + Bp_y + Cp_z + D$$

$dist > 0$ => ponto encontra-se no lado determinado pela direcção da normal

View Frustum Culling

- Teste
  - ponto está dentro do frustum?
    - Assumindo que as normais dos planos apontam para o interior do frustum,

```
int FrustumG::pointInFrustum(Vec3 &p) {

          int result = INSIDE;
        for(int i=0; i < 6; i++) {
                if (pl[i].distance(p) < 0)
                        return OUTSIDE;
        }
        return(result);
}
```

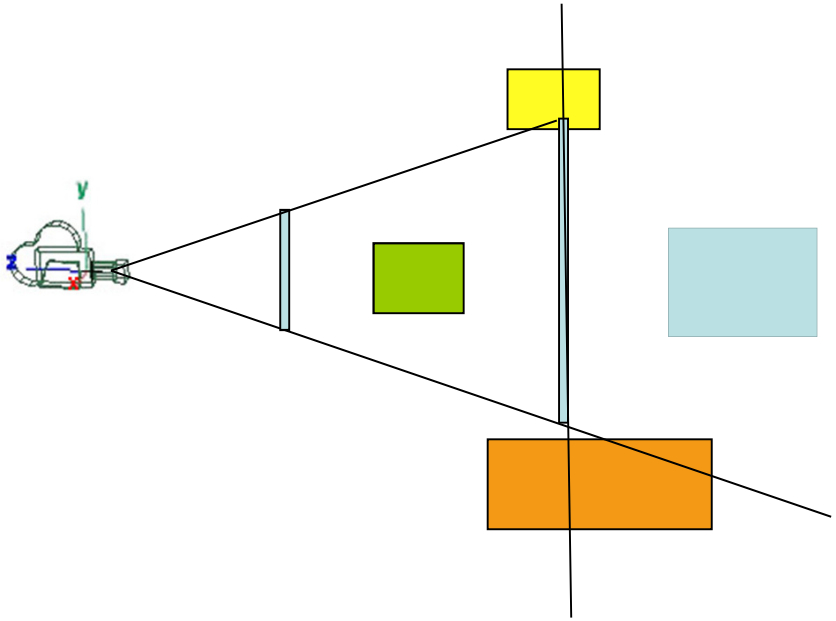
View Frustum Culling

- Teste
  - Esfera está total ou parcialmente dentro do view frustum?

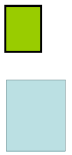
```
int FrustumG::sphereInFrustum(Vec3 &p, float radius) {
        float distance;
        int result = INSIDE;

        for(int i=0; i < 6; i++) {
                distance = pl[i].distance(p);
                if (distance < -radius)
                        return OUTSIDE;
                else if (distance < radius)
                        result = INTERSECT;
        }
        return(result);
}
```

View Frustum Culling

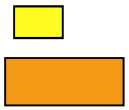
- Teste
  - Paralelepípedos: Teste aos cantos do objecto



Casos Simples :

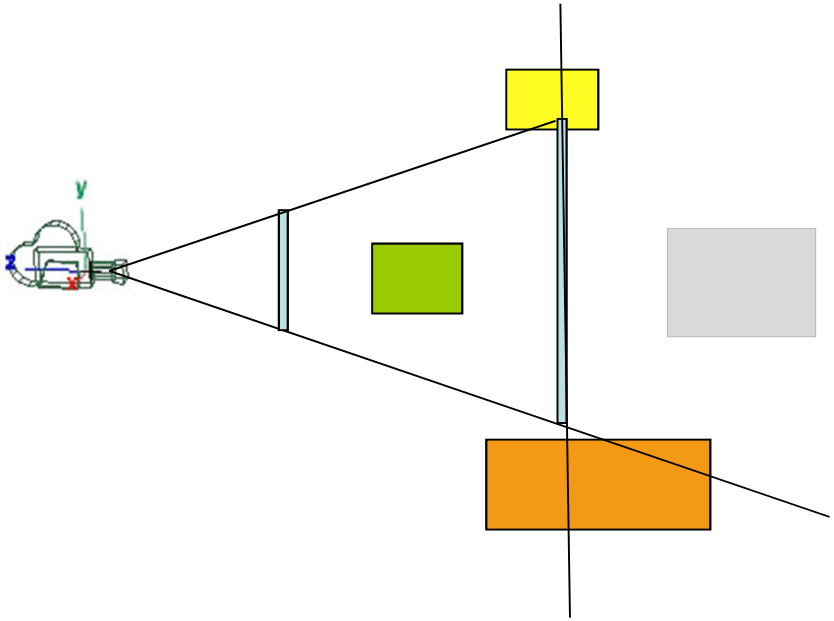


E os casos :



View Frustum Culling

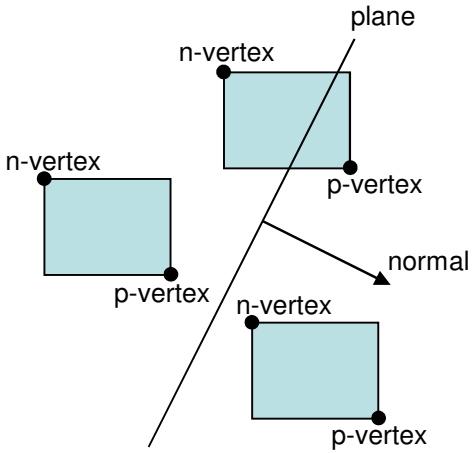
- Teste
  - Paralelepípedos: Teste aos cantos do objecto



Aceitar todos os paralelepípedos cujos pontos não se encontram do lado "errado" do mesmo plano

View Frustum Culling

- Teste
  - paralelepípedos




```
p = (xmin,ymin,zmin)
if (normal.x >= 0)
        p.x = xmax;
if (normal.y >=0))
        p.y = ymax;
if (normal.z >= 0)
        p.z = zmax;
```

```
n = (xmax,ymax,zmax)
if (normal.x >= 0)
        n.x = xmin;
if (normal.y >=0))
        n.y = ymin;
if (normal.z >= 0)
        n.z = zmin;
```

View Frustum Culling

- Operação pode também 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

$$A = P \times M$$
$$p' = Ap$$

  - 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 A = P * M

```
glPushMatrix();

glLoadMatrixf(P);
glMultMatrixf(M);
glGetFloatv(GL_MODELVIEW_MATRIX, A);

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 $p'=(x',y',z',w') = Ap$ é o ponto em Clip Space.
  - p' está dentro do volume de visualização se:

$$-w' < x' < w'$$
$$-w' < y' < w'$$
$$-w' < z' < w'$$

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 $p'=Ap=(x',y',z',w')$.
  - Sabemos que
    - $-w' < x' < w'$

View Frustum Culling

- World Space: Setup
  - Seja $A = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix}$

$$p' = A \times p = \begin{bmatrix} l_1 \times p \\ l_2 \times p \\ l_3 \times p \\ l_4 \times p \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

então (em Clip Space) se

$$-w' < x' < w'$$

temos (em World Space)

$$-p*l_4 < p*l_1 < p*l_4$$

View Frustum Culling

- World Space: Setup
  - $-p*l_4 < p*l_1 < p*l_4$
  - Se x está à direita do plano esquerdo então
    - $-p*l_4 < p*l_1$
    - $0 < p*l_1 + p*l_4$
    - $0 < p*(l_1 + l_4)$
    - $0 < x(a_{11}+a_{41}) + y(a_{12}+a_{42}) + z(a_{13}+a_{43}) + w(a_{14}+a_{44})$

Nota: os índices são (linha,coluna)

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.
- Teste é mais simples (resume-se às comparações)

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

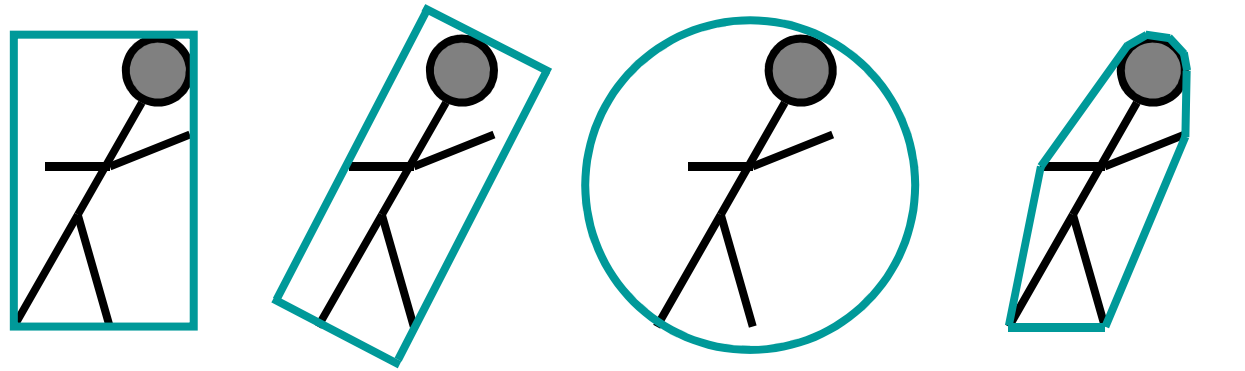
- **Bounding Volumes**
- Quadtrees e Octrees
- BSP

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?

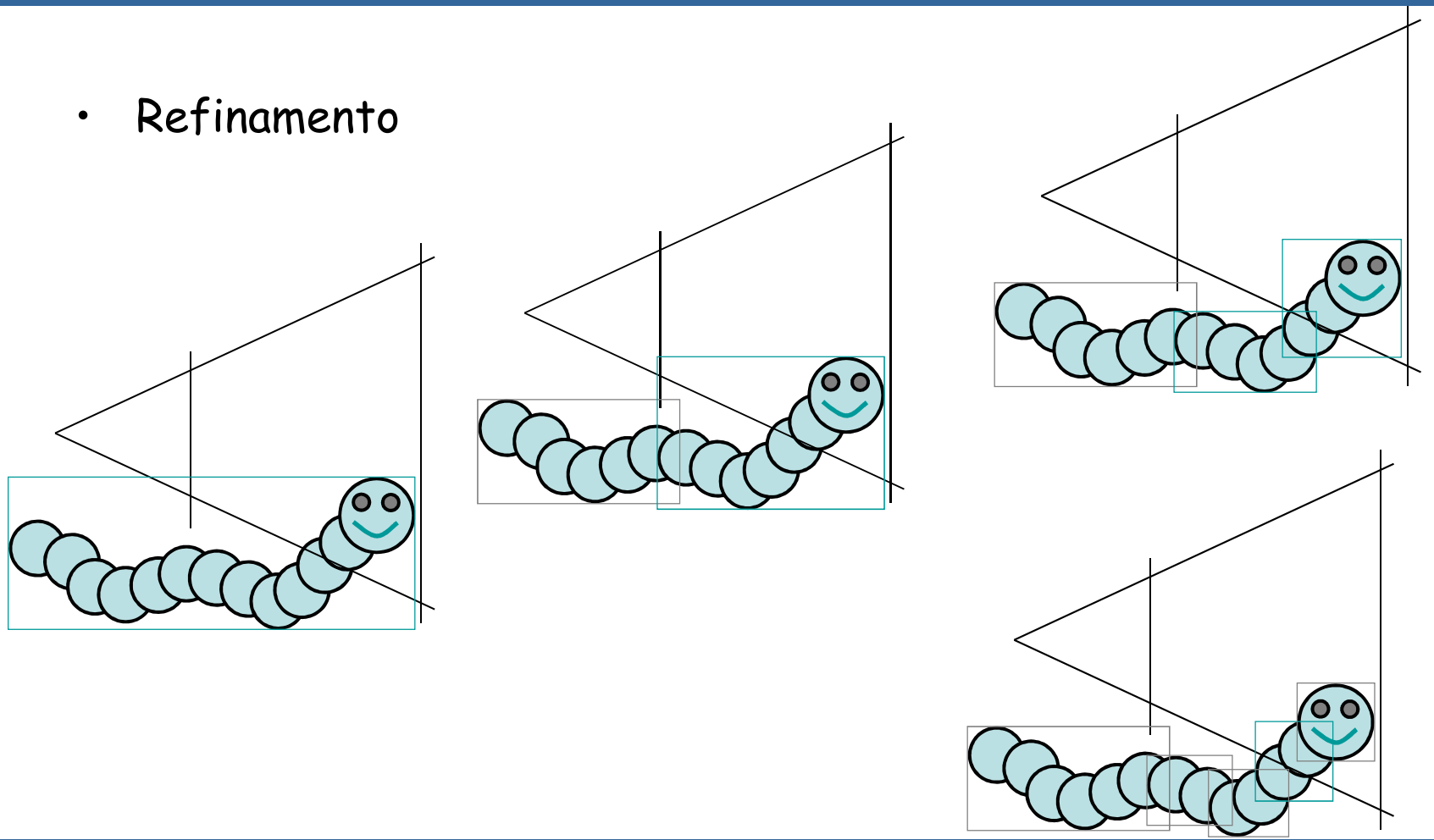
Bounding Volumes

- Alguns Tipos de Volumes



Bounding Volumes

- Refinamento

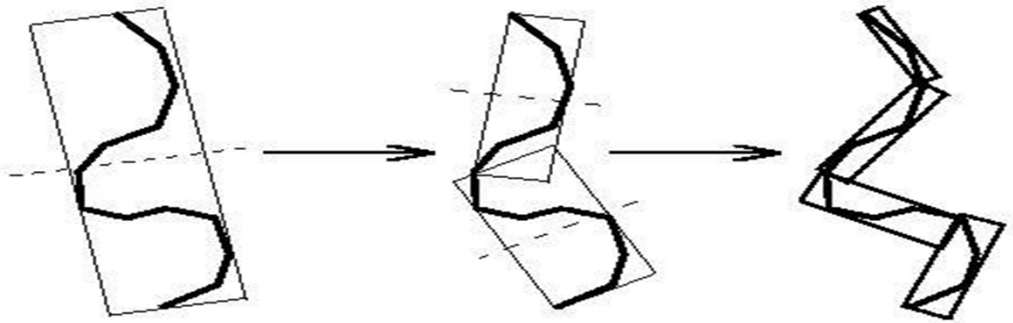


Bounding Volumes

- BV mais refinados

  => maior a probabilidade de rejeição num teste de visibilidade

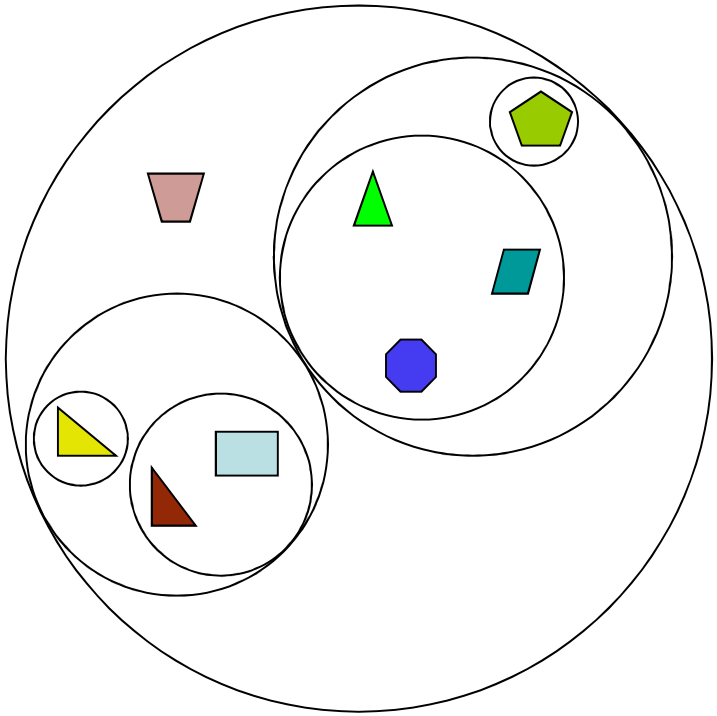
  => mais testes a realizar, potencialmente menos polígonos a desenhar

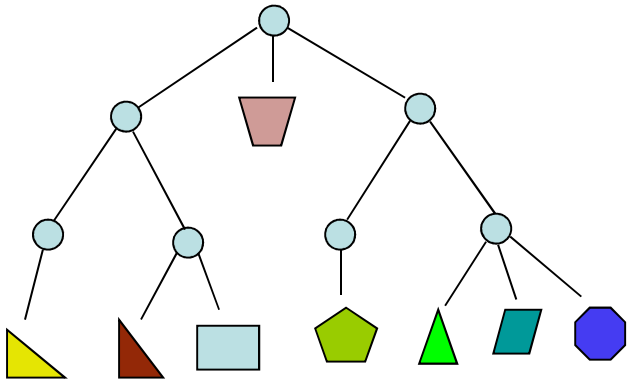


Bounding Volumes Hierárquicos

- 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.

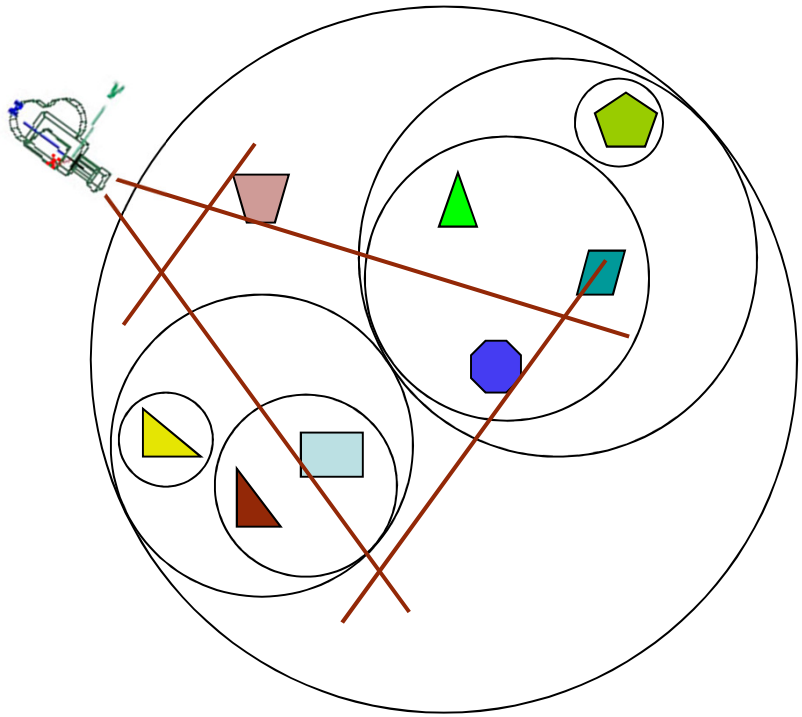
Bounding Volumes Hierárquicos

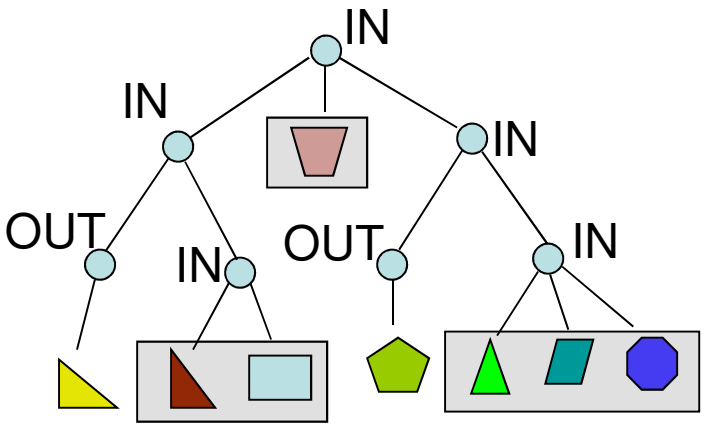




Bounding Volumes Hierárquicos

- 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?

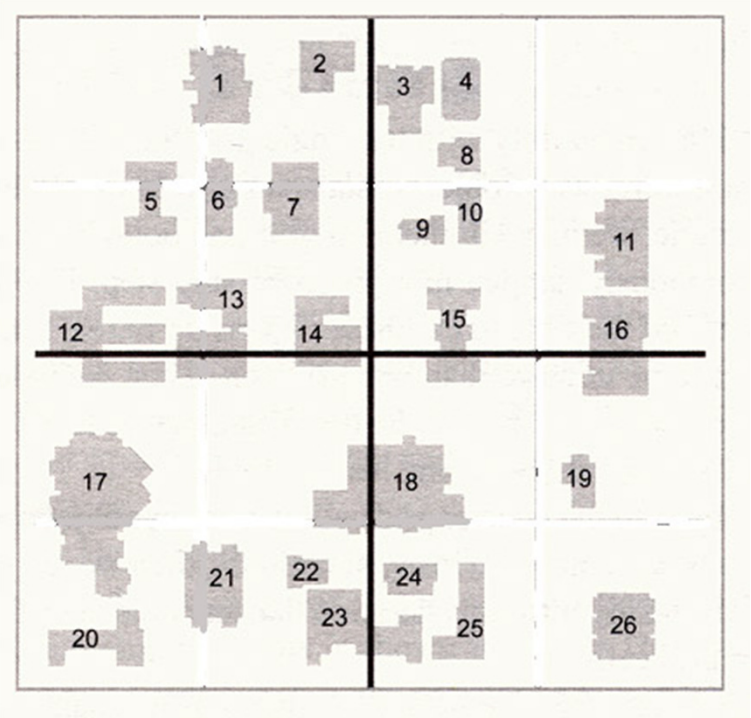
- Solução: Partição Espacial

Partições Hierárquicas

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

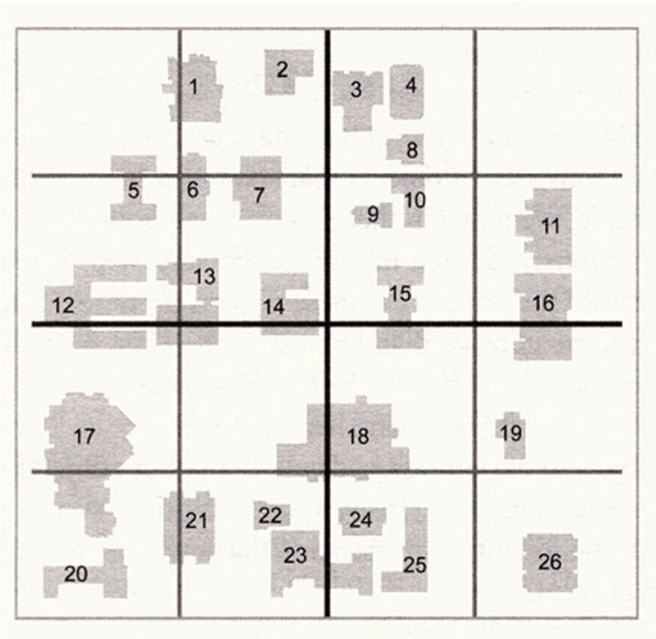
Partição Espacial - Quadtrees

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

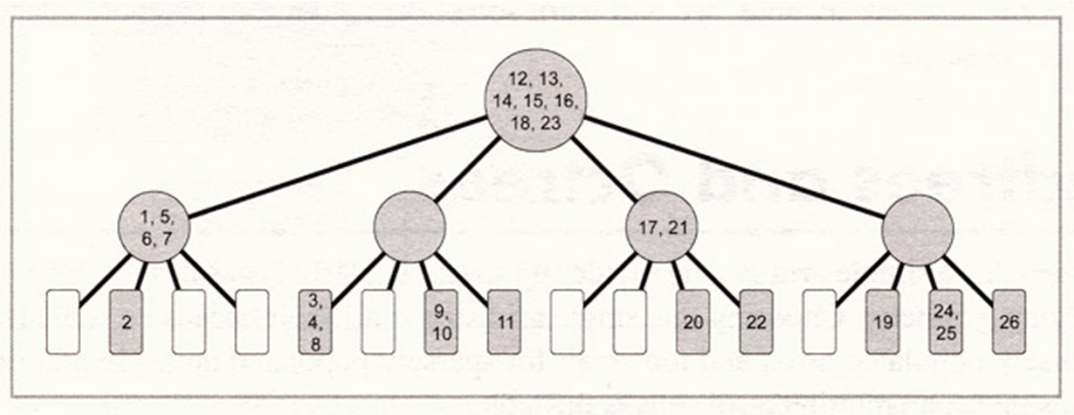


Partição Espacial - Quadtrees

- Cada bloco é recursivamente dividido em quatro blocos

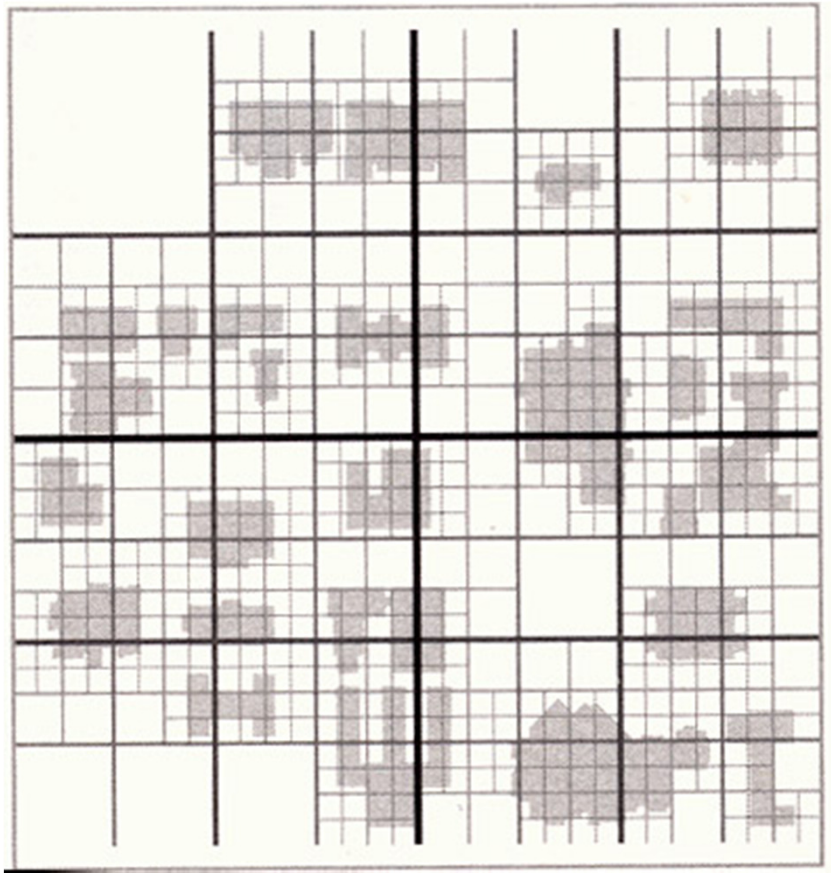


Representação da árvore quaternária



Partição Espacial - Quadtrees

- A divisão não é necessariamente homogénea



Partição Espacial - Quadtrees

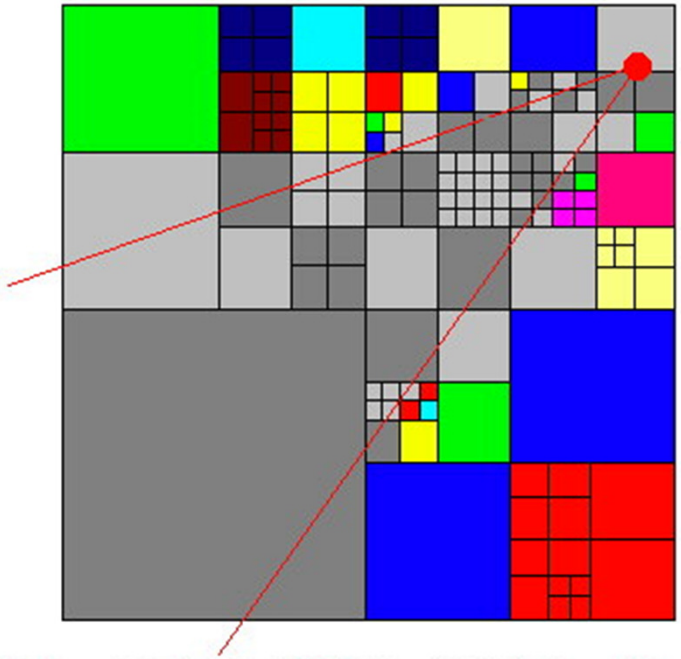
- Critérios de paragem da subdivisão:
  - Número de polígonos na célula é menor que um determinado limite;
  - A profundidade da árvore atingiu um determinado limite;
  - A dimensão da célula é demasiado pequena.

Partição Espacial - Quadtrees

- Que fazer com os objectos/polígonos que ocupam duas ou mais células?
  - Incluí-los na célula pai;
  - 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

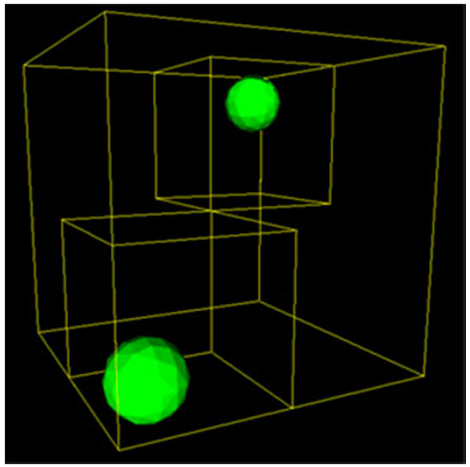


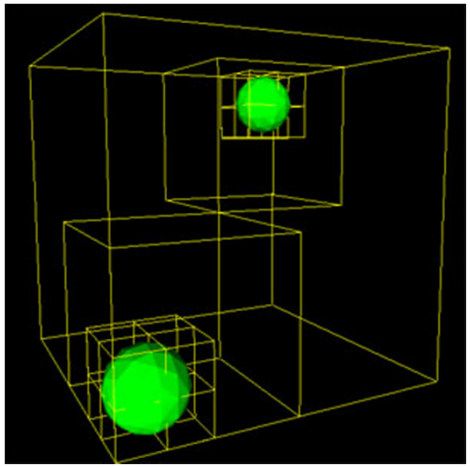
Partição Espacial - Quadtrees

- 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, octantes.
  - Cada cubo pode ser por sua vez sub dividido.





Partição Espacial - Octrees

Demo - Construção de uma Octree

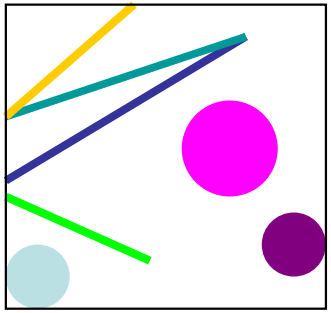
BVH vs. Spatial Partitioning (Kenneth E. Hoff III)

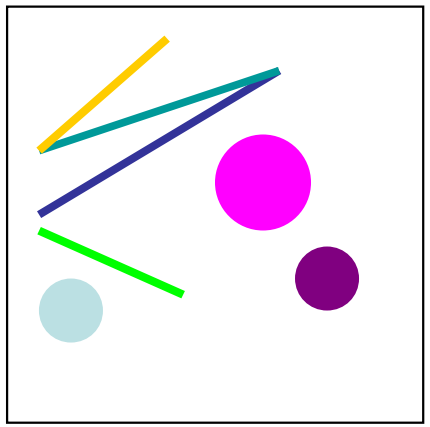
Bounding Volume Hierarchies

- Tightly fits objects
- Redundant spatial representation

Spatial Partitioning

- Tightly fills space
- Redundant object representation

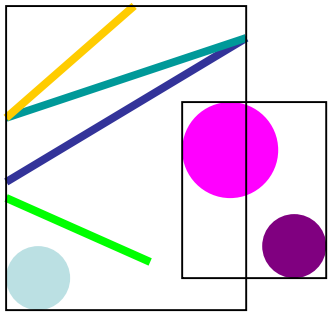




BVH vs. Spatial Partitioning (Kenneth E. Hoff III)

Bounding Volume Hierarchies

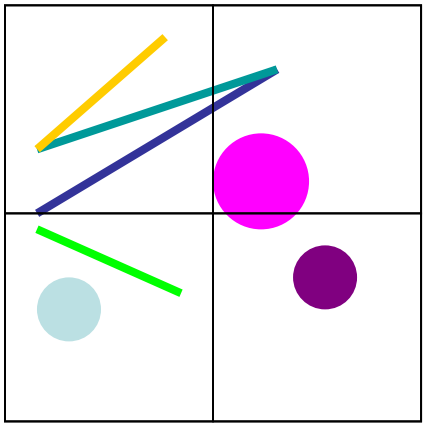
- Tightly fits objects
- Redundant spatial representation



Volumes overlap multiple objects

Spatial Partitioning

- Tightly fills space
- Redundant object representation

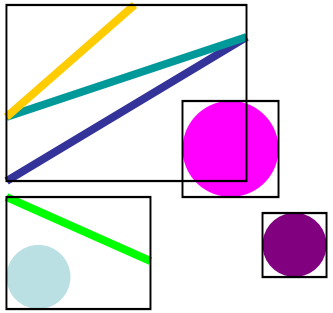


Objects overlap multiple volumes

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

Bounding Volume Hierarchies

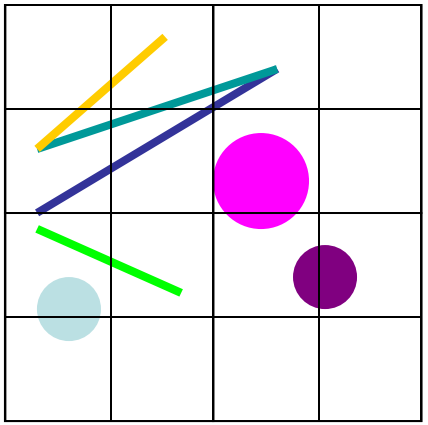
- Tightly fits objects
- Redundant spatial representation



Volumes overlap multiple objects

Spatial Partitioning

- Tightly fills space
- Redundant object representation

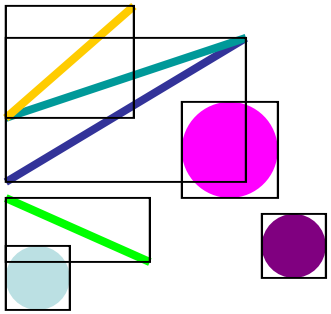


Objects overlap multiple volumes

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

Bounding Volume Hierarchies

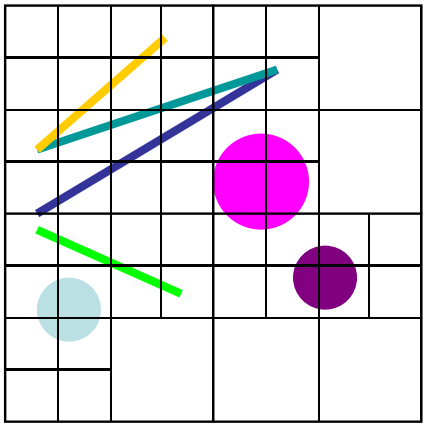
- Tightly fits objects
- Redundant spatial representation



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