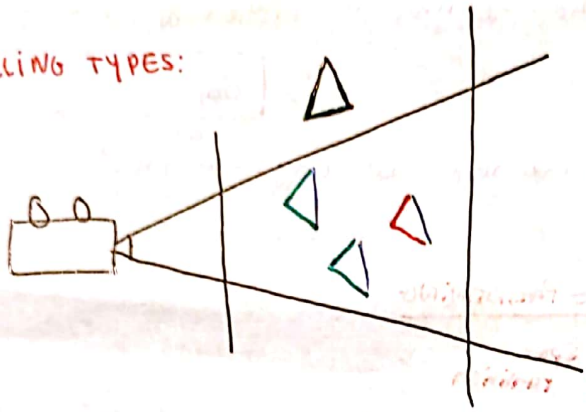
Notes on "Culling e Pontiros de Geometria" (CG)

- CULLING: Avoid (fully) processing every triangle/model
  - > Back face culling
  - > View frustum culling
  - > Occlusion culling.

- CULLING TYPES:



/ - Visible

/ - Back face culling

/ - View frustum culling

/ - Occlusion culling.

> BACK FACE CULLING

- Do not process triangles facing away from the camera.
  - Allows the elimination of a large number of triangles.
  - Performed in hardware for every triangle (∴ implies triangle submission).

> VIEW FRUSTUM CULLING.

- Eliminate triangle/object / volume outside the view frustum.

NOTES:

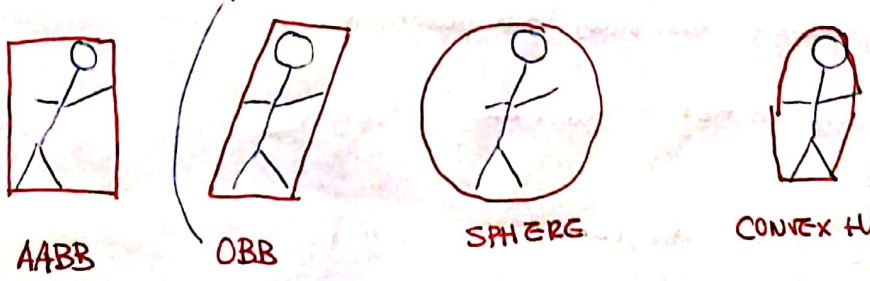
- Translation-Rotation Coherency (Assarson and Möller)

ex: if an object is reject by the near plane and the camera moves forward, then the object will still be outside the frustum.

- Temporal Coherency (Assarson and Tuilee)
  - Store for each object the plane that caused it to be rejected
  - The stored plane should be the first to be tested.

- BOUNDING VOLUMES

Cf. 39



(axis aligned bounding box)

- Testing the Bounding Volumes allows the elimination of complex geometry. w/ simple tests

- **Bounding Volume Granularity.**
  - greater probability of rejection since we have less "empty space".
  - more tests are required, potentially less triangles are drawn.

A Bounding Voluming based solution requires the explicit definition of objects

object = {triangles}

- What if our scene is a "triangle soup", w/ any semantics?

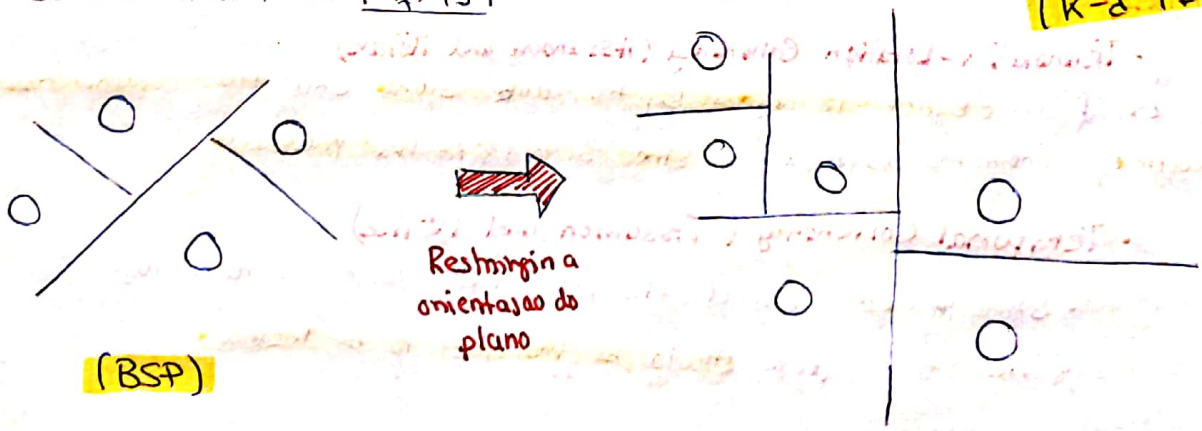
solution: SPACE PARTITIONING

- SPACE PARTITIONING → BSP (Binary Space Partition)
  - Using planes to recursively split the world in two.
  - Results in a binary tree
  - the planes can be arbitrary.

→ K-D trees

- Similar to BSPs but the planes are ⊥ to the axes
- Build a k-d tree: cf. 49

(K-d TREE)



(BSP)

→ Quadtrees

Divide the world recursively into quadrants.

- The recursion is not homogeneous.

→ Octrees

Recursively divide the world into octants

BVH (Bounding Volume Hierarchies) VS SPACE PARTITIONING

- Tightly fits objects
- Redundant spatial representation

Volumes overlap multiple objects

- Tightly fills space
- Redundant object representation

Objects overlap multiple volumes

- Hierarchical Partition

→ MASKING (Assarson and Möller)

- Considering an object partially inside the VF (View frustum), then the child nodes must be tested.
- If the object is completely on the inside of a plane, then
  ⇒ it's child nodes will also be on the inside of the same plane, i.e, the plane does not need to be tested.

—// EXERCICES //—

⑥ Cf. PP 14, 15, ... → ALGORITMO PI EXMAIR OS PLANOS DO VIEW FRUSTUM SEGUNDO A VISÃO GEOMÉTRICA.

R: - Precisamos de ir buscar os cantos do frustum, ou seja, a partir da posição da câmara, temos que viajar ao longo da direção de visualização (view vector) e depois temos que subir ou descer pl ir buscar os pontos.

- Só usamos variáveis que estejam nas duas instruções (cf. 14)

- A partir daí temos os planos (cf. 17) que se vinam fazendo:
  - usando alguns pontos, temos os vetor u e v
  - fazemos o o produto externo, obtemos a normal (n)
  - a partir da normal e de um pt do plano temos a eq. do plano
- Agora faz-se o teste da pág. 19.

⑦ ALGORITMO PI EXMAIR OS PLANOS DO VIEW FRUSTUM EM CLIP SPACE

R: No clip Space tudo o q é visível está dentro de um cubo com coordenadas de -1 a 1. Para testar se é necessário exmair ou não, temos de verificar se se encontra dentro dos limites:

Converts points from Local Space to Clip Space

cf. 26 $A = PVM$ → tem a info da câmara q está no View Matrix

$P' = AP$

P' point in Clip Space

cf. 29 $P' = (x', y', z', w') = AP$

P' está dentro do frustum se
$-w' < x' < w'$
$-w' < y' < w'$
$-w' < z' < w'$

① Compare os hab tipos de culling em termos computacionais

R:

- O Backface culling é feito sempre em hardware, triângulo a triângulo, o que significa que temos de submeter os triângulo p/ a placa e só depois é que dos podem ser eliminados. //APLICADO TRIÂNGULO A TRIÂNGULO

- O View Frustum culling em termos computacionais atua sobre objetos ou Bounding Boxes (Volumes Envolventes). Mas tem uma fase de setup que é mais demorado, no entanto o seu custo é baixo relativamente ao custo de desenho. Por exemplo, numa visão de uma rua, o view frustum culling é bom, mas p/ por exemplo a vista de uma cidade não. Tem uma complexidade no sentido de ter que testar as boxes p/o nr de boxes é relativamente reduzido relativamente ao nr de triângulos.

//APLICADO A VOLUMES ENVOLVENTES (BOUDING BOXES)

normalmente o view frustum é aplicado normalmente em software (CPU) exceto quando estamos a fazer tecelagem na própria placa

- O Occlusion culling é de longe o mais complicado → o mais complexo computacionalmente (em todo o seu setup)

pq envolve a relação espacial entre os objetos. Nós só sabemos q um objeto está ocludido q sabemos que está um objeto à sua frente, (com restrições) e isso não é fácil.

Uma maneira de fazer isto é desenhar os objetos sem cores, sem texturas, sem nada, para criar um mapa de profundidade p/ o nosso ecrã. Depois todos os que estão p/ lá não são desenhados. Assim, há o custo de desenhar aquele mapa, no entanto poupa-se no facto de depois desenhar apenas os pixéis visíveis.

↳ Implica o desenho ser ordenado, utilizando os space partioning no powerpoint

NOTA: Geralmente o custo de desenho de um pixel é maior que o de um vértice

⑨ Cf. 58 Os processos de partição espacial são em regra recursivos na construção da estrutura de dados. Indique 3 critérios possíveis p/ terminar a recursividade.

R: CRITÉRIOS PARA PARAR A SUBDIVISÃO RECURSIVA DA CENA

O CRITÉRIO MAIS USADO → a contagem de polígonos de uma célula atinge o limite. Se o nr de triângulos que está dentro de uma det. célula é pequeno não vale a pena continuar. Se fizermos dividir vamos criar ordens de desenho mais pequenas e portanto menos eficientes

→ a árvore fica d mt profundidade é muito difícil este critério ser atingido. Só atinge numa cena mais balanceada, e aí mais vale reavaliar a divisão

→ a célula é demarcado pequena basicamente quer dizer que a prob. de o view frustum a intersetar é muito pequena e portanto não vale a pena dividi-la.

(12) Tipos de volumes envolventes (Bounding Volumes) que podem ser usados numa partição hierárquica, comparando a sua eficiência em termos de cullins e complexidade algorítmica.

R: Cf. 39

- Os AABB (Axis aligned) e as Esferas são os mais fáceis de construir → os dois mais rápidos de construir mas serão mais facilmente falsos positivos
e mais fáceis de testar; São os que têm mais "espaço branco", ou seja, mais espaço não ocupado dentro da célula, dando mais facilmente um falso positivo.

- Os OBB (Object aligned) é preciso calcular o eixo principal, o que é relativamente fácil, mas o teste em si fica mais complicado.

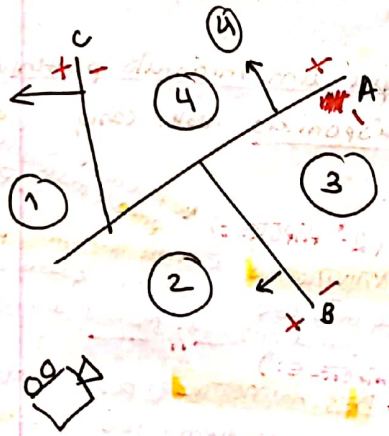
NOTA (GERAL): Tudo é um equilíbrio entre o custo do teste e o custo do desenho. !!!
Se o custo do teste começar a subir mais vale desenhar

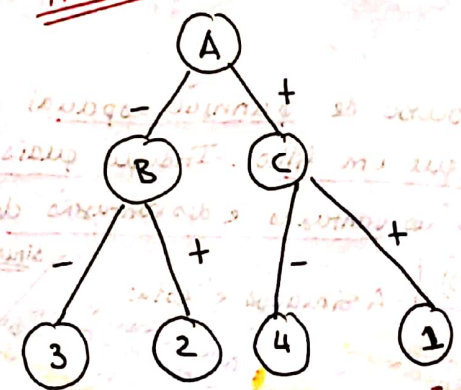
- O ConvexHull é o mais complicado pq implica estar a calcular o volume envolvente p/ uma determinada forma, o que é complicado e vai produzir imensos vértices; Acaba por não ser mt usado pq o custo de construção e teste é demasiado grande (p/ evitar os situações de falsos positivos).

- Os falsos positivos são a única coisa que interessa nisto dos Bounding Volumes, aqueles caixas que ficam nos cantos e nos gaps à volta do mesh. De resto quanto mais fácil for a caixa de desenhar, melhor.

(11) (RESPONDIDO NOUTRA SESSÃO DE DÚVIDAS)

R: ÁRVORE





ORDEM: 2º 1º 4º 3º

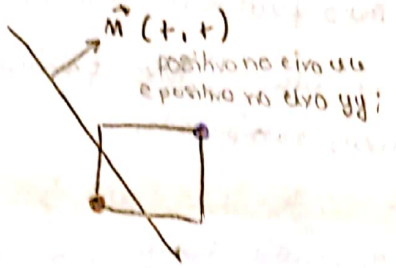
- como câmara ali, o (1) e o (4) nunca podem tapar o (3) e o (2).

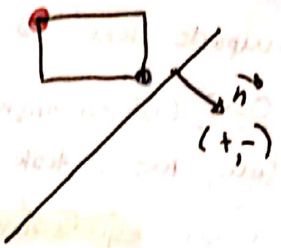
→ Primeiro desenho os do lado onde está a câmara o (2) primeiro, depois o (3)

Depois de vencida desenho o (1) porque está do lado positivo, onde está a câmara tb.

(13) Descreva detalhadamente o processo otimizado do teste de inclusão no VFC c/ paralelipedos alinhados c/ os eixos.

R: (Cf. 23) Se os paralelipedos tiverem alinhados c/ os eixos (os lados são todos perpendiculares a um eixo) podemos acelerar o processo olhando p/ os sinais que definem a normal.)

A) 

B) 

→ os sinais negativos e positivos dizem-nos qual o canto da caixa que vamos buscar. Se for positivo, vamos buscar o máximo, negativo vamos buscar o mínimo

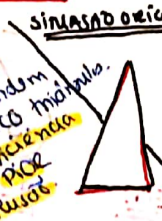
A) Como o ponto azul, que é o que as normais indicam, está do lado certo do plano, então a caixa está pelo menos parcialmente dentro.

O outro ponto, o vermelho, serve para testar se a caixa está totalmente dividida ou não (para que isso serve? se a caixa tiver filhos, mas estiver toda dentro do plano, não vale a pena estar a testar os filhos). Para testar esta parte, temos que ver o contrário dos sinais. Temos que ver o sinal com menor x e menor y.

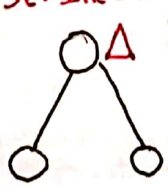
(Na situação B é a mesma coisa)

(10) Num processo de partição espacial é possível que um triângulo pertença a mais do que um filho. Indique quais as opções disponíveis nesse caso, apresentando as vantagens e desvantagens de cada uma.

R: (cf. PAG. 60). A situação é esta:

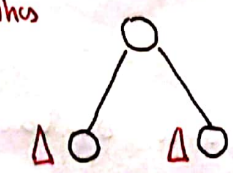


- VANTAGENS: Versão mais simples em termos de construção da árvore (não há duplicação nem corte...)
- DESVANTAGENS: Não compenso estar a gerar mais uma ordem de grandeza p/ ter pouco triângulo. Em termos de eficiência (FPS) é a PIOR solução.
- 1ª HIPÓTESE: Incluí-la na célula pai



Ficamos c/ triângulos não só nos filhos como nos ramos intermédios da árvore

- 2ª HIPÓTESE: Considero o triângulo associado a ambos os filhos

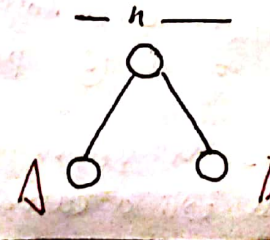


No fundo duplica o triângulo (o seu índice)

A melhor normalmente →

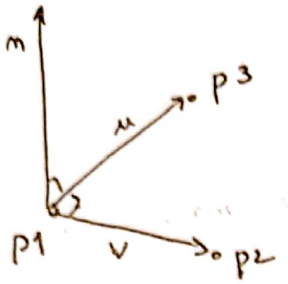
NOTA: Nem sempre pouco no nº de vértices normalmente quando não é binário.

- (2ª HIPÓTESE) NOTA: Em termos de desenhar os pixéis é indiferente relativamente à primeira hipótese.
- VANTAGENS: É mais eficiente que a 1ª opção pq não estou a criar aquela ordem a mais (apesar de processar mais alguns vértices)
- (3ª HIPÓTESE:)
- DESVANTAGENS: Apesar do nº de pixéis ser o mesmo, mas temos que processar mais triângulos e temos o trabalho extra de dividir. Não compensa. (Vantagens mínimas em certas ocasiões)
- 3ª HIPÓTESE: Dividi-lo de forma a que cada parte (original):



② Processo matemático pl obter a eq. normalizada do plano que contém os pontos p_1, p_2, p_3.

R:



ef. 17

$Ax + By + Cz = \emptyset$

$\vec{v} = p_2 - p_1$

$\vec{u} = p_3 - p_1$

$\vec{m} = (m_x, m_y, m_z) = \vec{v} \times \vec{u}$

$\vec{M} = \frac{\vec{m}}{|\vec{m}|}$

$A = m_x \quad B = m_y \quad C = n_z$ } normal ao plano

como o ponto p_1 está no plano

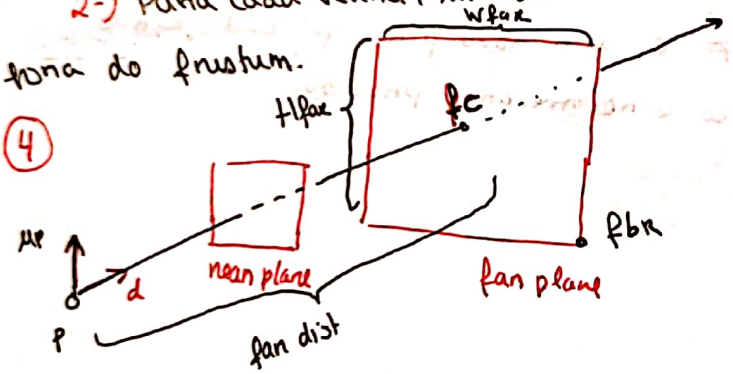
$Ap_{1x} + Bp_{1y} + Cp_{1z} + D = \emptyset$

$D = -Ap_x - Bp_y - Cp_z = -\vec{n} \cdot p_1$

③ O ALGORITMO DE view frustum culling consiste em:

1-) calculan as equações do plano (1 por frame)

2-) Para cada vértice / triângulo / objeto / volume testam se está dentro ou fora do frustum.



④ → considerando os vetores d e up, o ponto p, e as distâncias farDist, Wfar e Hfar, descreva o processo mat pl obter o ponto fbr.

R:

ef. 14

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

$Wnear = Hnear \times ratio$

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

$Wfar = Hfar \times ratio$

ef. 15

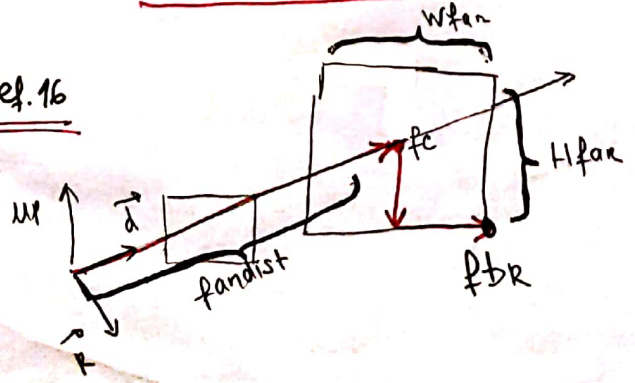
- A plane can be defined by a normal and a point.

. farplane can be defined by

normal: $-\vec{d}$

point: $fc = p + \vec{d} * fardist$

ef. 16



$fc = p + \vec{d} \times farDist$

$fbr = fc - \left(\vec{up} \times \frac{Hfar}{2}\right) + \left(\vec{r} \times \frac{Wfar}{2}\right)$

⑤ gluPhospective (fov, ratio, nearDist, farDist);
gluLookAt (px, py, pz, lx, ly, lz, ux, uy, uz);

Processo matemático p/ obter os dados da proj. anterior:
- vetores d, up e right, e distâncias Wfar e Hfar.

R: cf. 14

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

$$Wfar = Hfar \times ratio$$

$$\vec{up} = (ux, uy, uz)$$

$$\vec{d} = l - p = (lx, ly, lz) - (px, py, pz)$$

$$\vec{right} = \vec{d} \times \vec{up}$$

⑧ Descreva o processo de partição espacial baseado em k-D trees

R: colocar planos ⊥s aos eixos de maneira a agrupar os Δs.
Para tal o processo é:

cf. 49
→ Escolhe um eixo e coloca um plano ⊥ dividindo assim em 2 regiões.
→ selecionar um eixo ≠ e um novo plano p/ cada região
→ Iterar em todos os eixos e recomeçar o processo.