Composição

- Depois da renderização, texturização e aplicação de nevoeiro, os dados ainda não estão transformados em *pixels*, são fragmentos.
  - Cada fragmento tem coordenadas correspondentes a um *pixel*, assim como valores de cor e profundidade.
- Só depois do fragmento passar por certas operações é que se pode tornar num *pixel*.

Composição

- Se um fragmento é eliminado no decorrer de uma determinada operação, nenhuma das operações seguintes terão lugar.
- As operações feitas nesta etapa (ocorrendo na ordem indicada) são:
  - Scissor test (1º); – Alpha test (2º);
  - Stencil test (3º); – Depth test (4º);
  - Blending (5º); – Dithering (6º);
  - Logical operation (7º).

Composição

- Apesar da variedade das operações descritas, pela sua importância no processo de criação de imagens realista, apenas serão aprofundadas os tópicos de:
  - **Transparência**
    - que engloba as operações de *Alpha test* e *Blending*;
  - **Visibilidade**
    - que engloba as operações de *Scissor test* e *Depth test*.

Composição

– Buffers –

- Dentro da etapa de composição, um aspecto importante é o uso de diferentes *buffers* (espaços de memória) para guardar informações sobre a imagem final da cena 3D, vista pela câmara.
- Cada um desses *buffers* tem uma função específica e pode ser utilizado para criar diferentes efeitos visuais na imagem final.

Composição

– Buffers –

- Os *buffers* são:
  - Discretos;
  - Têm uma resolução espacial, medida em *mxn pixels*;
  - Têm uma resolução de profundidade, medida em *bits/pixel*, que é dada pelo número de planos de bits, onde cada plano tem *mxn pixels*;

Composição

– Buffers –

- Cada *pixel* é constituído por todos os *k* elementos de cada plano, na mesma posição espacial, podendo ser de um *bit*, um *byte*, um inteiro ou um real.

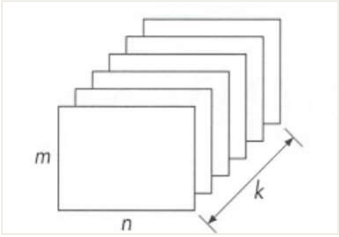


Figura 1. Representação de um *buffer*.

Composição

– Buffers –

- O *buffer* que todos os API gráficos usam é o da cor (pelo menos um).
  - É conhecido como ***frame buffer*** e contém a imagem final da cena 3D vista pela câmara;
  - Para implementações de aplicações que suportem uma visualização estereoscópica, os API gráficos usam dois *buffers* de cor para armazenar cada uma das imagens do par estéreo (os *buffers* esquerdo e direito).

Composição

– Buffers –

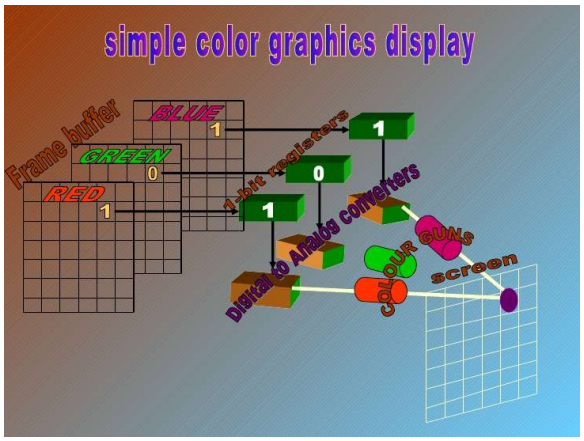


Figura 2. Funcionamento do *frame buffer* para o modelo de cor *RGB*.

Composição

– Buffers –

- Para suavizar a visualização de animações, os sistemas gráficos têm o *buffer* da frente e o *buffer* de trás (ou seja, são *double-buffered*).
  - Qualquer implementação gráfica tem de usar pelo menos o *buffer* esquerdo frontal.
- Opcionalmente, ainda, podem ser suportados *buffers* de cor auxiliares não visualizáveis, sem qualquer utilização particular.

Composição

– Buffers –

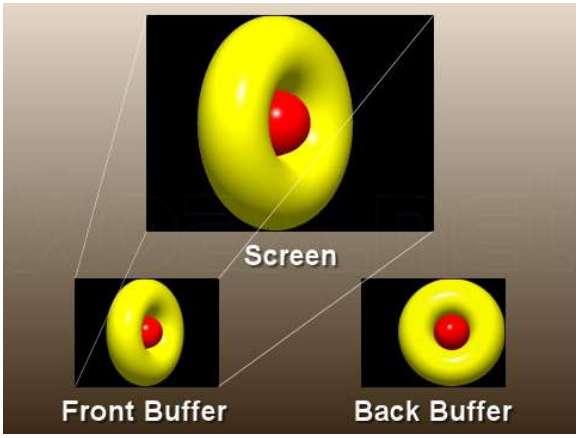


Figura 3. Funcionamento do *double buffer*.

Composição

– Buffers –

- Por outro lado, os API gráficos usam um outro *buffer* muito importante, o *Z-buffer* ou o *buffer* de profundidade.
  - Armazena um valor de profundidade para cada *pixel*.
  - A profundidade é medida usualmente em termos de distância dos polígonos dos objetos ao centro de projeção.
  - Este *buffer* é utilizado para remoção de superfícies ocultas (visibilidade).

Composição

– Buffers –

- Dos *buffers* mais conhecidos e usados na área da CG, resta falar do *buffer* de *stencil*.
- Este *buffer* de stencil é utilizado para restringir o desenho a certas zonas.
- Por isso é usado principalmente em operações de mascaramento.

Composição

– Buffers –

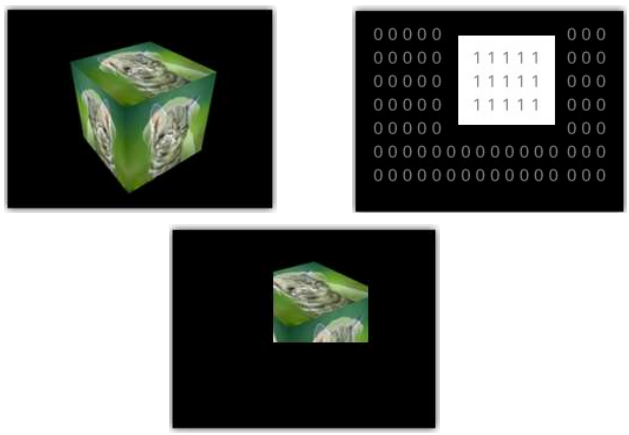


Figura 4. Funcionamento do *buffer* de *stencil*.

Composição

– Transparências –

- A luz que incide numa superfície pode ser absorvida, refletida e transmitida.
  - O comportamento da luz depende das características do material que reveste essa superfície.
- Uma superfície é opaca quando toda a luz que é transmitida é absorvida por ela.

Composição

– Transparências –

- Uma superfície pode ter diferentes níveis de **transparência** ou de **translucidez** quando permite que parte da luz que é transmitida a atravesse.

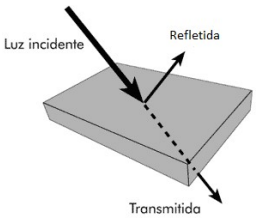


Figura 5. Transmissão de luz.

Composição

– Transparências –

- As superfícies através das quais se vê claramente (sem distorções) são conhecidas como **transparentes**.
- As superfícies através das quais se vê de forma distorcida são conhecidas com **translúcidas**.
  - Essa distorção é devida à luz que atravessa a superfície ser espalhada em todas as direções.

Composição

– Transparências –

- A quantidade de luz transmitida depende do grau de transparência da superfície e da existência de fontes de luz ou objetos iluminados atrás da superfície.

Figura 6. Grau de transparência.

Composição

– Transparências –

- Existem dois modos de aplicar os efeitos de transparência:
  - Sem refração
    - Os raios de luz não sofrem desvios ao atravessar a superfície.
      - Apesar de não ser tão realístico, este modo produz resultados razoáveis para objetos transparentes finos.
  - Com refração
    - A transparência é baseada nas leis físicas que regem a refração de luz (lei de Snell).

Composição

– Transparências –

- Na transparência com refração, o caminho que a luz refratada segue pode ser distinto do caminho seguido pela luz incidente (quando há diferenças de velocidade no material atravessado).
  - O IOR (*index of refraction ou* índice de refração) diz quanto a luz se irá curvar quando atravessar um material.

Composição

– Transparências –

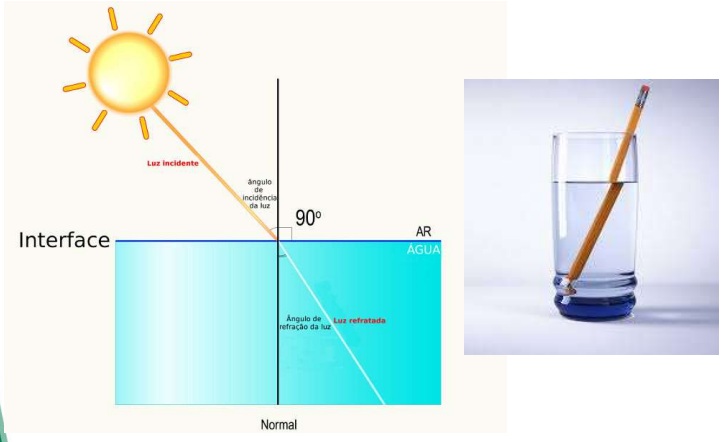


Figura 7. Efeito de refração da luz.

Composição

– Transparências –

- Na computação gráfica utilizam-se valores de IOR médios para os materiais que compõem o cenário.
  - Uma lista destes indices pode ser vista em https://pixelandpoly.com/ior.html), de onde se destacam:
    - o ar, com um IOR de 1.0;
    - a água, com um IOR de 1.325;
    - a cerveja, com um IOR de 1.345;
    - o vidro, com um IOR de 1.5.

Composição

– Transparências –

- O *blender* permite criar materiais transparentes sem refração:

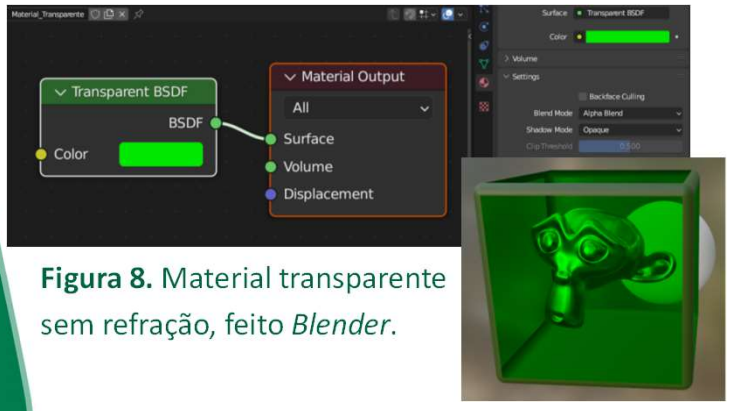


Figura 8. Material transparente sem refração, feito *Blender*.

Composição

– Transparências –

- E, também, permite criar materiais transparentes com refração:

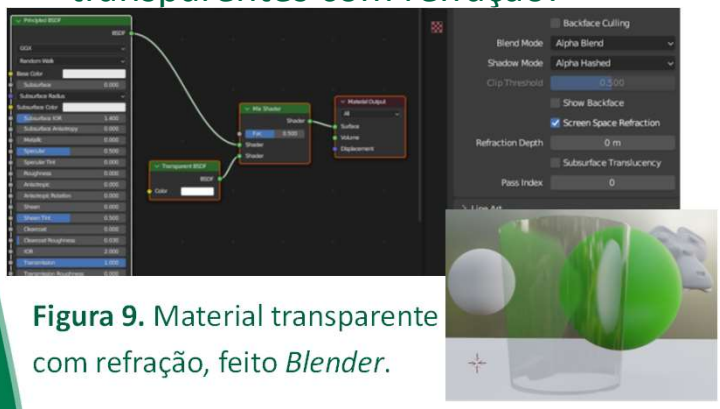


Figura 9. Material transparente com refração, feito *Blender*.

Composição

– Transparências –

- Um dos parâmetros usados para lidar com a transparência é o **canal *ALPHA***, inventado por Catmull & Smith (1971-1972).
- O canal *ALPHA* permite misturar as cores dos *pixels* na imagem final, tornando possível criar cenas com elementos transparentes e translúcidos.

Composição

– Transparências –

- A obtenção da transparência com base no canal ALPHA pode ser explicado da seguinte forma:

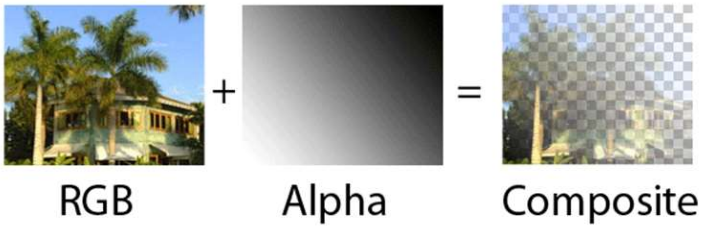


Figura 10. Explicação do canal *ALPHA*.

Composição

– Transparências –

- Tipicamente, os valores de ALPHA são armazenados com as componentes de cor vermelha (R), verde (G) e azul (B), no modelo de cores RGBA.

Figura 11. Exemplos de diferentes valores de RGBA.

Composição

– Transparências –

- Os valores de ALPHA, que descrevem a percentagem de opacidade dos objeto, estão no intervalo [0.0, 1.0].

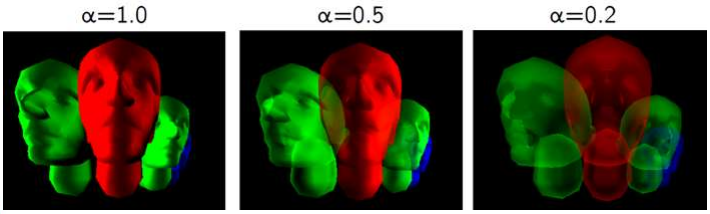


Figura 12. Opacidade/transparência baseada na alteração dos valores do canal *ALPHA*.

Composição

– Transparências –

- Os algoritmos mais básicos e antigos de mistura de cores não lidam com a refração nem com a espessura dos objetos.
  - Uma das maneiras mais conhecidas de se obter transparência é misturando as cores dos *pixels* na imagem final, através da fórmula:

$$Cor_{Final} = \alpha \, . \, Cor_{ObjectoTransparente} + (1 - \alpha) \, . \, Cor_{fundo}$$

Composição

– Transparências –

- A ordem por que é feita a mistura de cores importa.

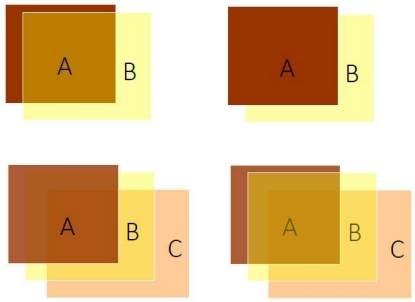
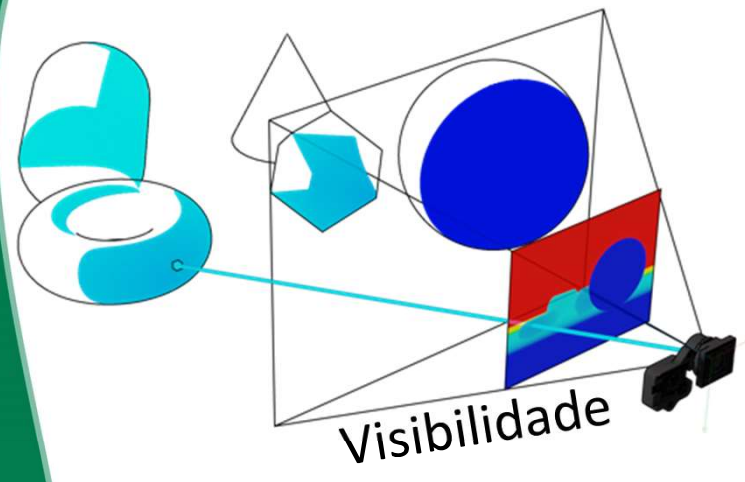


Figura 13. Ordem de mistura de cores na obtenção de transparências.



Composição

– Visibilidade –

- A imagem final de uma cena 3D, normalmente (dependendo do ponto de vista), não mostra ao mesmo tempo todos os polígonos de todos os objetos.
  - Também não se pretende que o que não é visíveis apareça na imagem.
- Assim, na computação gráfica, o **problema da visibilidade** é um tópico relevante.

Composição

– Visibilidade –

- O problema da visibilidade refere-se ao desafio de determinar quais os polígonos dos objetos que são visíveis a partir da posição e orientação do observador.
  - Quando se renderiza uma cena 3D, é preciso decidir o que está à frente (e deve ser exibido) e o que está atrás (deve ser oculto), tendo em conta o ponto de vista da câmara.

Composição

– Visibilidade –

- Em geral, resolve-se este problema encontrando-se os polígonos que serão invisíveis por estarem:
  - fora do campo de visão;
  - tapados por outros polígonos opacos do próprio objeto, tendo em conta o ponto de vista do observador.
  - tapados por outros polígonos opacos de outros objetos, tendo em conta o ponto de vista do observador;
  - tapados por sombras.

Composição

– Visibilidade –

- Assim, o problema da visibilidade inclui as seguintes operações:
  - Descartar o que não pode ser visto (***culling***), ligada ao *Depth test*;
  - Recortar os objetos para manter apenas o que pode ser visto (***clipping***), ligada ao *Scissor test*;
  - Desenhar apenas as partes visíveis dos objetos (uso dos algoritmos ***hidden line*** e ***hidden surface***);
  - Visibilidade a partir de fontes de luz.

Composição

– Visibilidade –

- Os algoritmos de visibilidade podem ser classificados em técnicas:
  - baseadas no espaço da imagem;
  - baseadas no espaço dos objetos;
  - Mistas.
- Cada uma destas técnicas tem as suas vantagens e desvantagens, em termos de qualidade de imagem, velocidade de renderização e complexidade de implementação.

Composição

– Visibilidade –

- Nas técnicas baseadas no espaço da imagem:
  - As entradas são vetoriais e as saídas são matriciais;
  - Está-se dependente da resolução da imagem;
  - A visibilidade é calculada apenas em *pixels*;
  - Aproveita-se a aceleração por *hardware*.

Composição

– Visibilidade –

- Nas técnicas baseadas no espaço dos objetos:
  - As entradas e as saídas são dados geométricos;
  - Não se está dependente da resolução da imagem;
  - Há menos vulnerabilidade ao *aliasing*;
  - A rasterização ocorre depois.

Composição

– Visibilidade –

- Os algoritmos mais conhecidos para lidar com o problema da visibilidade são os seguintes:
  - *Back-face culling* (baseado no espaço dos objetos);
  - Algoritmo do pintor (baseado no espaço dos objetos);
  - Algoritmo *Z-buffering* (mais comum para lidar com este problema e baseado no espaço da imagem.

Composição

– Visibilidade –

- Dado que o *Back-face culling* assume que todos os polígonos são definidos com os seus vetores normais a apontarem "para fora", determina os polígonos traseiros (que estão voltadas para o lado oposto do observador) e elimina-os do processo de desenho.
- Este algoritmo não pode ser aplicado a poliedros concavos.

Composição

– Visibilidade –

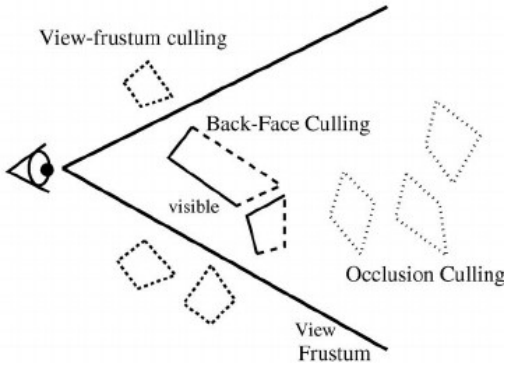


Figura 14. Funcionamento do *Back-face culling*.

Composição

– Visibilidade –

- O algoritmo do pintor, também, conhecido como algoritmo de prioridade em Z (*depth priority*), funciona da seguinte maneira:
  - Ordena todos os polígonos de acordo com a sua distância ao observador;
    - O problema da ordenação não é trivial.
  - Depois, "pinta" os polígonos por ordem decrescente (mais distantes da câmara primeiro).

Composição

– Visibilidade –

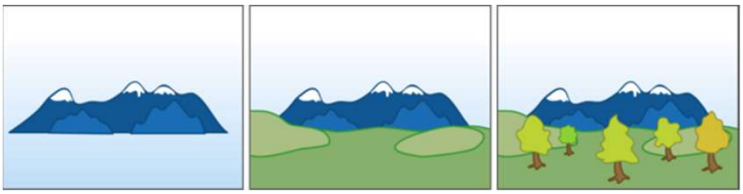


Figura 15. Funcionamento do algoritmo do pintor.

Composição

– Visibilidade –

- Os problemas de ordenação que o algoritmo do pintor pode ter resolvem-se com a segmentação dos polígonos sobrepostos (ou que se intersetam).

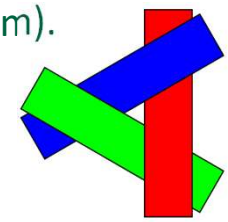


Figura 16. Problema de ordenação, quando há sobreposição entre polígonos.

Composição

– Visibilidade –

- O *Z-buffering* é um algoritmo baseado em hardware, que usa o *Z-buffer* e o *frame buffer*, da seguinte maneira:
  - À medida que a cena é renderizada, o algoritmo verifica se o novo *pixel* é mais próximo do que o *pixel* já renderizado e, se for o caso, atualiza o *Z-buffer* e atualiza o *pixel* correspondente na imagem (guardada no *frame buffer*).

Composição

– Visibilidade –

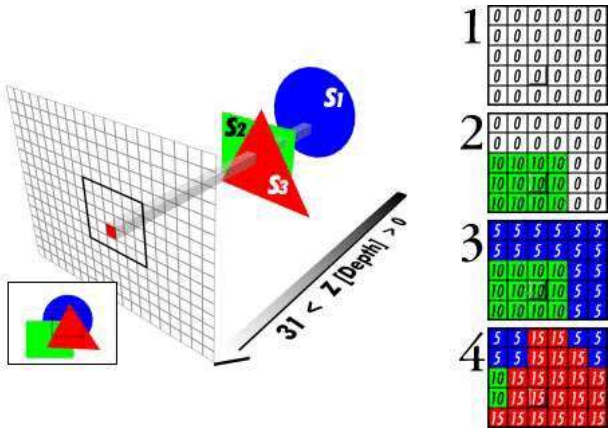


Figura 17. Funcionamento do *Z-buffering*.

Composição

– Visibilidade –

- Vantagens do algoritmo *Z-Buffering*:
  - Simples de implementar, pois não necessita de cálculos de intersecções;
  - Objetos opacos podem ser desenhados em qualquer ordem;
  - Espaço de memória independente do número de polígonos a processar;
  - É aplicável a qualquer forma;
  - É realizado por *hardware*.

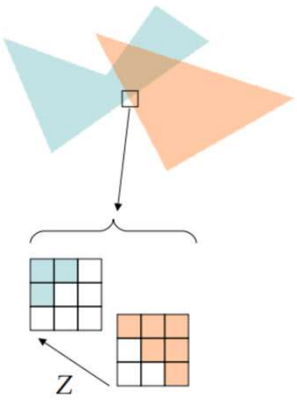
Composição

– Visibilidade –

- Desvantagens do *Z-Buffering*:
  - Lento, se o número de polígonos a processar for grande;
  - Perde tempo com objetos ocultos;
  - Lida mal com as transparências e com técnicas de *anti-aliasing*.
    - Para ultrapassar este problema, pode usar-se o:
      - algoritmo do pintor;
      - *A-Buffer (*extensão do *Z-Buffer*).

Composição

– Visibilidade –



No A-buffer, para cada *pixel* tem-se uma lista ordenada por profundidades *(Z)*, com cada nó a conter:

- máscara de *sub-pixels* ocupados;
- cor ou ponteiro do polígono;
- valor de profundidade.

Figura 18. Funcionamento do *A-Buffer*.

Composição

– Visibilidade –

- As árvores BSP (Binary Space Partitioning),
  - têm muitas aplicações em computação gráfica e, também, pode ser usada na resolução do problema da visibilidade;
  - são estrutura de dados utilizadas para organizar recursivamente os polígonos dos objetos que se encontram dentro de um espaço.
    - Cada nó pai da árvore gera sempre dois nós filhos.

Composição

– Visibilidade –

- Em termos de construção da árvore, os passos a seguir são os seguintes:
  1. Escolhe-se, arbitrariamente, um dos polígonos presente no espaço;
  2. Dividir esse espaço em dois, espaços frontal e traseiro, em relação ao seu vetor normal (pode requerer o uso de recorte de polígonos, com base na sua interseção com o plano de corte);
  3. Voltar a 1, até se ter apenas um (ou parte de um) polígono por nó.

Composição

– Visibilidade –

Figura 19. Construção da árvore BSP (ponto de partida).

Composição

– Visibilidade –

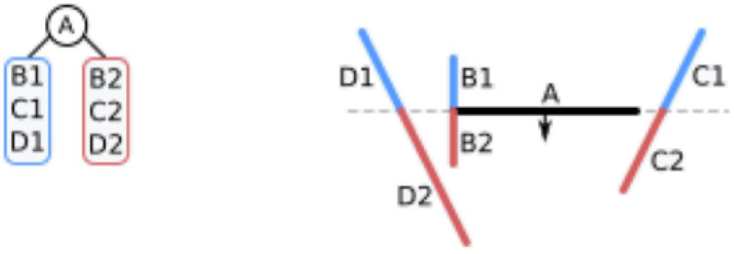


Figura 20. Construção da árvore BSP (primeira interação).

Composição

– Visibilidade –

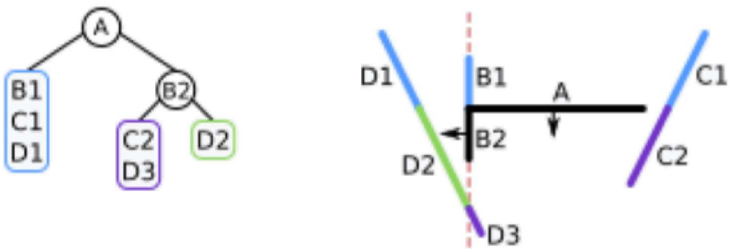


Figura 21. Construção da árvore BSP (segunda interação).

Composição

– Visibilidade –

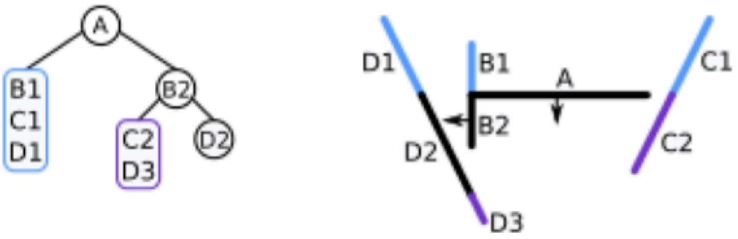


Figura 22. Construção da árvore BSP (terceira interação).

Composição

– Visibilidade –

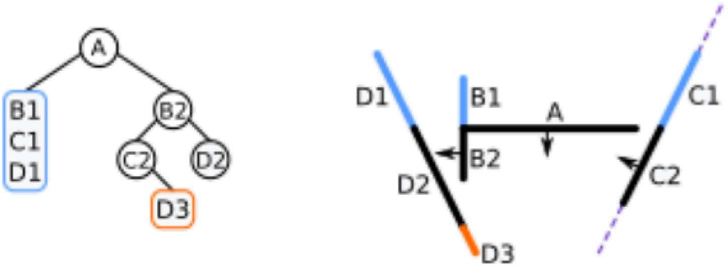


Figura 23. Construção da árvore BSP (quarta interação).

Composição

– Visibilidade –

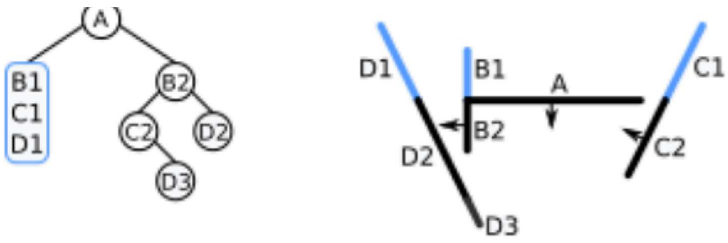


Figura 24. Construção da árvore BSP (quinta interação).

Composição

– Visibilidade –

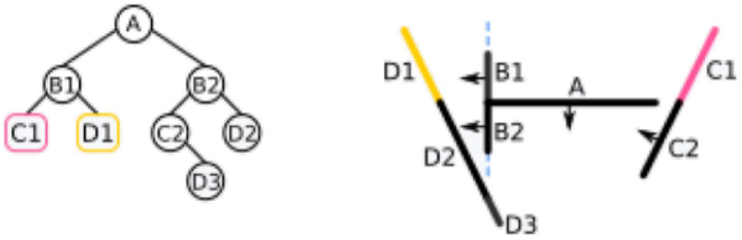


Figura 25. Construção da árvore BSP (sexta interação).

Composição

– Visibilidade –

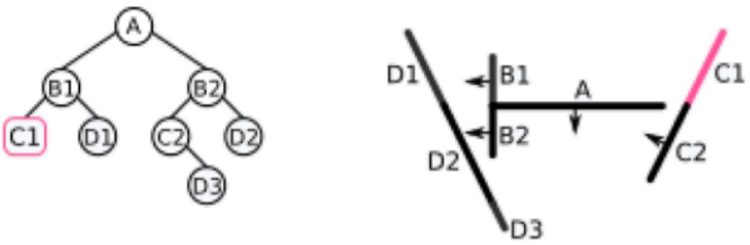


Figura 26. Construção da árvore BSP (sétima interação).

Composição

– Visibilidade –

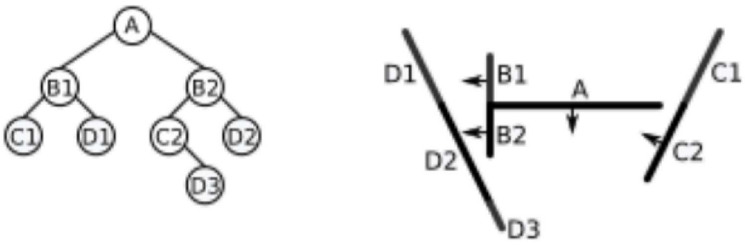


Figura 27. Construção da árvore BSP (ponto de chegada).

Composição

– Visibilidade –

- Com a árvore BSP pode-se obter a ordem de desenho dos polígonos, baseada na sua profundidade em relação ao observador,
  - A partir da frente para trás;
  - A partir de trás para frente (usando, por exemplo, o algoritmo do pintor).
- Pode, ainda, ter que se fazer o descarte (*culling*) dos polígonos que estão fora do *frustum* de visão.

Composição

– Visibilidade –

- O desenho dos polígonos da árvore faz-se da seguinte forma:
  - Se o nó actual for um nó folha, renderizar os polígonos do nó actual;
  - Se local de visualização estiver em frente do nó actual:
    - Renderizar a sub-árvore contendo polígonos atrás do nó actual;
    - Renderizar os polígonos do nó actual;
    - Renderizar a sub-árvore contendo polígonos em frente do nó actual;
  - …

Composição

– Visibilidade –

- O desenho dos polígonos da árvore faz-se da seguinte forma:
  - …
  - Se local de visualização estiver atrás do nó actual:
    - Renderizar a sub-árvore contendo polígonos em frente do nó actual;
    - Renderizar os polígonos do nó actual;
    - Renderizar a sub-árvore contendo polígonos atrás do nó actual
  - …

Composição

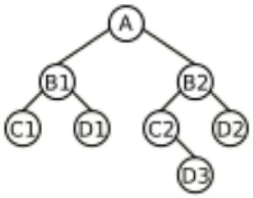
– Visibilidade –

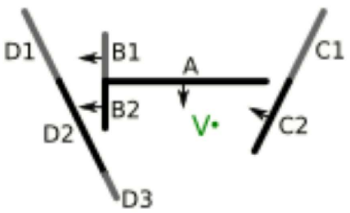
- O desenho dos polígonos da árvore faz-se da seguinte forma:
  - …
  - Se o local de visualização estiver exatamente no plano associado com o nó actual:
    - Renderizar a sub-árvore contendo polígonos em frente do nó actual;
    - Renderizar a sub-árvore contendo polígonos atrás do nó actual.

Composição

– Visibilidade –

- Tendo em conta a árvore BSP e o local de visualização:





a ordem de desenho (de trás para a frente) será ***D1, B1, C1, A, D2, B2, D3, C2***.