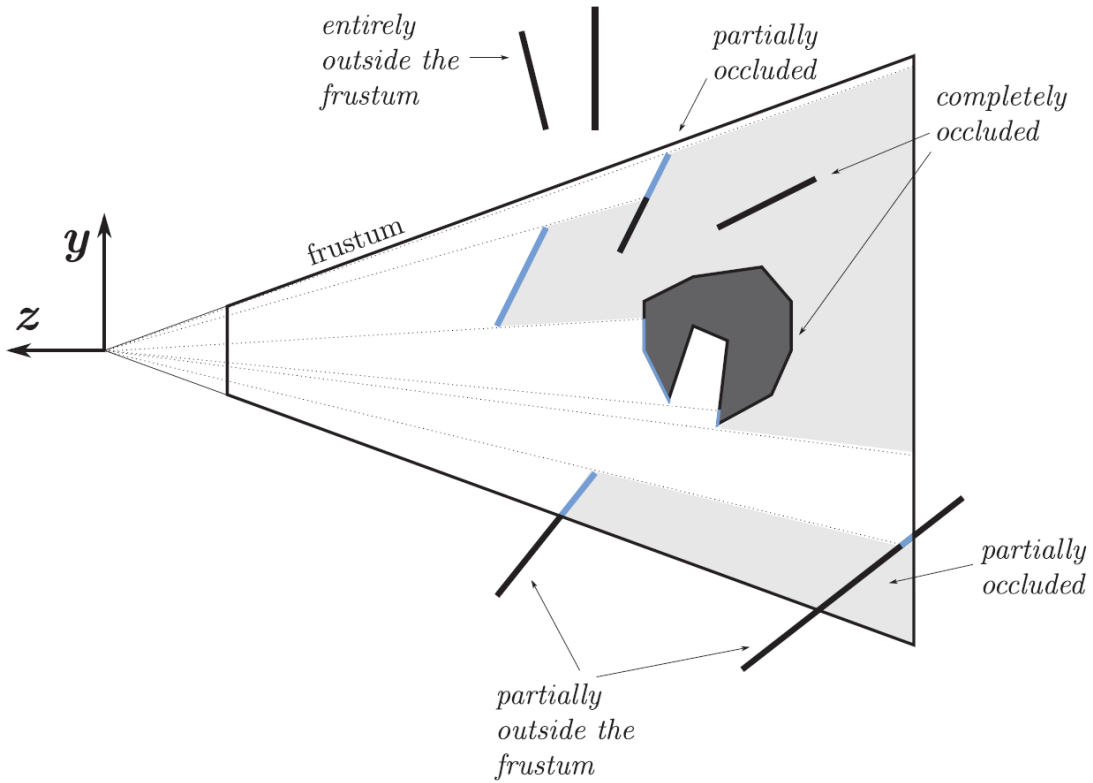
Geometria obliczeniowa

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Binarne podziały przestrzeni

W symulatorach lotu ważnym zadaniem jest wizualizacja sceny 3D. Aby renderować scenę dla każdego piksela musimy określić obiekt, który jest widoczny w tym pikselu. Nazywamy to usuwaniem niewidocznych powierzchni.



Dodatkowo musimy wykonać obliczenia oświetlenia i cieniowania, które są kosztowne.

W symulatorach lotu i nie tylko (np. grach) renderowanie musi być wykonywane w czasie rzeczywistym. Dlatego stosuje się szybkie i proste techniki cieniowania, a usuwanie niewidocznych powierzchni staje się ważnym czynnikiem.

Dodatkowo musimy wykonać obliczenia oświetlenia i cieniowania, które są kosztowne.

W symulatorach lotu i nie tylko (np. grach) renderowanie musi być wykonywane w czasie rzeczywistym. Dlatego stosuje się szybkie i proste techniki cieniowania, a usuwanie niewidocznych powierzchni staje się ważnym czynnikiem.

Popularną i prostą techniką usuwania niewidocznych powierzchni jest algorytm bufora Z.

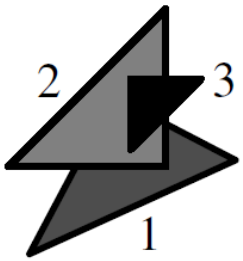
Na początku pracy algorytmu bufor z jest wypełniany maksymalną wartością głębi. Jednocześnie wszystkie piksele obrazu przyjmują barwę tła.

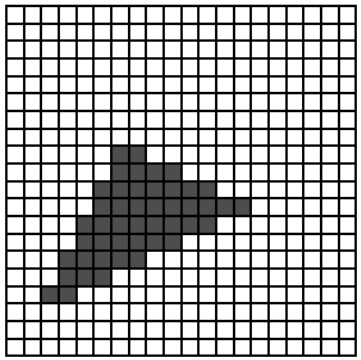
Następnie rysowane są wielokąty (w dowolnej kolejności) – to znaczy wypełniane są ich rzuty odpowiednią barwą.

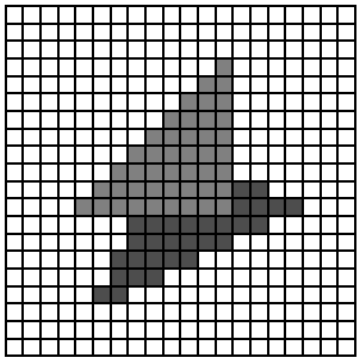
Podczas wypełniania zwykła procedura wypełniająca jest poszerzona o sprawdzenie głębi odpowiadającej danemu pikselowi. Piksel jest wypełniony tylko wtedy, kiedy jego głębia jest mniejsza niż wartość zapisana w buforze.

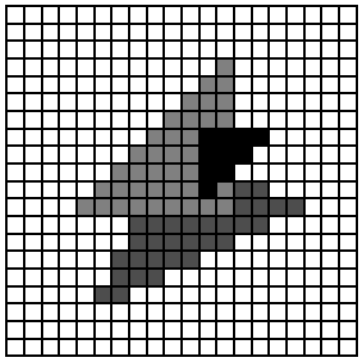
Algorytm bufora Z wymaga dodatkowej pamięci, która rośnie wraz ze wzrostem rozdzielczości renderowanej sceny. Ponadto każdy piksel pokryty przez obiekt wymaga dodatkowego testu dla głębi.

Algorytm malarza unika tych dodatkowych kosztów przez wstępne posortowanie obiektów względem ich odległości od punktu obserwacyjnego. Następnie obiekty rysowane są w porządku od najdalszego do najbliższego (tzw. porządek głębi).







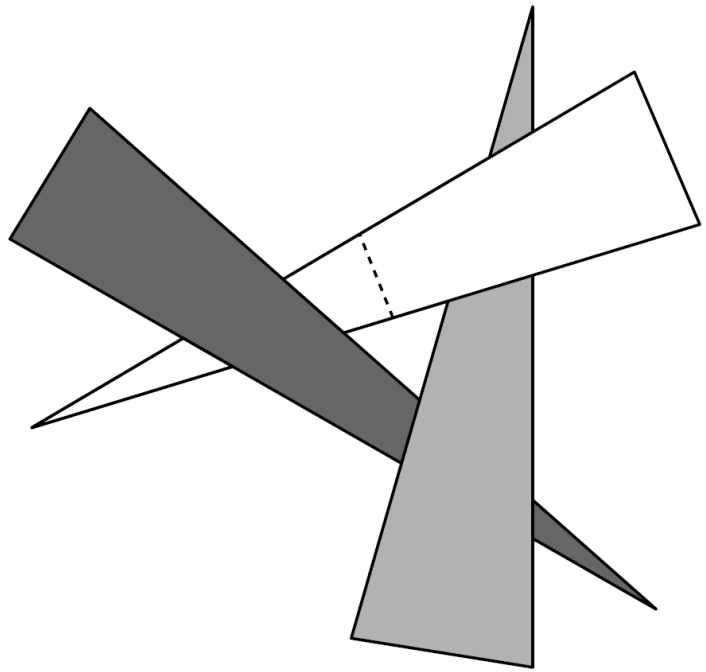


Numery oznaczają kolejność rysowania.

Cały proces przypomina sposób pracy malarza, gdy kładzie warstwy farby na powierzchni pozostałych warstw.

Skuteczność algorytmu malarza zależy od szybkości sortowania obiektów. Musimy być to w stanie zrobić szybko.

Porządek głębi nie zawsze może istnieć, np. w przypadku cyklicznego nałożenia żadne uporządkowanie nie da poprawnego obrazu.



W przypadkach cyklicznego nałożenia możemy podzielić pewne obiekty i następnie posortować względem głębi.

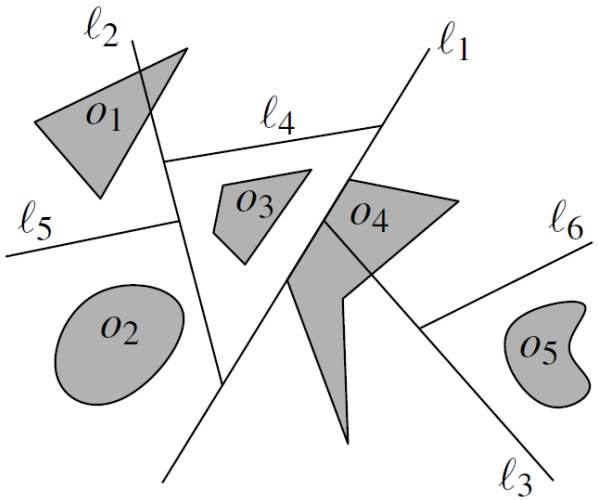
Obliczenie, które obiekty podzielić, jak je podzielić i w końcu posortować je jest kosztownym procesem.

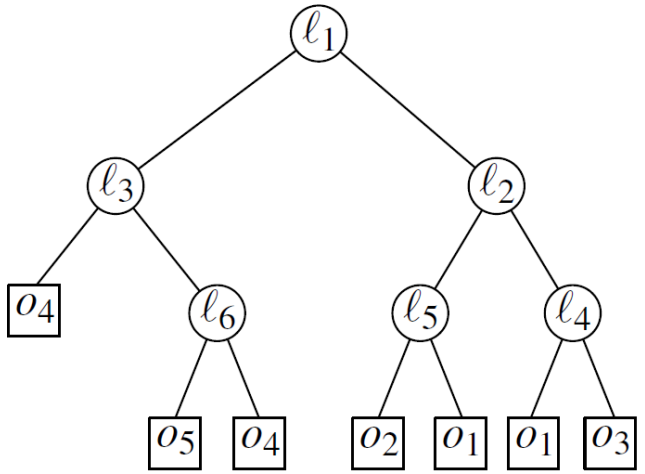
Ponadto porządek głębi zależy od położenia punktu obserwacyjnego, więc po zmianie położenia musimy ponownie obliczyć porządek.

Aby skorzystać z algorytmu malarza w środowisku czasu rzeczywistego musimy użyć dodatkowej struktury danych, która zapewni nam szybkie znalezienie porządku głębi dla dowolnego punktu obserwacyjnego.

Drzewa BSP

Strukturą danych nadającą się do naszych celów jest drzewo binarnego podziału przestrzeni (ang. binary space partitioning – BSP).





Dla danej hiperpłaszczyzny:

$$h : a_1 x_1 + a_2 x_2 + \ldots + a_d x_d + a_{d+1} = 0$$

niech h^+ oznacza otwartą dodatnią półprzestrzeń ograniczoną przez h, a h^- będzie otwartą ujemną półprzestrzenią, tzn.

$$h^+ = \{(x_1, x_2, \ldots, x_d) : a_1 x_1 + a_2 x_2 + \ldots + a_d x_d + a_{d+1} > 0\},$$
$$h^- = \{(x_1, x_2, \ldots, x_d) : a_1 x_1 + a_2 x_2 + \ldots + a_d x_d + a_{d+1} < 0\}.$$

Drzewo BSP dla zbioru S obiektów w d-wymiarowej przestrzeni jest to drzewo binarne $\mathcal{T}$ o własnościach:

- Jeśli $|S| \leqslant 1$, to $\mathcal{T}$ jest liściem. Fragment obiektu (o ile istnieje) jest pamiętany dokładnie w liściu. Jeśli liść jest oznaczony v, to $S(v)$ niech oznacza zbiór pamiętany w liściu.
- Jeśli $|S| > 1$, to korzeń v drzewa $\mathcal{T}$ pamięta hiperpłaszczyznę h_v wraz ze zbiorem $S(v)$ obiektów, które są całkowicie zawarte w h_v. Lewym dzieckiem v jest korzeń drzewa BSP $\mathcal{T}^-$ dla zbioru $S^- = \{h_v^- \cap s : s \in S\}$, a prawym dzieckiem v jest korzeń drzewa BSP $\mathcal{T}^+$ dla zbioru $S^+ = \{h_v^+ \cap s : s \in S\}$.

Rozmiar drzewa BSP jest całkowitym rozmiarem zbiorów $S(v)$ po wszystkich węzłach v drzewa (tzn. całkowita liczba stworzonych fragmentów obiektów).

Jeśli BSP nie zawiera bezużytecznych prostych rozdzielających (proste dzielące puste podprzestrzenie), to liczba węzłów drzewa jest co najwyżej liniowa względem rozmiaru drzewa BSP.

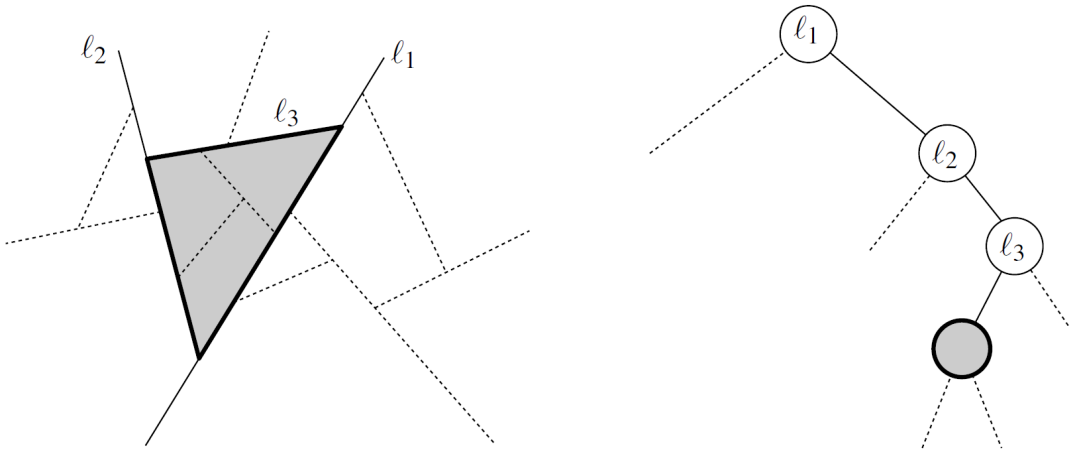
Rozmiar drzewa BSP jest całkowitym rozmiarem zbiorów $S(v)$ po wszystkich węzłach v drzewa (tzn. całkowita liczba stworzonych fragmentów obiektów).

Jeśli BSP nie zawiera bezużytecznych prostych rozdzielających (proste dzielące puste podprzestrzenie), to liczba węzłów drzewa jest co najwyżej liniowa względem rozmiaru drzewa BSP.

Liście w drzewie BSP reprezentują ściany w podziale wywołanym przez BSP.

Ogólniej, możemy utożsamić wypukły obszar z każdym węzłem v w drzewie BSP.

Obszar jest przecięciem półprzestrzeni $h_\mu^\diamond$, gdzie μ jest przodkiem v, a $\diamond = -$, gdy v jest w lewym poddrzewie μ lub $\diamond = +$, gdy jest w prawym poddrzewie. Obszar odpowiadający korzeniowi $\mathcal{T}$ jest pełną przestrzenią.

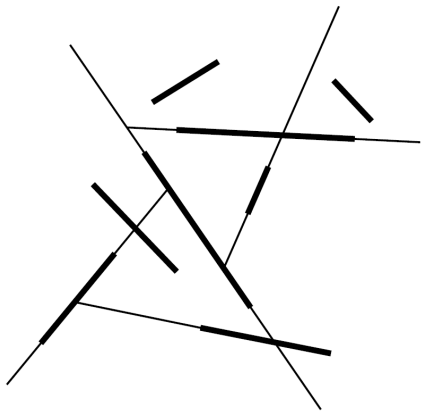


Szary węzeł odpowiada szaremu obszarowi $\ell_1^+ \cap \ell_2^+ \cap \ell_3^-$.

Hiperpłaszczyzny rozdzielające mogą być dowolne.

W celach obliczeniowych stosuje się ograniczenie zbioru dopuszczalnych hiperpłaszczyzn rozdzielających.

Powiedzmy, że chcemy wyznaczyć drzewo BSP dla zbioru odcinków na płaszczyźnie. Naturalnymi kandydatami na proste rozdzielające są przedłużenia odcinków. Drzewo BSP używające takich prostych nazywa się autopodziałem.

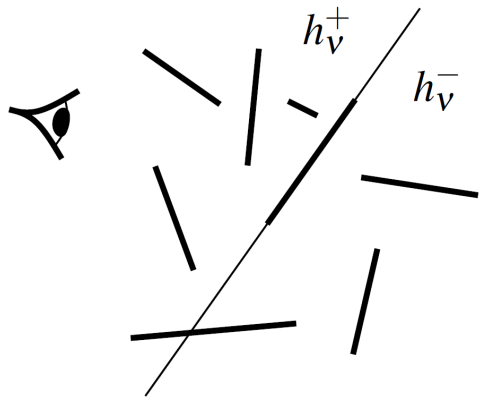


Dla zbioru wielokątów planarnych w przestrzeni 3D autopodziałem jest drzewo BSP, które jako płaszczyzn rozdzielających używa płaszczyzn przechodzących przez dane wielokąty.

Autopodziały nie zawsze tworzą drzewa BSP o minimalnym rozmiarze, ale są one rozsądnie małe.

Algorytm malarza i drzewa BSP

Załóżmy, że mamy drzewo BSP $\mathcal{T}$ dla obiektów sceny 3D. Niech p_{view} będzie punktem obserwacyjnym oraz załóżmy, że leży on powyżej płaszczyzny rozdzielającej z korzenia $\mathcal{T}$.

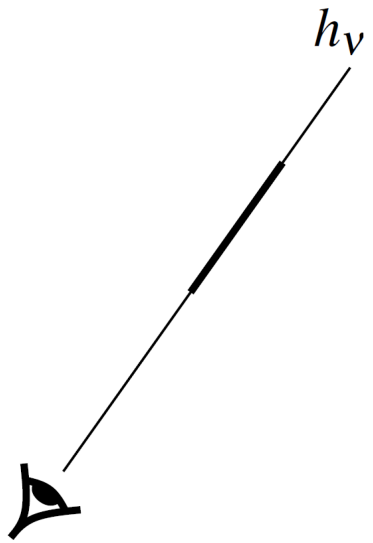


Żaden obiekt z h_v^- nie może zasłaniać obiektów z h_v^+. Zatem najpierw renderujemy obiekty z $\mathcal{T}^-$ a potem z $\mathcal{T}^+$. Porządek obiektów (fragmentów) w $\mathcal{T}^+$ i $\mathcal{T}^-$ otrzymujemy rekurencyjnie w ten sam sposób.

Algorithm PAINTERSALGORITHM($\mathcal{T}, p_{\text{view}}$)

1. Let v be the root of $\mathcal{T}$.
2. **if** v is a leaf
3. **then** Scan-convert the object fragments in $S(v)$.
4. **else if** $p_{\text{view}} \in h_v^+$
5. **then** PAINTERSALGORITHM($\mathcal{T}^-, p_{\text{view}}$)
6. Scan-convert the object fragments in $S(v)$.
7. PAINTERSALGORITHM($\mathcal{T}^+, p_{\text{view}}$)
8. **else if** $p_{\text{view}} \in h_v^-$
9. **then** PAINTERSALGORITHM($\mathcal{T}^+, p_{\text{view}}$)
10. Scan-convert the object fragments in $S(v)$.
11. PAINTERSALGORITHM($\mathcal{T}^-, p_{\text{view}}$)
12. **else** $(* \; p_{\text{view}} \in h_v \; *)$
13. PAINTERSALGORITHM($\mathcal{T}^+, p_{\text{view}}$)
14. PAINTERSALGORITHM($\mathcal{T}^-, p_{\text{view}}$)

W algorytmie nie rysujemy wielokątów z $S(v)$, gdy p_{view} leży na prostej rozdzielającej h_v ponieważ wielokąty są płaskimi, dwuwymiarowymi obiektami i są niewidoczne z punktów leżących na płaszczyźnie zawierającej je.



Efektywność algorytmu malarza z drzewem BSP zależy od rozmiaru drzewa BSP.

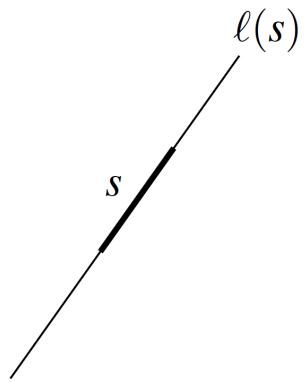
Musimy wybrać płaszczyzny rozdzielające tak, aby rozdrobnienie obiektów było bliskie minimalnego.

Zanim zobaczymy strategię budowy małych drzew BSP musimy zdecydować jakie rodzaje obiektów dopuszczamy. Ponieważ obecnie karty graficzne najlepiej radzą sobie z renderowaniem trójkątów, więc sceny 3D zazwyczaj modelowane są za pomocą trójkątów. Gdyby jednak użyto wielokątów zamiast trójkątów, to zawsze wielokąty możemy striangulować.

Konstrukcja drzew BSP

Zacznijmy od przypadku 2D i odcinków. Niech S będzie zbiorem n nieprzecinających się odcinków na płaszczyźnie.

Ograniczymy się wyłącznie do autopodziałów. Zatem niech $\ell(s)$ oznacza prostą zawierającą odcinek s.



Przy takich założeniach od razu przychodzi na myśl prosty algorytm rekurencyjny.

Algorithm 2DBSP(S)

Input. A set $S = \{s_1, s_2, \ldots, s_n\}$ of segments.

Output. A BSP tree for S.

1. **if** card$(S) \leqslant 1$
2. **then** Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set S is stored explicitly.
3. **return** $\mathcal{T}$
4. **else** ($*$ Use $\ell(s_1)$ as the splitting line. $*$)
5. $S^+ \leftarrow \{s \cap \ell(s_1)^+ : s \in S\}$; $\mathcal{T}^+ \leftarrow$2DBSP(S^+)
6. $S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\}$; $\mathcal{T}^- \leftarrow$2DBSP(S^-)
7. Create a BSP tree $\mathcal{T}$ with root node v, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{s \in S : s \subset \ell(s_1)\}$.
8. **return** $\mathcal{T}$

Algorytmy buduje drzewo BSP, ale czy jest ono małe?

Strategia brania s_1 może nie być najlepszym wyjściem.

Jeśli wybierzemy odcinek $s \in S$ taki, że $\ell(s)$ przecina tak mało odcinków jak to możliwe, to dostaniemy podejście zachłanne. Ale takie podejście nie działa zbyt dobrze dla pewnych konfiguracji odcinków. Ponadto znalezienie takiego odcinka byłoby czasochłonne.

Możemy zastosować podejście losowe, tzn. wybieramy odcinek s w sposób losowy spośród odcinków znajdujących się w S.

Algorithm 2DRANDOMBSP(S)
1. Generate a random permutation $S' = s_1, \ldots, s_n$ of the set S.
2. $\mathcal{T} \leftarrow$ 2DBSP(S')
3. **return** $\mathcal{T}$

Rozmiar wynikowego drzewa BSP zależy od użytej permutacji. Dlatego podamy oczekiwany rozmiar drzewa BSP, tzn. średni rozmiar po wszystkich $n!$ permutacjach.

Oczekiwana liczba fragmentów tworzonych przez algorytm 2DRANDOMBSP wynosi $\mathcal{O}(n \log n)$.

Przy dowodzeniu wielkości drzewa BSP z poprzedniego slajdu można pokazać, że

- dla każdego zbioru n odcinków istnieje drzewo BSP o rozmiarze $n + 2n \ln n$,
- co najmniej połowa wszystkich permutacji prowadzi do drzewa BSP o rozmiarze $n + 4n \ln n$.

Możemy wykorzystać ten fakt w algorytmie. Po uruchomieniu 2dRandomBsp sprawdzamy rozmiar drzewa. Jeśli przekracza $n + 4n \ln n$, to znów rozpoczynamy algorytm przy użyciu permutacji losowej. Oczekiwana liczba prób wynosi dwa.

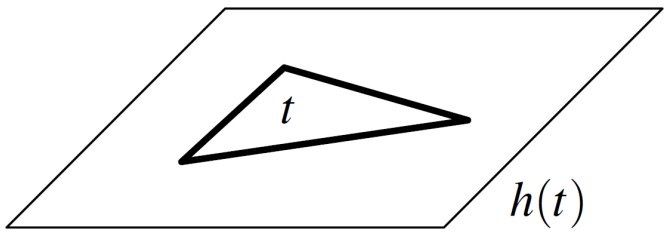
BSP rozmiaru $\mathcal{O}(n \log n)$ można obliczyć w oczekiwanym czasie $\mathcal{O}(n^2 \log n)$.

Czas działania algorytmu jest dość kiepski, ale w wielu zastosowaniach nie jest to aż tak ważne, ponieważ konstrukcja jest wykonywana off-line.

BSP rozmiaru $\mathcal{O}(n \log n)$ można obliczyć w oczekiwanym czasie $\mathcal{O}(n^2 \log n)$.

Czas działania algorytmu jest dość kiepski, ale w wielu zastosowaniach nie jest to aż tak ważne, ponieważ konstrukcja jest wykonywana off-line.

Algorytm 2D bezpośrednio uogólnia się na przypadek 3D. Niech S będzie zbiorem n nieprzecinających się trójkątów w $\mathbb{R}^3$. Ograniczamy się do autopodziałów. Niech $h(t)$ oznacza płaszczyznę zawierającą trójkąt t.



Algorithm 3DBSP(S)
Input. A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.
Output. A BSP tree for S.

1. **if** card$(S) \leqslant 1$
2. **then** Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set S is stored explicitly.
3. **return** $\mathcal{T}$
4. **else** ($*$ Use $h(t_1)$ as the splitting plane. $*$)
5. $S^+ \leftarrow \{t \cap h(t_1)^+ : t \in S\}$; $\mathcal{T}^+ \leftarrow$3DBSP(S^+)
6. $S^- \leftarrow \{t \cap h(t_1)^- : t \in S\}$; $\mathcal{T}^- \leftarrow$3DBSP(S^-)
7. Create a BSP tree $\mathcal{T}$ with root node v, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{t \in S : t \subset h(t_1)\}$.
8. **return** $\mathcal{T}$

Rozmiar otrzymanego drzewa BSP ponownie zależy od kolejności trójkątów. Pewne porządki dają więcej fragmentów niż inne. Zatem jak dla wersji 2D możemy spróbować ustawienia trójkątów w sposób losowy.

W praktyce daje to zwykle dobre efekty, ale nie wiadomo, jak teoretycznie analizować oczekiwane zachowanie się tego algorytmu.