Wyznaczanie powierzchni widocznych

Mirosław Głowacki

Akademia Górniczo-Hutnicza w Krakowie

Wiadomości wstępne

Treść wykładu

- Dwa główne typy algorytmów:
  - algorytmy z precyzją ekranową
  - algorytmy z precyzją obrazową
- Wykorzystywane metody
  - spójność
  - problem przekształcenia perspektywicznego
  - bryły i prostokąty ograniczające
  - przednie i tylne ściany wielokątów
  - podział adaptacyjny - hierarchia
- Podstawowe algorytmy stosowane w praktyce:
  - algorytm z-bufora
  - algorytm przeglądania
  - metoda śledzenia promieni
  - algorytm podziału przestrzennego
    - algorytm drzewa binarnego podziału przestrzeni
  - algorytm podziału powierzchni Warnocka
  - algorytm z listą priorytetów
  - algorytm sortowania ze względu na głębokość
  - algorytmy wyznaczania powierzchni krzywoliniowych

Wyznaczanie powierzchni widocznych

- Ten proces jest znany jako
  - *wyznaczanie linii* albo *powierzchni widocznych* albo
  - *eliminowanie linii* albo *powierzchni niewidocznych*
- Zakłada się, że linie są krawędziami powierzchni nieprzezroczystych, które mogą zasłaniać krawędzie innych powierzchni znajdujących się dalej od obserwatora

Wyznaczanie powierzchni widocznych

- Dlatego mówimy o ogólnym procesie jako o *wyznaczaniu powierzchni widocznych*
- Chociaż ta podstawowa zasada jest prosta, jej realizacja wymaga *znacznej mocy obliczeniowej*
- W konsekwencji wymaga *dużego czasu* obliczeniowego na konwencjonalnych maszynach

Wyznaczanie powierzchni widocznych

- Dwa podstawowe podejścia do rozwiązania tego problemu.
- W obu przypadkach możemy traktować obiekt jako składający się z jednego albo większej liczby wielokątów (albo bardziej złożonych powierzchni).

Pierwsze podejście

- W pierwszym podejściu określa się, który z *n* obiektów jest widoczny w każdym pikselu obrazu
- Pseudokod dla tego podejścia wygląda następująco:

for *(każdy **piksel** obrazu)*

{

wyznaczyć obiekt najbliższy obserwatora, który jest napotkany przez promień rzutowania przechodzący przez piksel;

narysować piksel o odpowiedniej barwie;

}

Pierwsze podejście

- Bezpośredni sposób wykonania tego dla jednego piksela wymaga sprawdzenia wszystkich *n* obiektów w celu ustalenia, który leży najbliżej obserwatora wzdłuż promienia rzutowania przechodzącego przez piksel
- Dla *p* pikseli nakład jest proporcjonalny do ***np***, przy czym *p* jest większe niż 1 milion dla monitora o przeciętnie dużej rozdzielczości

Drugie podejście

- Polega na porównywaniu obiektów bezpośrednio każdy z każdym i eliminowaniu tych obiektów albo ich części, które są niewidoczne
- Możemy to wykonać w sposób bezpośredni, porównując każdy z *n* obiektów ze sobą i z innymi obiektami i odrzucać niewidoczne części

Drugie podejście

- Nakład obliczeniowy jest tu proporcjonalny do n^2
- Chociaż to drugie podejście mogłoby się wydawać lepsze dla $n << p$, to jego poszczególne kroki są na ogół:
  - bardziej złożone,
  - zajmują więcej czasu,
  - w efekcie metoda często jest wolniejsza i trudniejsza w implementacji.

Drugie podejście

- Można to przedstawić za pomocą pseudokodu:

  for *(każdy **obiekt**)*

  {

  wyznaczyć te części obiektu, których rzut nie jest zasłonięty przez inne części tego lub innych obiektów;

  narysować te części z wykorzystaniem odpowiedniej barwy;

  }

Precyzja obrazowa i obiektowa

- Te dwa prototypowe podejścia będziemy określali odpowiednio jako:
  - algorytm z *precyzją obrazową*
  - algorytm z *precyzją obiektową*
- Algorytm z precyzją obrazową są na ogół wykonywane z dokładnością, jaką ma urządzenie wyświetlające, i określają widoczność w każdym pikselu
- Algorytmy z precyzją obiektową są wykonywane z dokładnością, z jaką jest zdefiniowany każdy obiekt, i określają widoczność każdego obiektu

Precyzja obrazowa i obiektowa

- Ponieważ obliczenia z precyzją obiektową są wykonywane bez względu na rozdzielczość określonego wyświetlacza, musi po nich następować krok, w którym obiekty są faktycznie wyświetlane z wymaganą rozdzielczością
- Tylko ten ostatni krok wyświetlania musi być powtarzany, jeżeli trzeba zmienić na przykład wielkość skończonego obrazu po to, żeby pokryć inną liczbę pikseli na monitorze rastrowym
- Wynika to stąd, że geometria każdego rzutu obiektu widocznego jest reprezentowana z pełną rozdzielczością bazy danych obiektu

Precyzja obrazowa i obiektowa

- Dla kontrastu rozważmy powiększanie obrazu utworzonego przez algorytm z precyzją obrazową
- Ponieważ obliczenia związane z powierzchnią widoczną były wykonane początkowo z małą rozdzielczością, muszą być wykonane ponownie, jeżeli chcemy pokazać dalsze szczegóły
- Dlatego algorytmy z precyzją obrazową są narażone na powstawanie zakłóceń przy wyznaczaniu widoczności, podczas gdy algorytmy z precyzją obiektową nie

Precyzja obrazowa i obiektowa

- W przeszłości:
  - Wyświetlacze wektorowe miały duże przestrzenie adresowe (4096 x 4096 nawet we wczesnych systemach) i ostre ograniczenia, jeśli chodzi o liczbę odcinków i obiektów, które mogły być wyświetlane.
  - Wyświetlacze rastrowe miały ograniczoną przestrzeń adresową (256 x 256 we wczesnych systemach) i zdolność do wyświetlania potencjalnie nieograniczonej liczby obiektów
- W późniejszych algorytmach często łączono obliczenia z precyzją obiektową z obliczeniami z precyzją obrazową
  - obliczenia z precyzją obiektową były wybierane ze względu na dokładność
  - obliczenia z precyzją obrazową ze względu na szybkość

Wykorzystywane własności obiektów

Spójność

- Algorytmy wyznaczania powierzchni widocznych mogą wykorzystywać ***spójność*** – stopień, w jakim fragmenty sceny albo ich rzuty wykazują lokalne podobieństwa
- Sceny na ogół zawierają obiekty, których właściwości *zmieniają się gładko* (nie skokowo) od jednej części do drugiej.
- Spójność jest występującym brakiem nieciągłości właściwości takich jak
  - głębokość,
  - barwa,
  - tekstura
- i *efektów*, które one wytworzą w obrazach, umożliwiających rozróżnienie obiektów

Spójność

- Spójność wykorzystujemy wówczas, gdy ponownie korzystamy z obliczeń wykonywanych dla jednej części otoczenia albo obrazu w stosunku do innych bliskich części:
  - albo bez zmian
  - albo z przyrostowymi zmianami,
- Są one bardziej efektywne do wykonania niż przetwarzanie informacji na nowo

Spójność

- Zidentyfikowano wiele różnych rodzajów spójności:
  - spójność obiektów
  - spójność ściany
  - spójność krawędzi
  - implikowana spójność krawędzi
  - spójność liniowa
  - spójność powierzchni
  - spójność głębokości
  - spójność ramek

Spójność obiektów

- Jeżeli jeden obiekt jest całkowicie odseparowany od drugiego, to może być potrzebne wykonanie porównań tylko między tymi dwoma obiektami, a nie między ich ścianami składowymi albo krawędziami.
- Na przykład, jeżeli wszystkie części obiektu *A* są dalej od obserwatora niż wszystkie części obiektu *B*, to żadna ze ścian *A* nie musi być porównywana ze ścianami *B* w celu sprawdzenia, czy zasłaniają one ściany *B*

Spójność ściany

- Właściwości powierzchni na ogół zmieniają się gładko na powierzchni, dzięki czemu obliczenia dla jednej części ściany można modyfikować przyrostowo w celu zastosowania do sąsiednich części
- W niektórych modelach może istnieć gwarancja, że ściany nie przenikają się

Spójność krawędzi

- Widoczność krawędzi może zmieniać się tylko tam, gdzie przecina ona widoczną krawędź albo przecina inną widoczną ścianę
- **Implikowana spójność krawędzi**. Jeżeli jedna płaska ściana przecina drugą, to ich linia przecięcia (**implikowana krawędź**) może być określona na podstawie dwóch punktów przecięcia

Spójność liniowa i powierzchni

- **Spójność liniowa**. Zbiór widocznych odcinków obiektu określony dla jednej linii obrazu na ogół niewiele się różni od zbioru dla poprzedniej linii.
- **Spójność powierzchni**. Grupa sąsiednich pikseli jest często pokryta przez tę samą widoczną ścianę
- Specjalnym rodzajem spójności powierzchni jest *spójność segmentowa*, która odnosi się do widoczności ściany w obrębie segmentu sąsiednich pikseli w linii

Spójność głębokości

- Sąsiednie części tej samej powierzchni mają na ogół *podobną odległość* od obserwatora (**głębokość**), podczas gdy różne powierzchnie w tym samym miejscu ekranu mają na ogół znacznie różniące się głębokości
- Po obliczeniu głębokości jednego punktu na powierzchni, głębokość punktów pozostałej części powierzchni może być często określona za pomocą prostego równania różnicowego

Spójność ramek

- Jest prawdopodobne, że obrazy tej samej sceny w dwóch kolejnych chwilach są bardzo podobne, mimo niewielkich zmian obiektów i punktu obserwacji
- Obliczenia wykonane dla jednego obrazu mogą być użyte dla kolejnego obrazu w sekwencji

Przekształcenie perspektywiczne

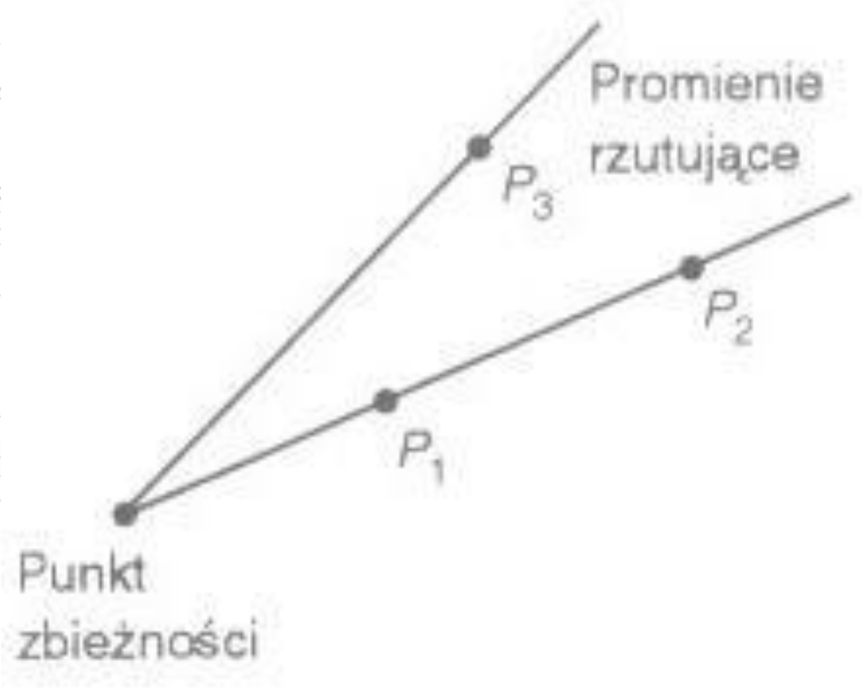
- Określenie powierzchni widocznych musi nastąpić oczywiście w przestrzeni 3D przed rzutowaniem na płaszczyznę 2D
- Rzutowanie, które niszczy informację o głębokości potrzebną dla porównań głębokości.
- Niezależnie od rodzaju wybranego rzutu podstawowe porównanie głębokości w punkcie może być na ogół sprowadzone do następujących pytań:

1. Czy jeden z dwóch punktów

$$P_1 = (x_1, y_1, z_1)$$
$$P_2 = (x_2, y_2, z_2)$$

zasłania drugi?

2. Czy leżą one na tym samym promieniu rzutującym?



Przekształcenie perspektywiczne

- Jeżeli tak, to porównuje się je w celu określenia, który z punktów jest bliżej obserwatora.
- Jeżeli nie, to żaden punkt nie zasłania drugiego.
- Porównania głębokości są na ogół wykonywane po zastosowaniu przekształcenia normalizującego, tak że promienie rzutujące są:
  - równoległe do osi z w rzucie równoległym
  - albo wychodzą z punktu zbieżności dla rzutu perspektywicznego
- W rzucie równoległym punkty są na tym samym promieniu rzutowania, jeżeli:

$$x_1 = x_2 \qquad y_1 = y_2$$

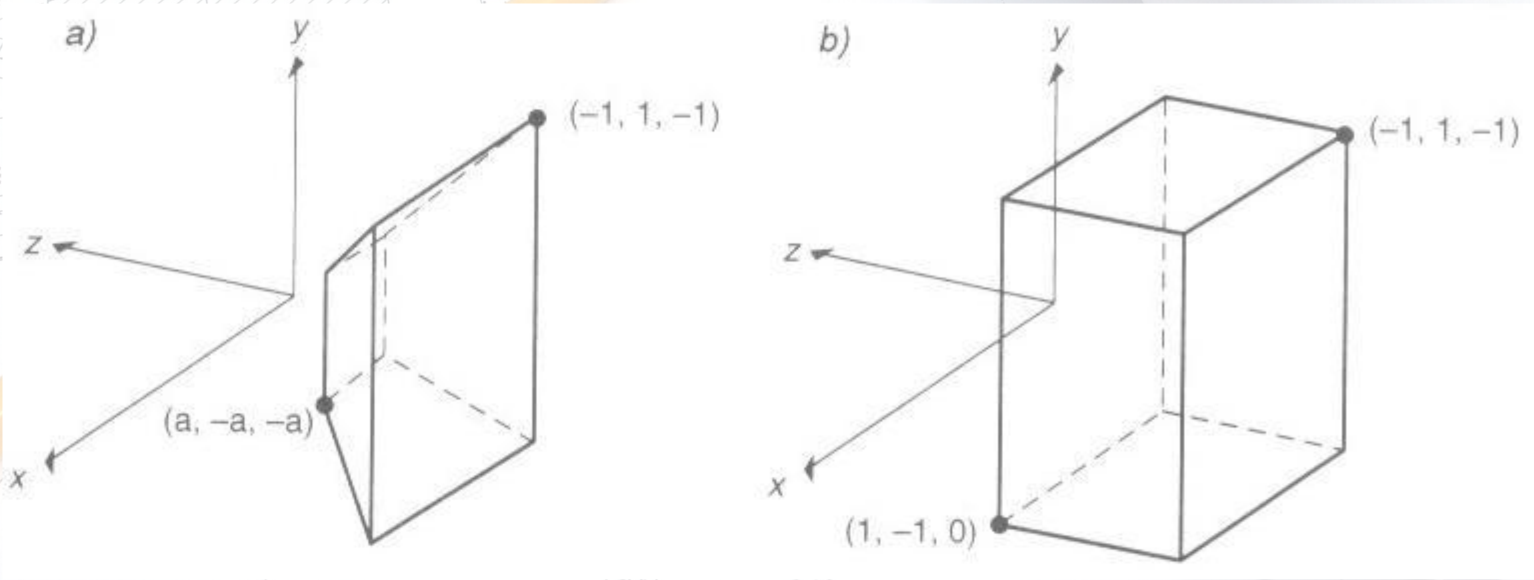
- Dla rzutu perspektywicznego musimy, niestety, wykonać cztery dzielenia w celu sprawdzenia, czy

$$x_1/z_1 = x_2/z_2 \qquad y_1/z_1 = y_2/z_2$$

- Jeśli tak, to punkty leżą na tym samym promieniu rzutowania

Przekształcenie perspektywiczne

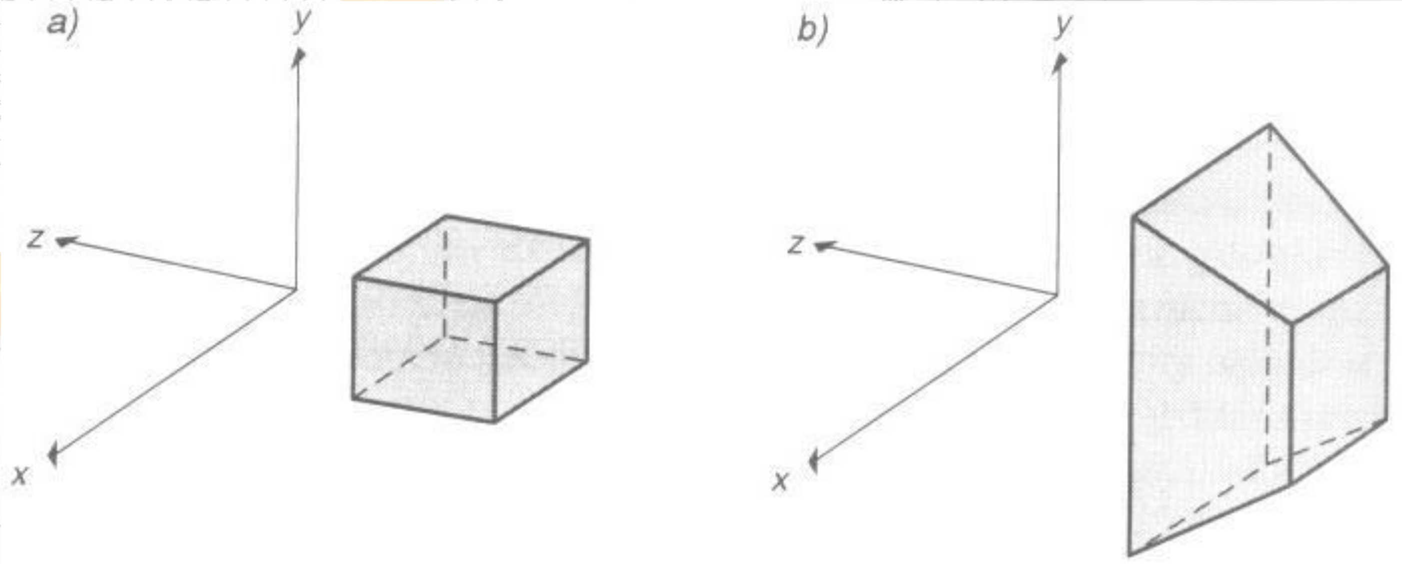
- Niepotrzebnych dzieleń można uniknąć dokonując najpierw przekształcenia obiektu 3D w układ współrzędnych ekranu 3D
- Rzut równoległy przekształconego obiektu będzie taki sam jak rzut perspektywiczny nieprzekształconego obiektu.
- Wtedy test sprawdzający przesłanianie punktów jest taki sam jak w rzucie równoległym.



Znormalizowana bryła widzenia perspektywicznego przed i po przekształceniu perspektywicznym

Przekształcenie perspektywiczne

- Przekształcenie perspektywiczne zniekształca obiekty i przesuwa środek rzutowania do nieskończoności na osi z i promienie rzutowania stają się równoległe.
- Na rysunku pokazano zniekształcenie, jakiemu ulega sześcian przy tym przekształceniu.



Przekształcenie perspektywiczne

- Istotą tego przekształcenia jest to, że zachowuje:
  - względną głębokość,
  - odcinki i płaszczyzny i
  - równocześnie wykonuje perspektywiczny skrót.
- Dzielenie prowadzące do tego skrótu jest wykonywane tylko raz dla punktu, a nie za każdym razem, gdy dwa punkty są porównywane.
- Macierz

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_{min}} & \dfrac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad z_{min} \neq -1$$

przekształca znormalizowaną bryłę widzenia perspektywicznego w prostopadłościan ograniczony przez:

$$-1 \leq x \leq 1, \quad -1 \leq y \leq 1, \quad -1 \leq z \leq 0$$

Przekształcenie perspektywiczne

- Przed zastosowaniem macierzy **M** można wykonać obcinanie względem znormalizowanej bryły widzenia, czyli ostrosłupa ściętego, ale wtedy wyniki obcinania musza być pomnożone przez **M**.
- Atrakcyjniejsze rozwiązanie polega na włączeniu **M** do normalizującego przekształcenia perspektywicznego
- Wtedy potrzebne jest tylko jedno mnożenie przez macierz, a potem obcinanie we współrzędnych jednorodnych przed dzieleniem.
- Jeżeli wyniki mnożenia oznaczymy jako

$$(X, Y, Z, W)$$

to dla granice obcinania będą następujące:

$$-W \leq X \leq W, \qquad -W \leq Y \leq W, \quad -W \leq Z \leq 0$$

Przekształcenie perspektywiczne

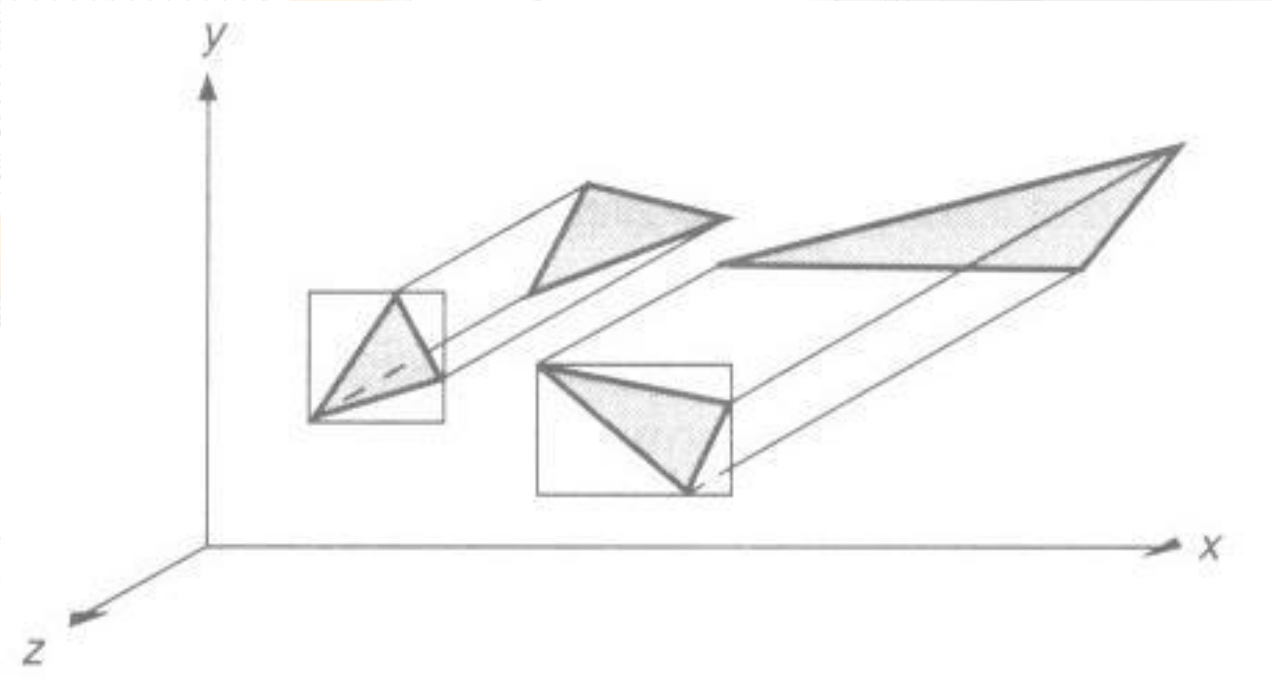
- Te ograniczenia wynikają z równania dla układu rzeczywistego, w którym zastąpi się ***x, y*** i ***z*** odpowiednio przez **x/W**, **y/W** *i* **z/W** .
- Macierz **M** została zapisana przy założeniu, że bryła widzenia jest w ujemnej półprzestrzeni ***z.***
- Ze względu jednak na wygodę zapisu w naszych przykładach będą często używane raczej malejące dodatnie wartości ***z*** niż malejące ujemne wartości *z* po to, żeby pokazać rosnącą odległość od obserwatora.
- W wielu systemach graficznych zamiast prawoskrętnego układu współrzędnych korzysta się z lewoskrętnego układu współrzędnych widzenia, w którym rosnące dodatnie wartości *z* odpowiadają zwiększającej się odległości od obserwatora.

Przekształcenie perspektywiczne

- Możemy teraz wyznaczyć powierzchnie widoczne bez obawy przed komplikacjami sugerowanymi przez wzajemne zasłanianie punktów na wspólnej prostej.
- Oczywiście w przypadku rzutu równoległego przekształcenie perspektywiczne **M** nie jest konieczne, ponieważ przekształcenie normalizujące dla rzutu równoległego powoduje, że promienie rzutowania są równoległe do osi **z**.

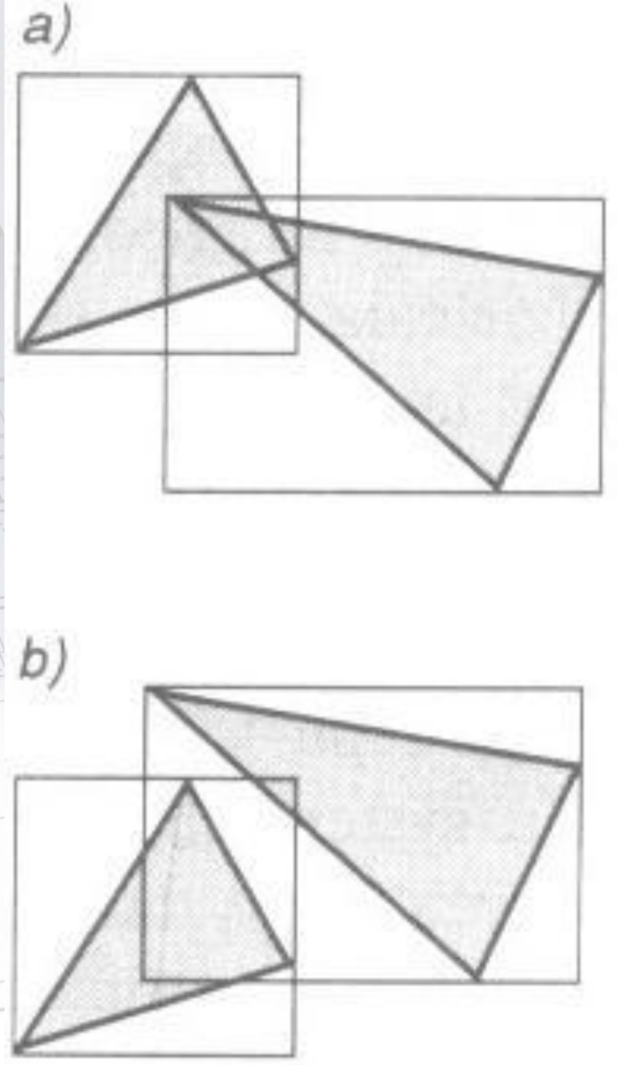
Prostokąty i bryły ograniczające

- Prostokąty ograniczające wprowadzone w celu uniknięcia niepotrzebnego obcinania są również często używane, aby uniknąć niepotrzebnych porównań obiektów albo ich rzutów.
- Na rysunku pokazano dwa obiekty (wielokąty 3D), ich rzuty i prostokąty opisane na nich o bokach równoległych do osi.



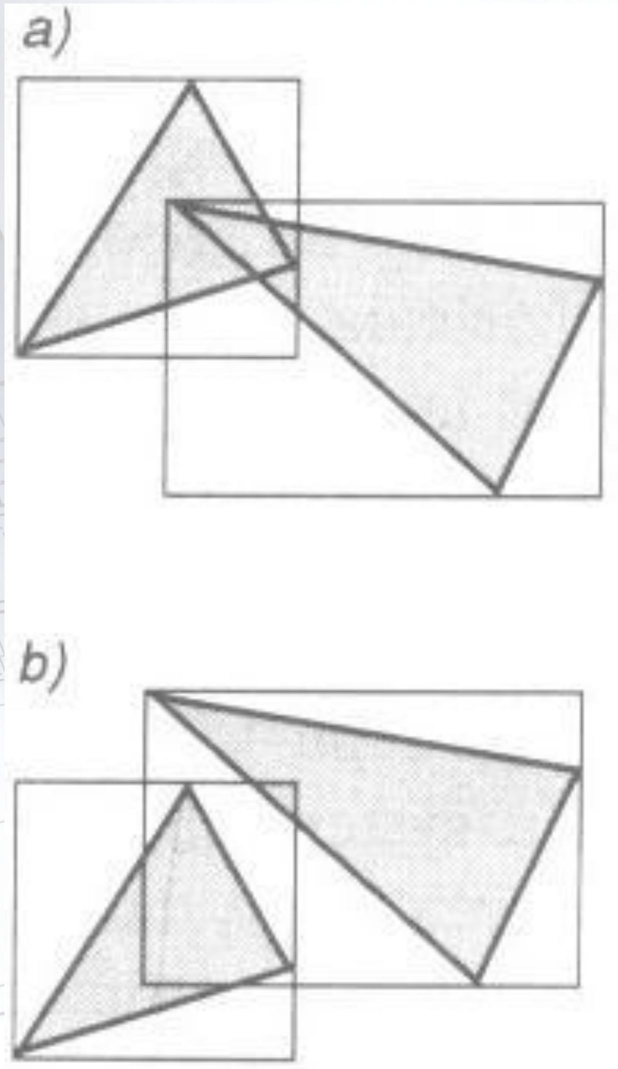
Prostokąty i bryły ograniczające

- Zakłada się, że obiekty mają być poddane przekształceniu perspektywicznemu za pomocą macierzy **M**.
- Dla wielokątów rzut prostokątny na płaszczyznę *(x,y)* jest wykonywany w sposób trywialny na zasadzie pominięcia współrzędnych z wierzchołków.
- Jeżeli prostokąty ograniczające nie nachodzą na siebie nie trzeba wykonywać testowania rzutów ze względu na przecinanie się ich ze sobą.
- Jeżeli prostokąty ograniczające przecinają się, to może wystąpić jedna z dwóch sytuacji
  - albo rzuty przecinają się jak na rys. a)
  - albo nie przecinają się jak na rys. b)



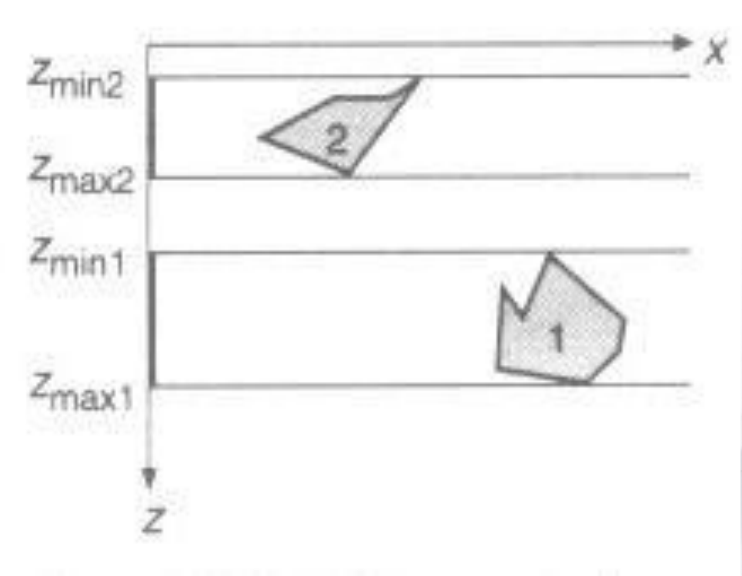
Prostokąty i bryły ograniczające

- W obu przypadkach trzeba wykonać więcej porównań w celu sprawdzenia, czy rzuty się przecinają.
- W sytuacji z rysunku b) wynik porównań pokaże, że dwa rzuty nie przecinają się, co oznacza, że fakt przecinania się prostokątów ograniczających był fałszywym alarmem.
- Testowanie prostokątów ograniczających daje w efekcie to samo co w przypadku testów odrzucania przy obcinaniu.



Prostokąty i bryły ograniczające

- Prostokąty ograniczające mogą być używane, do otaczania samych obiektów, a nie ich rzutów
- W takim przypadku miejsce prostokąta zajmuje bryła określana jako ***bryła ograniczająca***.
- Można także koncepcję prostokąta ograniczającego przenieść do jednego wymiaru w celu określenia, czy na przykład dwa obiekty nachodzą na siebie we współrzędnej ***z***.
- Na rysunku pokazano wykorzystanie koncepcji w takim przypadku – obszar ograniczający jest nieskończoną objętością ograniczającą wartość minimalną i maksymalną *z* dla każdego obiektu.

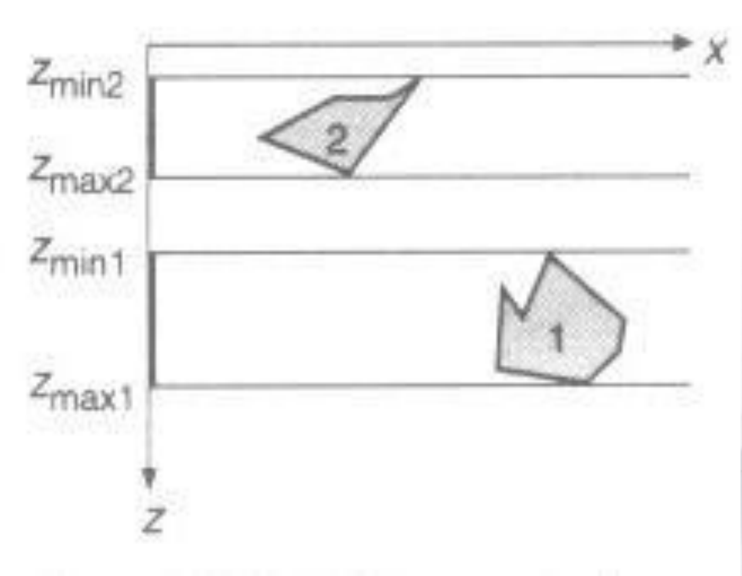


Prostokąty i bryły ograniczające

- Nie ma nakładania się obiektów we współrzędnej z, jeżeli:

$$z_{max2} < z_{min1} \quad lub \quad z_{max1} < z_{min2}$$

- Porównywanie wartości ograniczeń minimalnej i maksymalnej w jednym lub więcej wymiarach jest znane jako ***testowanie min-max***.
- Przy porównywaniu typu *min-max* obszarów ograniczających najbardziej skomplikowaną częścią pracy jest **znalezienie** samych **obszarów** ograniczających.



Prostokąty i bryły ograniczające

- Dla wielokątów i innych obiektów, które są całkowicie zawarte w wypukłym wielokącie opisanym na zbiorze definiujących punktów, obszar ograniczający można obliczyć przeszukując iteracyjnie listę współrzędnych punktów i rejestrując największe i najmniejsze wartości dla każdej współrzędnej.
- Obszary ograniczające są używane do:
  - porównywania dwóch obiektów
  - porównywania ich rzutów,
  - określania, czy promień rzutowania przecina obiekt
- Wiąże się to z obliczaniem przecięcia punktu z rzutem 2D albo wektora z obiektem 3D.

Prostokąty i bryły ograniczające

- Dotychczas omawialiśmy tylko obszary ograniczające typu *min-max*, możliwe jest użycie innych brył ograniczających.
- Jakiej bryły użyć najlepiej? Odpowiedź zależy:
  - od kosztu wykonywania testów na samej bryle ograniczającej,
  - od tego, jak dobrze bryła zabezpiecza otaczany obiekt przed testami, które nie dają przecięcia.
- Weghorst, Hooper i Greenberg traktują wybór bryły ograniczającej jako problem **minimalizacji** całkowitej funkcji **kosztu** *T* **testu** przecięcia dla obiektu.

Prostokąty i bryły ograniczające

- Można to wyrazić jako

$$T = bB + oO$$

- przy czym
  - *b* jest liczbą określającą, ile razy bryła ograniczająca jest testowana ze względu na przecięcie,
  - *B* jest kosztem wykonania testu przecięcia z bryłą ograniczającą,
  - *o* jest liczbą określającą, ile razy obiekt jest testowany ze względu na przecięcie (ile razy rzeczywiście jest przecinana bryła ograniczająca),
  - *O* jest kosztem wykonania testu przecięcia obiektu.

Prostokąty i bryły ograniczające

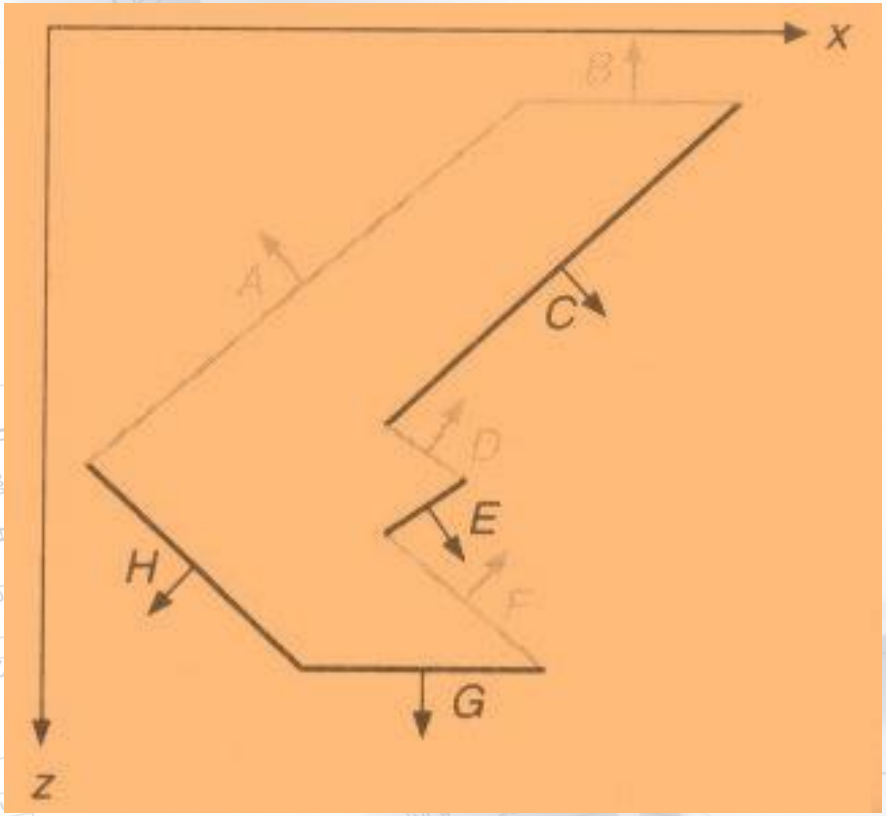
- Ponieważ test przecięcia obiektu jest wykonywany tylko wówczas, gdy rzeczywiście jest przecinana bryła ograniczająca, to

$$o \leq b$$

- *O* i *b* są stałymi dla określonego obiektu
- Jest wykonywany zbiór testów
- *B* i *o* zmieniają się zależnie od kształtu i wielkości bryły ograniczającej.
- Im ciaśniejsza jest bryła ograniczająca, co minimalizuje *o,* tym na ogół jest większe *B.*
- Efektywność bryły ograniczającej może również zależeć od orientacji obiektu albo rodzaju obiektów, z którymi obiekt będzie przecinany.

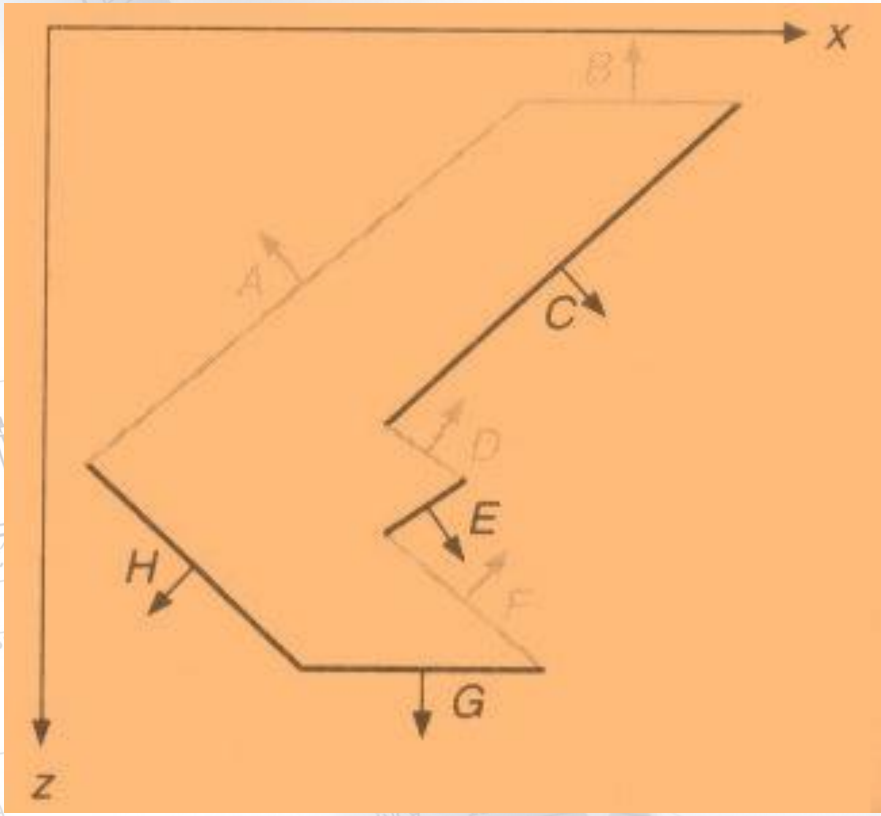
Wybieranie tylnych ścian

- Jeżeli obiekt jest aproksymowany przez wielościan, to jego ściany wielokątowe całkowicie otaczają jego wnętrze.
- Załóżmy, że wszystkie wielokąty zostały tak zdefiniowane, że ich normalne są skierowane na zewnątrz wielościanu.
- Jeżeli żadna część wnętrza wielościanu nie jest widoczna przez przednią płaszczyznę obcinającą, to te wielokąty, których normalne są skierowane od obserwatora, leżą w tej części wielościanu, której widoczność jest całkowicie zablokowana przez bliższe wielokąty.



Wybieranie tylnych ścian

- Takie *wielokąty niewidoczne* można wyeliminować z dalszego procesu za pomocą metody ***wybierania tylnych ścian***.
- Przez analogię te wielokąty, które nie są tylnymi, są określane jako *przednie ściany*.
- Na rysunku eliminowane są wielokąty skierowane do tyłu (A, B, D, F),
- Wielokąty skierowane do przodu (C, E, G, H) są zachowane



Wybieranie tylnych ścian

- We współrzędnych oka **tylny wielokąt** może być zidentyfikowany przez nieujemny iloczyn skalarny normalnej powierzchni i wektora od punktu zbieżności do dowolnego punktu wielokąta.
- Iloczyn skalarny jest:
  - dodatni dla tylnego wielokąta
  - zerowy dla wielokąta widzianego jako krawędź
- Promień rzutujący przechodzący przez wielościan przecina taką samą liczbę wielokątów skierowanych do tyłu jak wielokątów skierowanych do przodu.
- Punkt w rzucie wielościanu leży na rzutach takiej samej liczby wielokątów skierowanych do tyłu jak wielokątów skierowanych do przodu.
- Dlatego wybieranie wielokątów skierowanych do tyłu zmniejsza o połowę liczbę wielokątów, które trzeba wziąć pod uwagę dla każdego piksela w algorytmie powierzchni widocznych z precyzją obrazową.
- Średnio około połowa wielokątów w wielościanie jest skierowana do tyłu.

Podział przestrzenny

- Podział przestrzenny umożliwia rozbicie dużego problemu na kilka mniejszych.
- Podstawowe podejście polega na przypisaniu obiektów albo ich rzutów do **przestrzennie spójnych grup** w czasie przetwarzania wstępnego.
- Na przykład możemy podzielić rzutnię regularną siatką prostokątną **2D** i określić, w której komórce siatki leży rzut obiektu.
- Rzuty muszą być porównane ze względu na nakładanie się tylko z innymi rzutami, które znajdują się w odpowiednich polach siatki.

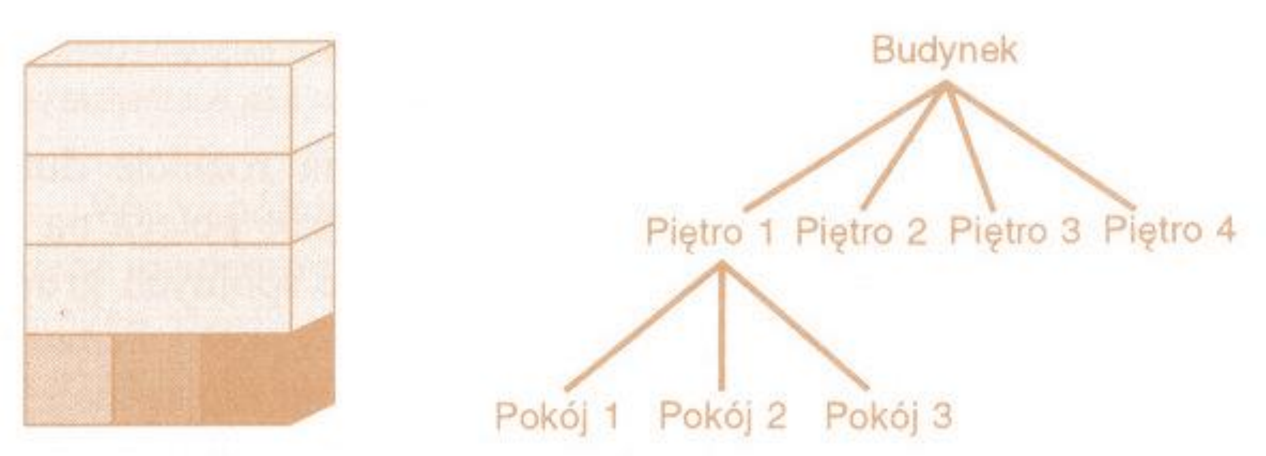
Podział przestrzenny

- Podział przestrzenny może być używany do nakładania regularnej siatki 3D na obiekty w środowisku.
- Proces określania, które obiekty są przecinane przez promień rzutujący, można wobec tego przyspieszyć przez
  - wstępne określenie, które części przecina promień,
  - późniejsze testowanie tylko tych obiektów, które leżą w tych częściach.
- Jeżeli obrazowane obiekty są nierównomiernie rozłożone w przestrzeni, to efektywniejsze może być stosowanie **podziału adaptacyjnego**, w którym wielkość każdej części zmienia się.

Podział adaptacyjny

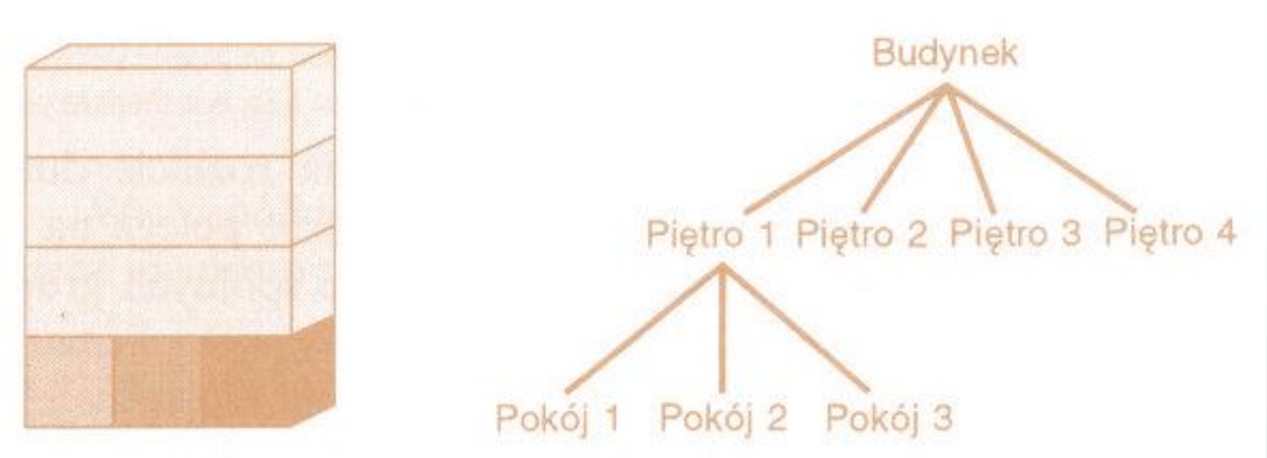
- Jedno z podejść do podziału adaptacyjnego polega na tym, żeby **dzielić przestrzeń rekursywnie**, dopóki dla każdej części nie zostanie spełnione pewne kryterium zakończenia podziału – na przykład osiągnięcie w danej części pewnej dopuszczalnie dużej liczby obiektów.
- Szczególnie atrakcyjne dla tego celu są **drzewa czwórkowe**, **drzewa ósemkowe** i **drzewa BSP.**

Hierarchia

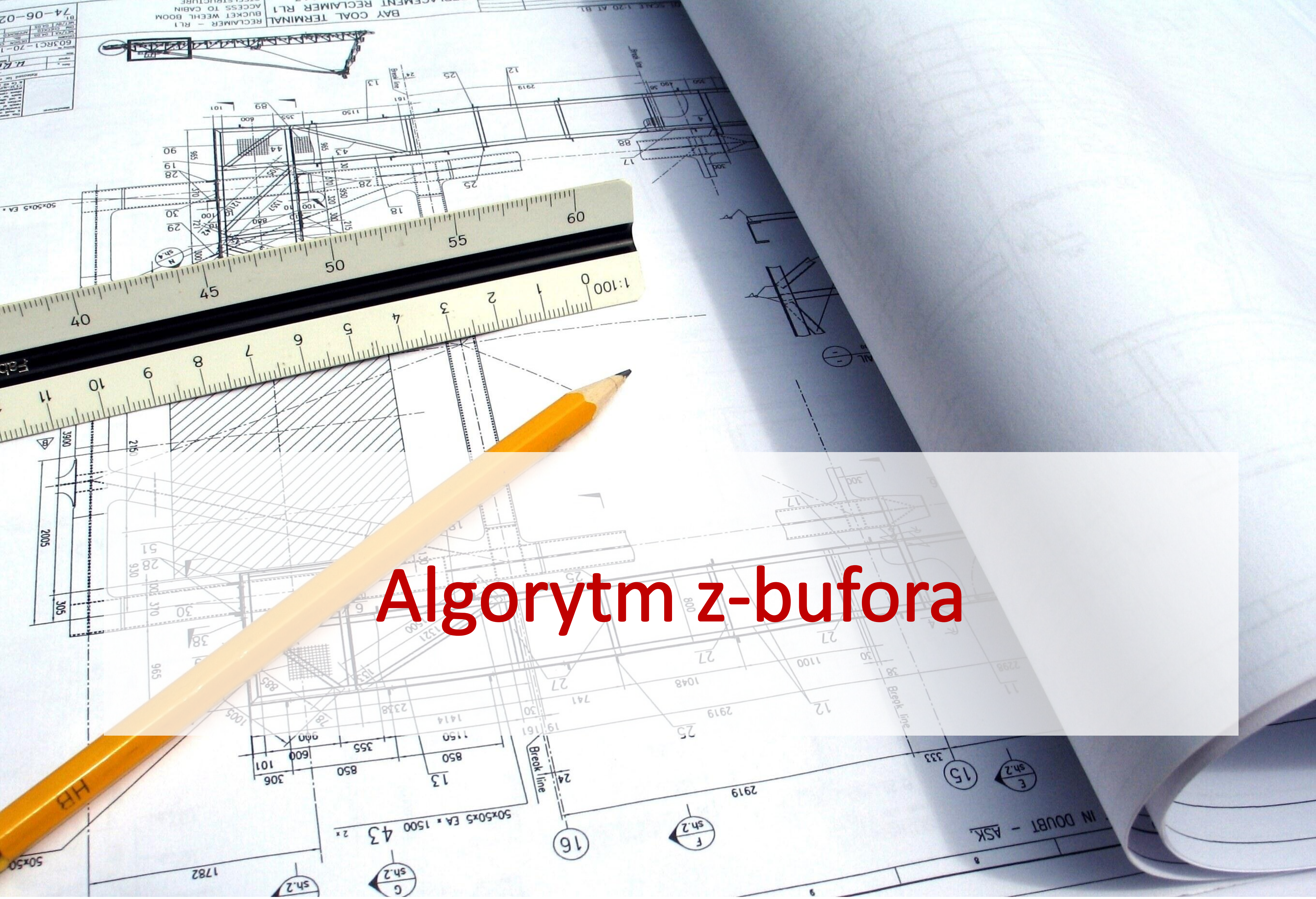


- Hierarchia może być użyteczna do wiązania struktury i ruchu różnych obiektów.
- **Zagnieżdżony model hierarchiczny**, w którym każdy potomek jest traktowany jako część swojego przodka, może być również użyty do ograniczenia liczby porównań obiektów, które są potrzebne w algorytmie powierzchni widocznych.
- Obiekt na jednym poziomie hierarchii może służyć jako rozszerzenie dla swoich potomków, jeżeli są oni całkowicie w nim zawarci, jak pokazano na rysunku.

Hierarchia



- Jeżeli dwa obiekty w hierarchii nie przetną się, to obiekty potomne jednego z tych przodków nie muszą być testowane na przecięcie z obiektami będącymi potomkami drugiego.
- Podobnie, jeżeli stwierdzi się, że promień rzutujący przecina obiekt w hierarchii, to musi być testowany na przecięcie z potomkami obiektu.
- Takie wykorzystanie hierarchii jest ważną odmianą **spójności obiektu**.
- Na rysunku, gdy promień rzutujący przecina budynek i podłogę, trzeba wykonać testowanie przecięć z pokojami od 1 do 3



Algorytm z-bufora

Algorytm z-bufora

- Podczas odwzorowywania trójwymiarowej sceny na obraz dwuwymiarowy jednostka rasteryzująca przechowuje wyniki swojej pracy w wydzielonej pamięci lokalnej akceleratora 3D, nazywanej buforem ramki (ang. frame buffer).
- To właśnie z tego obszaru w ostatniej fazie generowania obrazu na ekran monitora wysłana zostanie gotowa, pojedyncza klatka tworzonej sceny 3D.
- W pamięci RAM karty graficznej wydzielona zostaje również matryca odpowiadająca swoją wielkością rozdzielczości ekranu, a głębokością 16, 24 lub 32 bitom, w zależności od zastosowanej głębi współrzędnej ***z***.

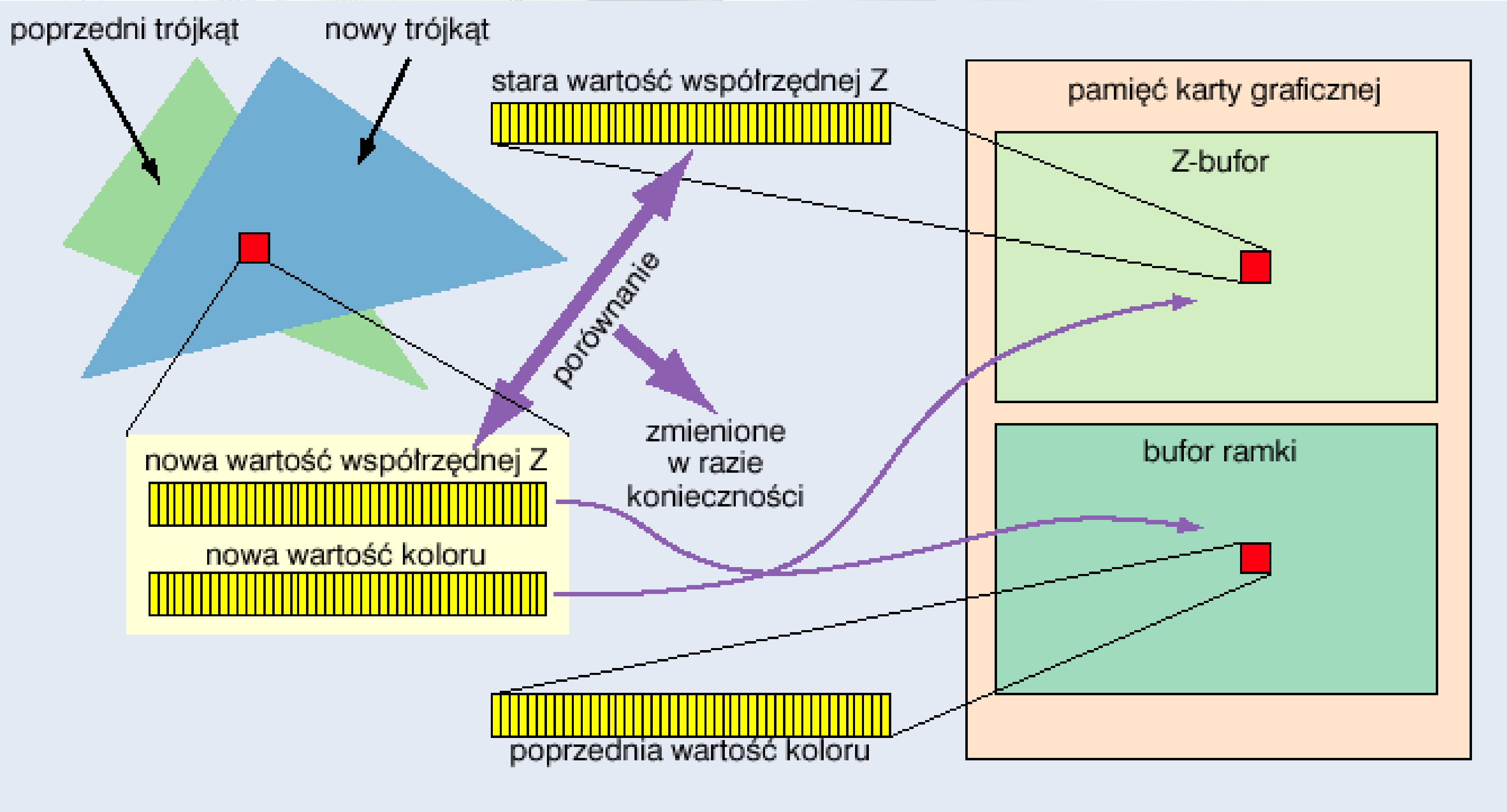
Algorytm z-bufora

- Ten duży objętościowo fragment pamięci lokalnej karty, mający za zadanie przechowywać dane o współrzędnej **z** nazywany jest z-buforem (ang. z-buffer).
- Zasada działania z-bufora *z* polega na tym, że np. dla dwóch trójkątów przed ich narysowaniem porównuje się ich współrzędne **z** z wartością zapamiętaną w *z*-buforze.
- Jeśli nowy rysowany punkt ma wartość niższą (to znaczy zasłania poprzedni obiekt), jest rysowany
- Cały proces powtarzany jest dla każdego obiektu generowanej sceny 3D.

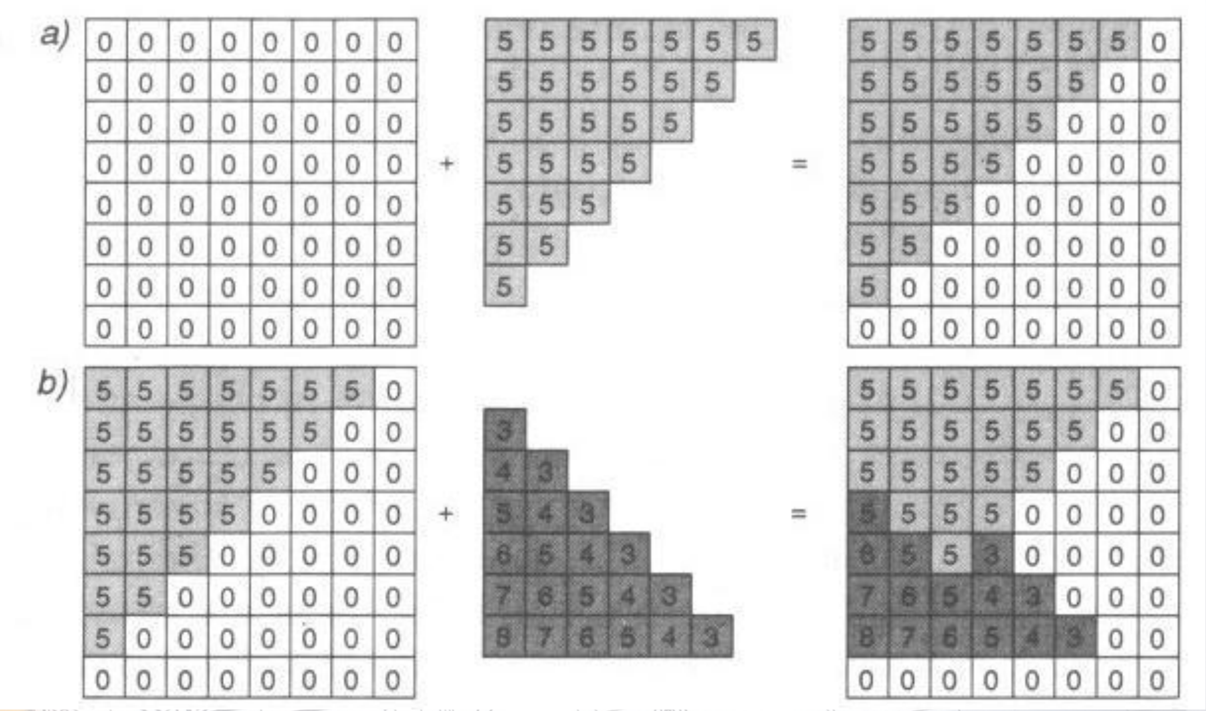
Algorytm z-bufora

- **Algorytm z-bufora**, albo *bufora głębokości,* z grupy algorytmów z precyzją obrazową, został opracowany przez Catmulla.
- Jest to jeden z najprostszych do realizacji sprzętowej albo programowej algorytmów znajdowania powierzchni widocznych.
- Wymaga on korzystania obok pamięci obrazu ***F***, w której są pamiętane wartości barw, z **z-bufora Z** o takiej samej liczbie pozycji, w której jest pamiętana wartość *z* dla każdego piksela.
- Na początku bufor jest zerowany, co odpowiada zapamiętaniu wartości **z** dla tylnej ściany obcinającej, a do bufora obrazu jest wpisywana barwa tła.
- Największa wartość, jaka może być wpisana w *z*-buforze, reprezentuje ***z*** dla przedniej ściany obcinającej.
- Wielokąty są przeglądane wierszami – kolejność wielokątów jest dowolna.

Alorytm z–bufora



Algorytm z-bufora



- Na rysunku pokazano dodanie dwóch wielokątów do obrazu.
- Barwa każdego piksela jest reprezentowana przez odcień szarości, a ***z*** jest reprezentowane przez liczbę.
- Pamiętając naszą dyskusję o spójności głębokości możemy uprościć obliczanie z dla każdego punktu w przeglądanym wierszu korzystając z faktu, że wielokąt jest płaski.

Algorytm *z*-bufora

- Zwykle w celu obliczenia ***z*** należy rozwiązać równanie płaszczyzny

$$Ax + By + Cz + D = 0$$

dla zmiennej

$$z = \frac{-D - Ax - By}{C}$$

- Teraz jeżeli w punkcie (*x*, *y*) z równania otrzymamy *z*, to w punkcie (*x*+Δ*x*, *y*) wartość wynosi

$$z_1 = \frac{A}{C}\Delta x$$

- Jeżeli *z*(*x*, *y*) jest dane, to obliczenie *z*(*x+1*, *y*) wymaga tylko jednego odejmowania bo A/C jest stałe i Δ*x* = 1.

Algorytm z-bufora

- W czasie procesu konwersji, jeżeli rozpatrywany punkt nie jest dalej od obserwatora niż punkt, którego barwa i głębokość są zapisane w buforach, to nowa barwa i głębokość zastępują stare wartości.
- Nie jest potrzebne żadne wstępne sortowanie ani porównanie typu obiekt-obiekt.
- Cały proces to nic więcej jak szukanie największej wartości Z_i dla każdego zbioru par

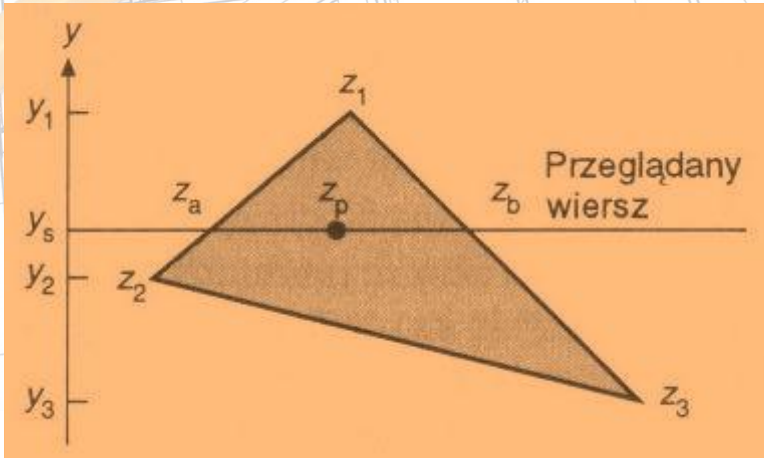
$$\{Z_i(x,y), F(x,y)\}$$

dla ustalonych x i y.

- W pamięci obrazu i w *z*-buforze są zapisywane informacje związane z największym ***z*** napotkanym dotychczas dla każdego (x, y).

Algorytm *z*-bufora

- Podobne obliczenie przyrostowe można wykonać w celu obliczenia pierwszej wartości *z* w następnym przeglądanym wierszu, zmniejszając ***z*** o B/C dla każdego Δy.
- W przypadku gdy płaszczyzna nie została określona albo gdy wielokąt nie jest płaski, to wówczas ***z***(*x*, *y*) można określić interpolując współrzędne wierzchołków wielokąta wzdłuż par krawędzi, a potem wzdłuż każdego przeglądanego wiersza, jak na rysunku.



$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

Algorytm z-bufora

- Algorytm z-bufora nie wymaga, żeby obiekty były wielokątami.
- Rzeczywiście jedną z dominujących zalet tej metody jest to, że może być używana do renderingu dowolnego obiektu, jeżeli dla każdego punktu rzutu można obliczyć barwę i wartość z
- Nie trzeba pisać żadnego bezpośredniego algorytmu przecinania.
- Algorytm z-bufora wykonuje sortowanie wagowe względem ***x*** i ***y*** bez żadnych porównań, a przy sortowaniu względem ***z*** jest potrzebne tylko jedno porównanie na piksel dla każdego wielokąta zawierającego ten piksel.
- Czas zajmowany przez obliczenia związane z wyznaczaniem widocznych powierzchni jest w przybliżeniu niezależny od liczby wielokątów w obiektach, ponieważ średnio liczba pikseli pokrytych przez każdy wielokąt zmniejsza się wraz ze wzrostem liczby wielokątów w bryle widzenia.
- Średnia wielkość zbioru przeglądanych par zmierza do wartości stałej.

Algorytm z-bufora

- Chociaż algorytm z-bufora wymaga dużej pamięci dla z-bufora, jest łatwy do implementacji.
- Jeżeli jest problem z pamięcią, to obraz może być przeglądany pasami.
- Ze względu na prostotę z-bufora i brak dodatkowych struktur danych, zmniejszające się koszty pamięci zainspirowały wiele sprzętowych implementacji z-bufora.
- Ponieważ algorytm z-bufora działa z precyzją obrazową, mogą się pojawiać zakłócenia.
- Ten problem jest rozwiązywany za pomocą algorytmu z-bufora, gdzie wykorzystuje się dyskretną aproksymację do bezwagowego próbkowania powierzchni.
- Z-bufor jest często implementowany sprzętowo dla liczb całkowitych od **16**- do **32**-bitowych, realizacje programowe natomiast (i niektóre sprzętowe) mogą wykorzystywać wartości zmiennopozycyjne.

Algorytm z-bufora

- 16-bitowy z-bufor daje wystarczający zakres dla wielu zastosowań
- Dla **CAD/CAM**, 16 bitów zapewnia zbyt małą precyzję reprezentowania środowisk, w których:
  - obiekty zdefiniowane z dokładnością milimetrową są umieszczane w odległościach mierzonych w kilometrach.
- Sytuacja je jeszcze gorsza, jeżeli jest wykorzystywane rzutowanie perspektywiczne – kompresja odległych wartości **z** wynikająca z dzielenia perspektywicznego ma istotny wpływ na porządkowanie głębokości i przecięcia odległych obiektów.
- Dwa punkty umieszczone blisko rzutni po przekształceni mogą dać dwie różne wartości całkowite, natomiast gdy są daleko od rzutni, mogą dać taką samą wartość.

Algorytm przeglądania

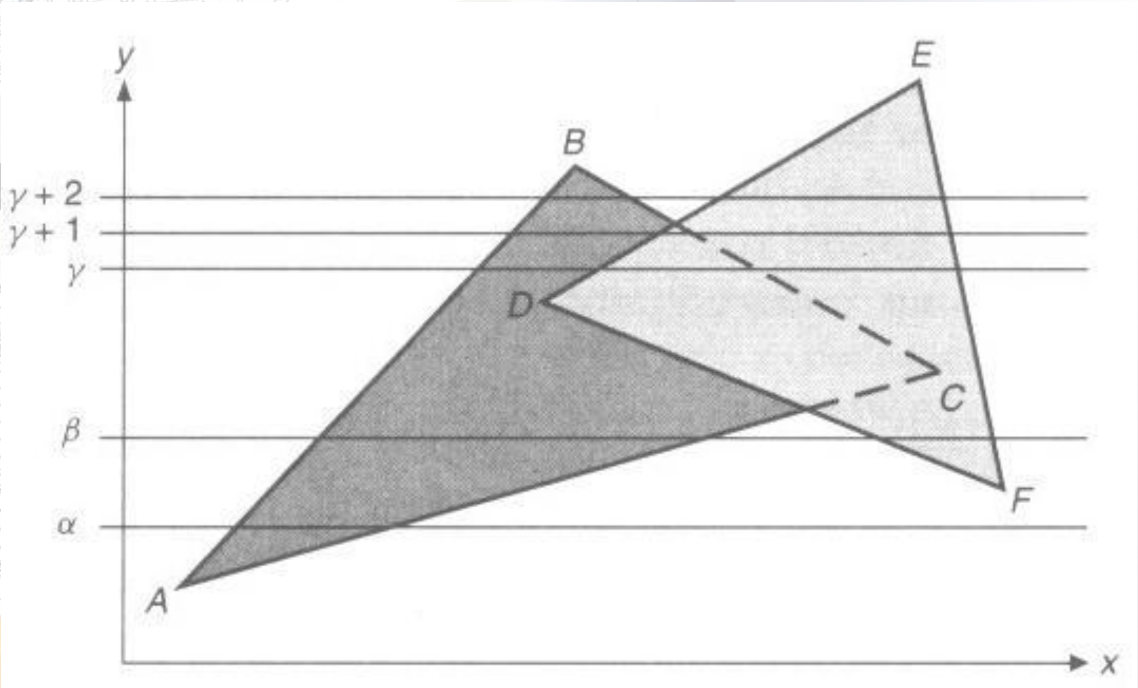
Algorytm przeglądania

- Algorytmy przeglądania, opracowane pierwotnie przez Wylie'a, Romneya, Evansa i Erdahla, Bouknighta i Watkinsa, działają z precyzją obrazową i tworzą obraz jednego wiersza na raz.
- Podstawowa koncepcja polega na rozszerzeniu algorytmu konwersji wierszowej oraz wykorzystaniu różnych form spójności, w tym spójności przeglądania wierszowego i spójności krawędziowej.
- Różnica polega na tym, że zajmujemy się nie jednym wielokątem, a kilkoma.
- W pierwszym kroku jest tworzona tablica krawędzi (ET) dla wszystkich nie-poziomych krawędzi wszystkich wielokątów zrzutowanych na rzutnię.
- Tak jak poprzednio krawędzie poziome są pomijane.
- Pozycje w tablicy ET są podzielone na grupy ze względu na najmniejszą współrzędną *y* każdej krawędzi, a w obrębie grupy są porządkowane ze względu na rosnącą współrzędną *x* ich niższego punktu końcowego.

Algorytm przeglądania

- Każda grupa zawiera:
  - współrzędną ***x*** końca o mniejszej współrzędnej *y*
  - współrzędną ***y*** drugiego końca krawędzi
  - przyrost ***x***, używany przy przejściu z jednego przeglądanego wiersza do następnego (jest odwrotnością nachylenia krawędzi)
  - numer identyfikacyjny wielokąta, wskazujący wielokąt, do którego należy krawędź
- Potrzebna jest również tablica wielokątów (PT), która - oprócz numeru identyfikacyjnego - zawiera przynajmniej następujące informacje o każdym wielokącie:
  - współczynniki równania płaszczyzny.
  - informację o cieniowaniu albo barwie wielokąta.
  - wskaźnik boolowski we-wy wykorzystywany w przeglądaniu wiersza ustawiany początkowo na wartość *false*.

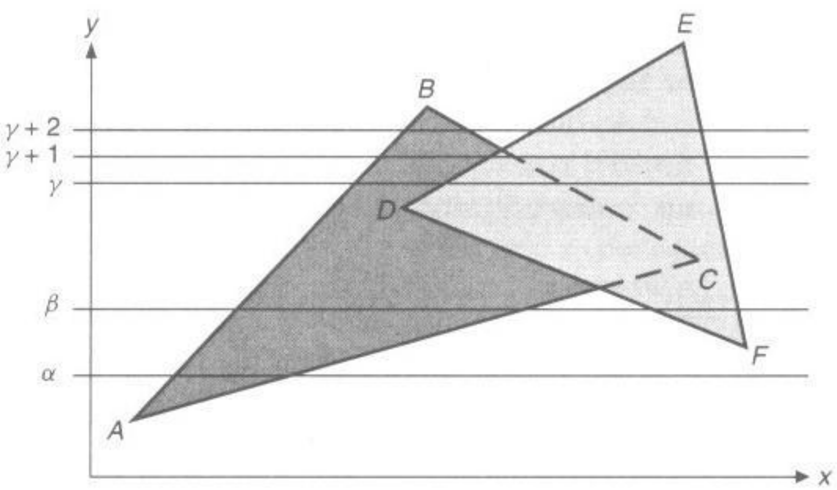
Algorytm przeglądania



- Na rysunku pokazano rzut dwóch trójkątów na płaszczyznę (x, y)
- Ukryte krawędzie są pokazane liniami przerywanymi.
- Uporządkowana tablica ET dla tej figury zawiera pozycje dla AB, AC, FD, FE, CB i DE.
- Tablica PT ma pozycje dla ABC i DEF.

Algorytm przeglądania

- **Tablica krawędzi aktywnych** (AET) jest również potrzebna.
- Jest ona zawsze utrzymywana w porządku wynikającym z **rosnących** wartości ***x***.
- Na dolnym rysunku pokazano pozycje tablic **ET**, **PT** i **AET**.
- W chwili, gdy algorytm **dochodzi** do **przeglądania** wierszy, tablica AET zawiera *AB* i *AC* (w tej kolejności).



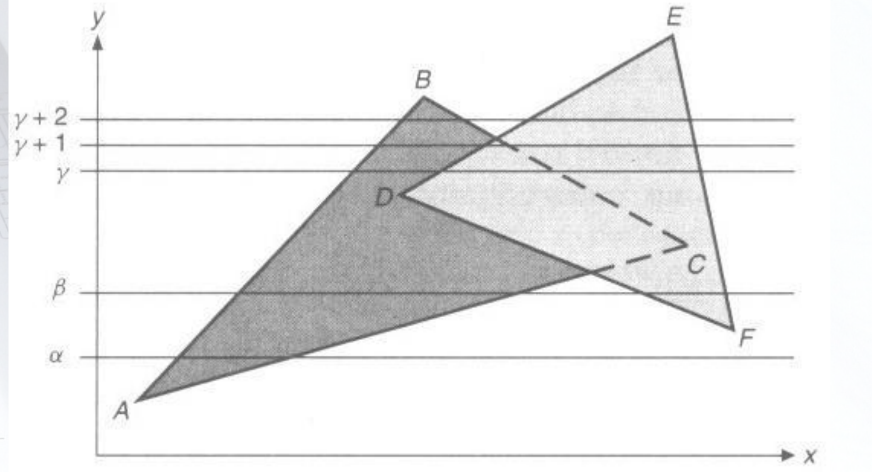
Pozycja ET					
	x	y_{max}	Δx	ID	•

Pozycja PT				
	ID	Równanie płaszczyzny	Informacja o cieniowaniu	We-wy

Zawartość AET

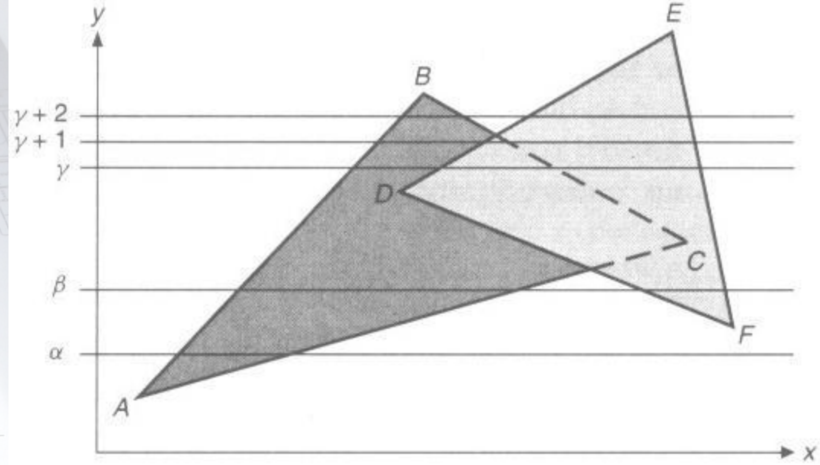
Przeglądany wiersz	Pozycje			
α	*AB*	*AC*		
β	*AB*	*AC*	*FD*	*FE*
$\gamma, \gamma + 1$	*AB*	*DE*	*CB*	*FE*
$\gamma + 2$	*AB*	*CB*	*DE*	*FE*

Algorytm przeglądania



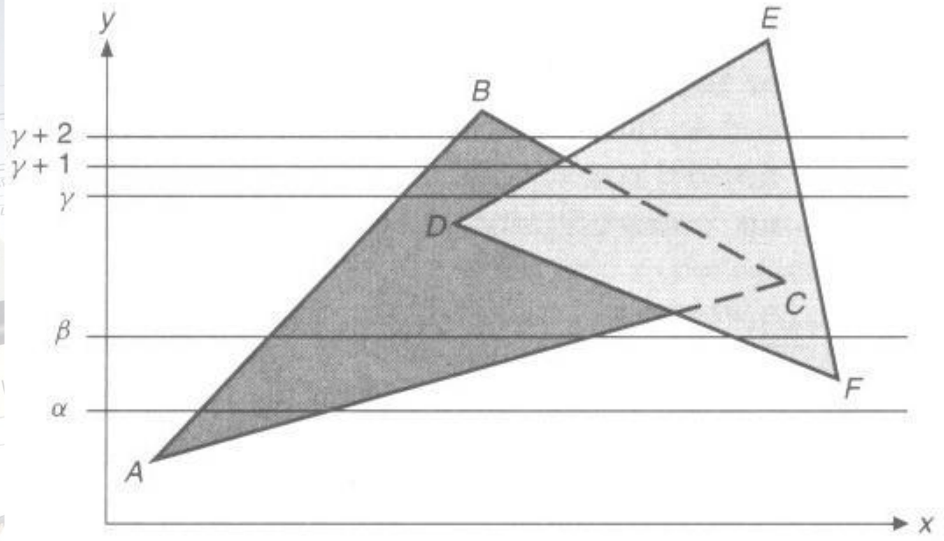
- Krawędzie są przetwarzane od lewa do prawa.
- W celu przetworzenia *AB* najpierw został zanegowany wskaźnik we-wy wielokąta *ABC.*
- W omawianym przypadku wskaźnik przyjmuje wartość true i wobec tego segment znajduje się wewnątrz wielokąta i wielokąt musi być rozważany.
- Ponieważ segment jest tylko w jednym wielokącie *ABC,* musi on być **widoczny** i trzeba go pocieniować od krawędzi *AB* do następnej krawędzi w tablicy AET, czyli krawędzi *AC.*
- Jest to przypadek spójności segmentu.
- Na krawędzi *AC* wskaźnik dla *ABC* jest zamieniany na *false* i segment już nie jest wewnątrz żadnego wielokąta.

Algorytm przeglądania



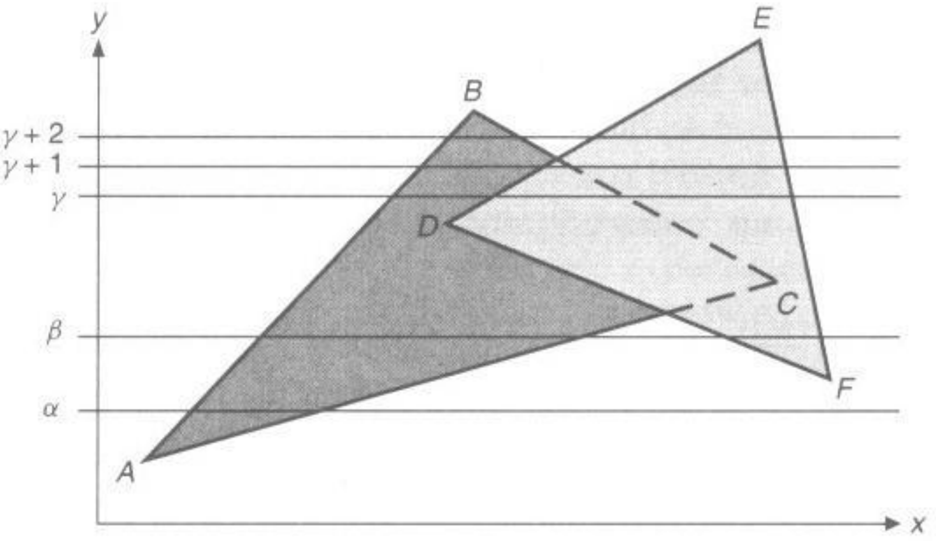
- Następnie, ponieważ *AC* jest ostatnią krawędzią w AET, proces przeglądania wiersza zostaje zakończony.
- Tablica AET jest uaktualniana na podstawie tablicy ET i ponownie porządkowana ze względu na ***x***, ponieważ niektóre z jej krawędzi mogłyby się przeciąć
- Teraz jest przetwarzany następny wiersz.

Algorytm przeglądania



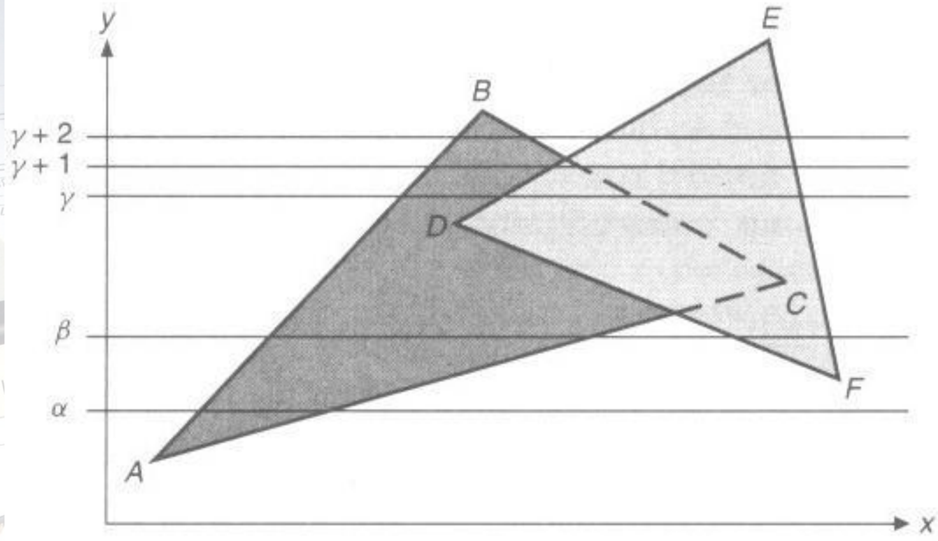
- Gdy dojdzie się do przeglądania wiersza β, w tablicy AET jest następująca kolejność: *AB, AC, FD, FE*.
- Przetwarzanie odbywa się mniej więcej tak samo jak poprzednio.
- W jednym wierszu są dwa wielokąty, ale każdy segment jest tylko w jednym wielokącie.

Algorytm przeglądania



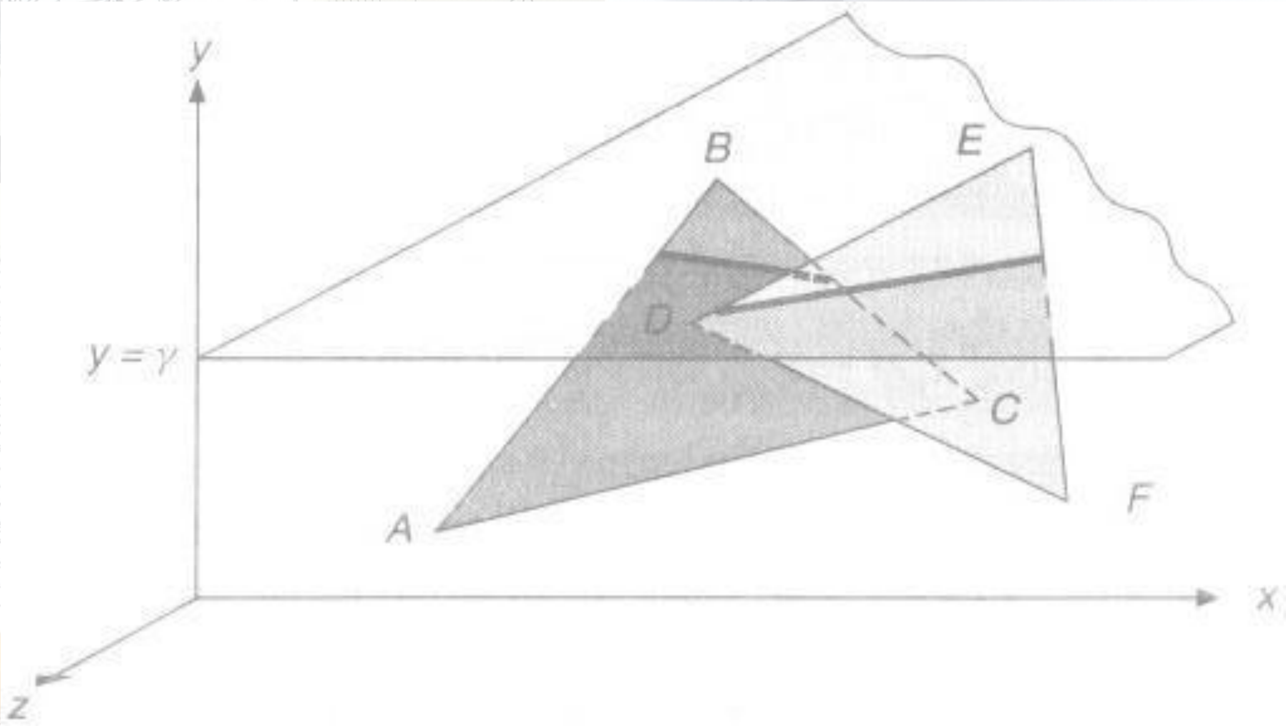
- Znacznie ciekawsza sytuacja jest dla $y = \gamma$.
- Wejście do *ABC* powoduje, że jego wskaźnik przyjmuje wartość *true*.
- Barwa *ABC* jest używana dla segmentu aż do następnej krawędzi *DE*.
- W tym miejscu wskaźnik dla *DEF* również przyjmuje wartość *true* i segment jest w dwóch wielokątach.
- Musimy teraz zdecydować, czy *ABC* czy *DEF* jest bliżej obserwatora

Algorytm przeglądania



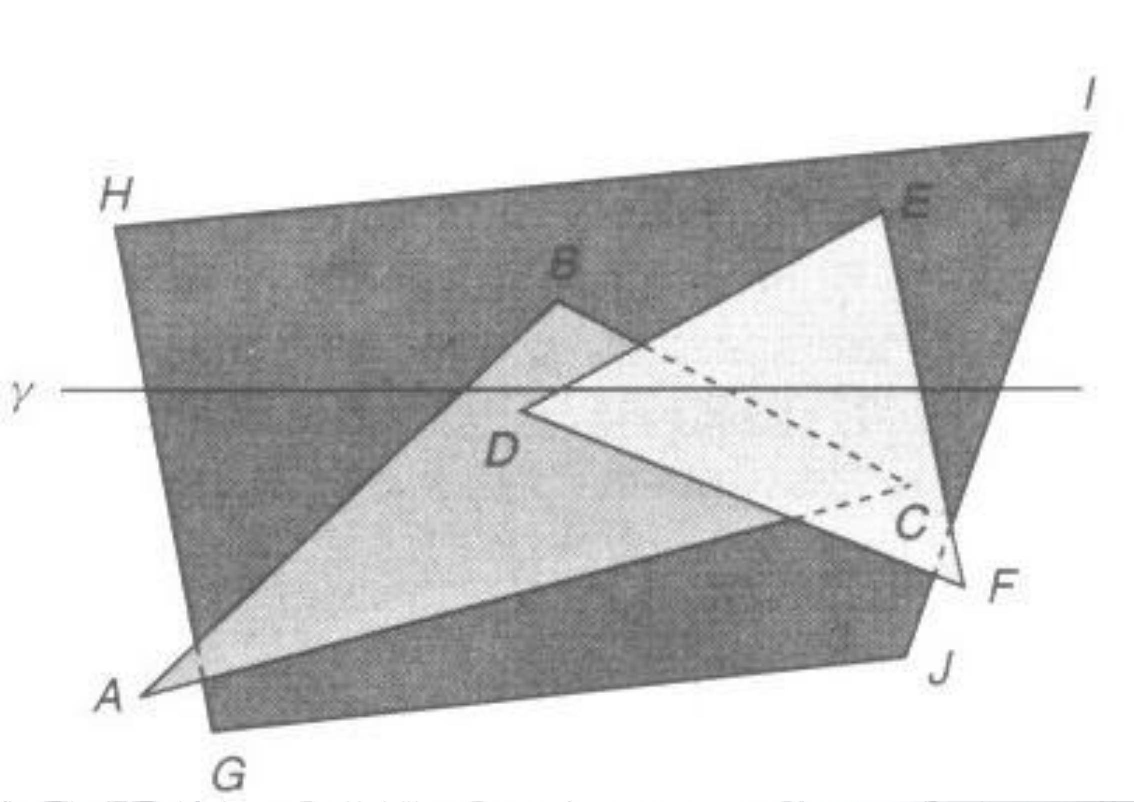
- W tym celu obliczamy równania płaszczyzn obu wielokątów dla ***z*** przy $y = \gamma$ dla *x* przecięcia wiersza z krawędzią *DE.*
- Ta wartość ***x*** znajduje się w tablicy AET w pozycji dla *DE.*
- W omawianym przykładzie ***z*** jest większe dla *DEF*, dlatego wielokąt ten jest widoczny.
- Zakładając, że wielokąty nie przecinają się, segmentowi przypisuje się barwę taką jak w trójkącie *DEF* aż do krawędzi *CB*, gdzie wskaźnik trójkąta *ABC* przyjmuje wartość *false*
- Segment jest znowu tylko w jednym wielokącie *DEF,* którego barwa jest przypisana segmentowi aż do krawędzi *FE.*

Algorytm przeglądania



- Na rysunku pokazano wzajemne położenie dwóch wielokątów i płaszczyzny – dwie grube linie wskazują przecięcia wielokątów z płaszczyzną.

Algorytm przeglądania

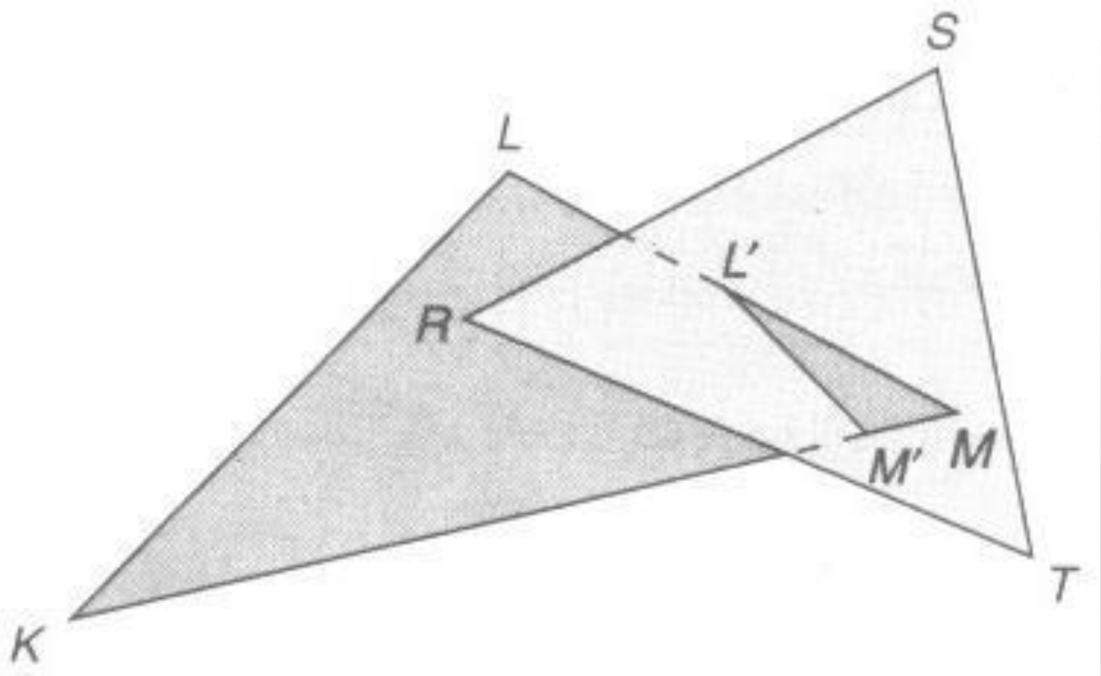


- Załóżmy, że zarówno za *ABC*, jak i za *DEF* jest duży wielokąt *GHIJ* .
- Jeżeli przeglądając wiersz dojdziemy do krawędzi *CB*, to segment będzie wciąż w wielokątach *DEF* i *GHIJ* i obliczenia głębokości są wykonywane ponownie.

Algorytm przeglądania

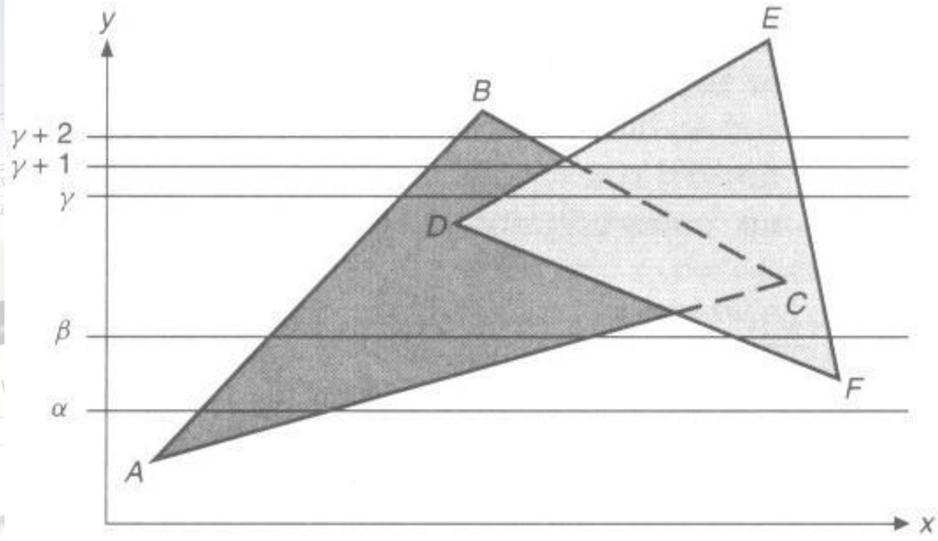
- Takich obliczeń można jednak uniknąć, jeżeli założymy, że żaden z wielokątów nie przecina drugiego.
- To założenie oznacza, że gdy segment opuszcza *ABC*, wówczas relacja głębokości między *DEF i GHIJ* nie może się zmienić i *DEF* jest w dalszym ciągu z przodu.
- Dlatego obliczanie głębokości nie jest konieczne wówczas, gdy przy przeglądaniu wychodzi się poza zasłonięty wielokąt, a jest potrzebne tylko wówczas, gdy wychodzi się poza zasłaniający wielokąt.

Algorytm przeglądania



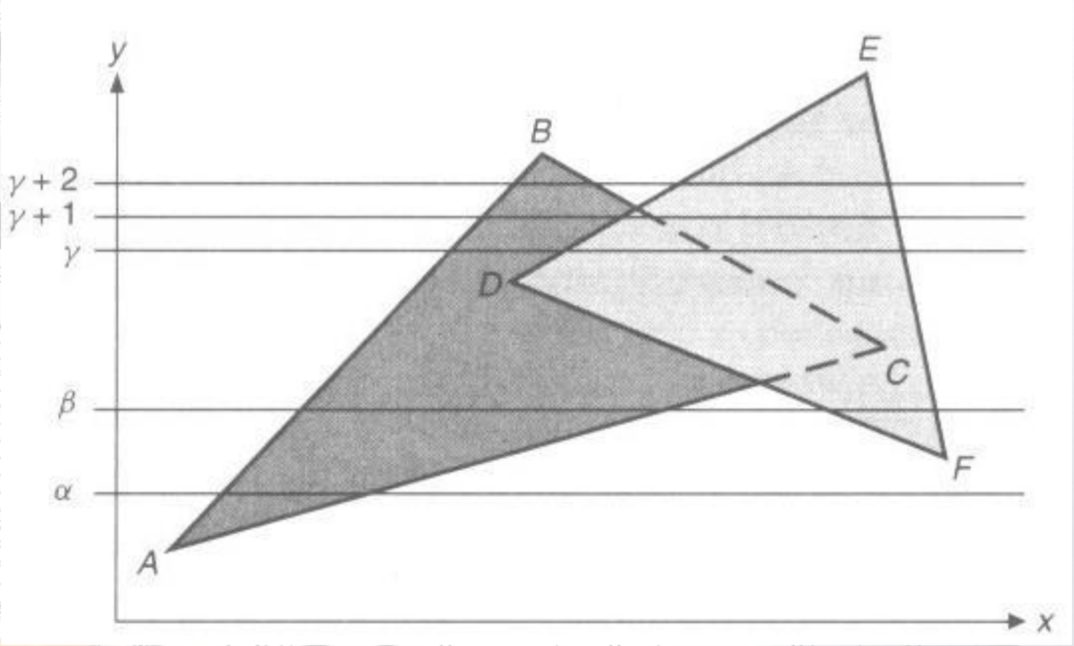
- W celu poprawnego wykorzystania algorytmu dla przecinających się wielokątów z rysunku dzielimy KLM na KLL'M' i L'M’M, wprowadzając **fałszywą krawędź** M'L'.
- Można też tak zmodyfikować algorytm, żeby znajdować punkt przecięcia w przeglądanym wierszu w trakcie przetwarzania przeglądanego wiersza.

Algorytm przeglądania



- W innej modyfikacji tego algorytmu wykorzystuje się **spójność głębokości**.
- Załóżmy, że wielokąty wzajemnie nie przecinają się.
- Jeżeli dla przeglądanego wiersza w AET są te same krawędzie co w poprzednim przeglądanym wierszu i na dodatek w tej samej kolejności, to nie pojawia się żadna zmiana relacji głębokości w żadnym segmencie przeglądanego wiersza
- Nie są wtedy potrzebne nowe obliczenia związane z głębokością
- Wobec tego widoczny segmentu w poprzednio przeglądanym wierszu określa segment w bieżącym wierszu.

Algorytm przeglądania



- Tak jest w przypadku wierszy $y = \gamma$ i $y = \gamma+1$ na rysunku, dla których segmenty od *AB* do *DE* i od *DE* do *FE* są widoczne.
- Spójność głębokości jest jednak na tym rysunku tracona, gdy przejdziemy z wiersza $y = \gamma+1$ do $y = \gamma+2$, ponieważ krawędzie *DE* i *CB* zmieniają kolejność w AET (sytuacja, w której algorytm musi się przystosować).

Algorytm przeglądania

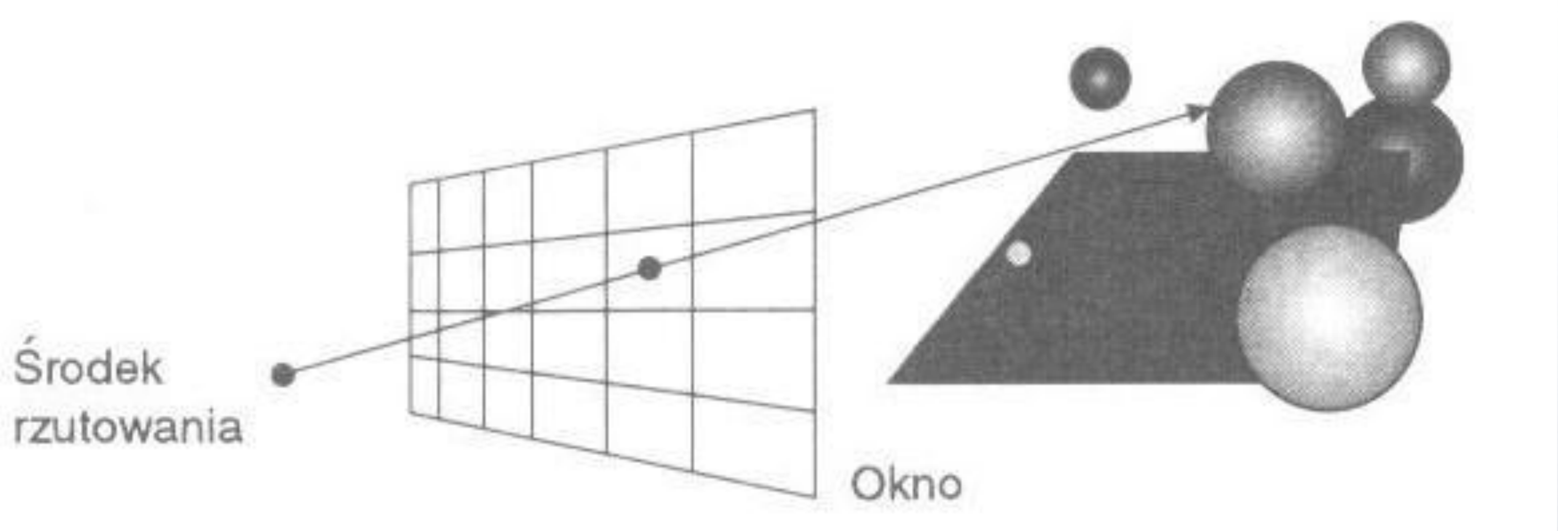
- Nie zastanawialiśmy się dotychczas, jak traktować tło.
  - Najprostszy sposób polega na tym, żeby na początku wpisać do bufora obrazu barwę tła, tak żeby algorytm musiał tylko przetwarzać wiersze, które przecinają krawędzie.
  - Inny sposób polega na włączeniu do definicji sceny dostatecznie dużego wielokąta, który jest dalej z tyłu niż wszystkie inne, jest równoległy do rzutni i jest odpowiednio cieniowany.
  - Wreszcie można tak zmodyfikować algorytm, żeby barwa tła była umieszczana bezpośrednio w pamięci obrazu zawsze wówczas, gdy przeglądany punkt nie jest w żadnym wielokącie.

Metoda śledzenia promieni

Wyznaczanie powierzchni widocznych metodą śledzenia promieni

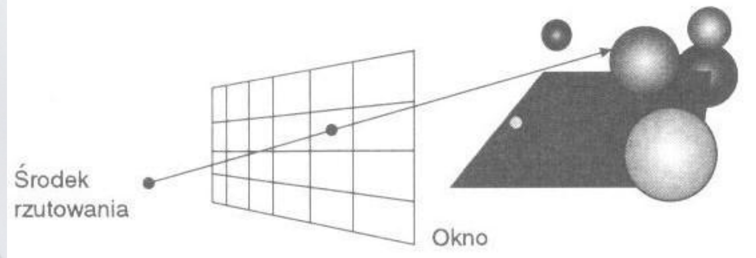
- Metoda śledzenia promieni określa widoczność powierzchni na zasadzie śledzenia umownych promieni światła od oka obserwatora do obiektów sceny.
- Jest to dokładnie prototypowy algorytm z precyzją obrazu.
- Jest wybierany:
  - środek rzutowania (oko obserwatora) oraz
  - pole wizualizacji na dowolnej rzutni.
- Można sobie wyobrazić, że pole wizualizacji jest podzielone regularną siatką o wymaganej rozdzielczości.
- Elementy siatki odpowiadają pikselom.

Wyznaczanie powierzchni widocznych metodą śledzenia promieni



- Dla każdego piksela w polu wizualizacji, ze środka rzutowania jest wyprowadzany promień przez środek piksela w kierunku sceny
- Barwa piksela jest ustawiana zgodnie z barwą obiektu dla **najbliższego punktu przecięcia**.

Metoda śledzenia promieni



- Metoda śledzenia promieni była pierwotnie opracowana przez Appela oraz Goldsteina i Nagela.
- Appel wykorzystywał rzadką siatkę promieni w celu wyznaczenia cieniowania, z uwzględnieniem sprawdzania, czy punkt znajduje się w cieniu.
- Goldstein i Nagel wykorzystywali swój algorytm do symulowania trajektorii torów pocisków i cząstek nuklearnych – dopiero później zastosowali go do grafiki.
- Whitted i Kay rozszerzyli metodę śledzenia promieni tak, żeby było możliwe uwzględnianie odbić zwierciadlanych i załamań.
- Omawiane zagadnienie łączy się z problemem **cieni, odbić i załamań**, a więc efektów, z rozwiązywania których metoda śledzenia promieni jest najbardziej znana
- Tutaj zajmiemy się metodą śledzenia promieni tylko jako algorytmem wyznaczania powierzchni widocznych.

Obliczanie przecięć

- Elementem kluczowym każdego programu śledzenia promieni jest zadanie wyznaczenia przecięcia obiektu z promieniem.
- W celu wykonania tego zadania korzystamy z takiej samej reprezentacji parametrycznej.
- Każdy punkt (x, y, z) wzdłuż promienia od (x_0, y_0, z_0)
- do (x_1, y_1, z_1) jest określany wartością parametru ***t*** taką, że:

$$x = x_0 + t(x_1 - x_0)$$

$$y = y_0 + t(y_1 - y_0)$$

$$z = z_0 + t(z_1 - z_0)$$

Obliczanie przecięć

- Dla wygody określamy wartości $\Delta x, \Delta y, \Delta z$ tak, że:

$$\Delta x = x_1 - x_0$$
$$\Delta y = y_1 - y_0$$
$$\Delta z = z_1 - z_0$$

- Wobec tego

$$y = y_0 + t\Delta y$$
$$x = x_0 + t\Delta x$$
$$z = z_0 + t\Delta z$$

- Jeżeli środek rzutowania jest w (x_0, y_0, z_0) a środek piksela w oknie jest w (x_1, y_1, z_1) to *t* zmienia się od 0 do 1 między tymi punktami.

Obliczanie przecięć

- Ujemne wartości *t* reprezentują punkty poza środkiem rzutowania, a wartości *t* większe niż 1 odpowiadają punktom po stronie rzutni dalszej od środka rzutowania.
- Dla każdego rodzaju obiektu musimy znaleźć reprezentację, która umożliwi określenie *t* na przecięciu obiektu z promieniem.
- Jednym z obiektów, dla których najłatwiej jest to zrobić, jest kula i stąd nadmiar kul obserwowanych w typowych obrazach obliczanych metodą śledzenia promieni!
- Kula o środku w (a, b, c) i promieniu *r* może być reprezentowana za pomocą równania

$$(x-a)^2 + (y-b)^2 (z-c)^2 = r^2$$

Obliczanie przecięć

- Rozpisując równanie, podstawiając za *x*, *y* i *z* wartości przyrostowe i porządkując równanie otrzymujemy równanie kwadratowe ze względu na t:

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 - 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$$

- Współczynniki równania są wyrażone w zależności od stałych otrzymanych z równania kuli i równania promienia
- Jeżeli nie ma pierwiastków rzeczywistych, to kula i promień nie przecinają się
- Jeżeli jest tylko jeden pierwiastek rzeczywisty, to promień jest styczny do kuli.
- W przeciwnym przypadku dwa pierwiastki określają punkty przecięcia z kulą ten, dla którego wartość *t* jest mniejsza, jest bliższy.

Obliczanie przecięć

- Często musimy określić normalną do powierzchni w punkcie przecięcia po to, żeby określić barwę powierzchni.
- Jest to szczególnie łatwe w przypadku kuli, ponieważ (nieznormalizowana) normalna jest wektorem od środka do punktu przecięcia
- Kula o środku (a, b, c) ma normalną do powierzchni

$$((x-a)/r, (j-b)/r, (z-c)/r)$$

- w punkcie przecięcia (x, y, z) .

Obliczanie przecięć

- Znajdowanie przecięcia promienia z wielokątem jest nieco trudniejsze.
- W celu wyznaczenia punktu przecięcia promienia z wielokątem:
  - najpierw sprawdzamy, czy promień przecina płaszczyznę wielokąta
  - następnie czy punkt przecięcia leży w wielokącie.
- Ponieważ równanie płaszczyzny ma postać

$$Ax + By + Cz + D = 0$$

- po podstawieniu zależności przyrostowych otrzymujemy:

$$A(x_0 + t\Delta x) + B(y_0 + t\Delta y) + C(z_0 + t\Delta z) + D = 0$$

-

Obliczanie przecięć

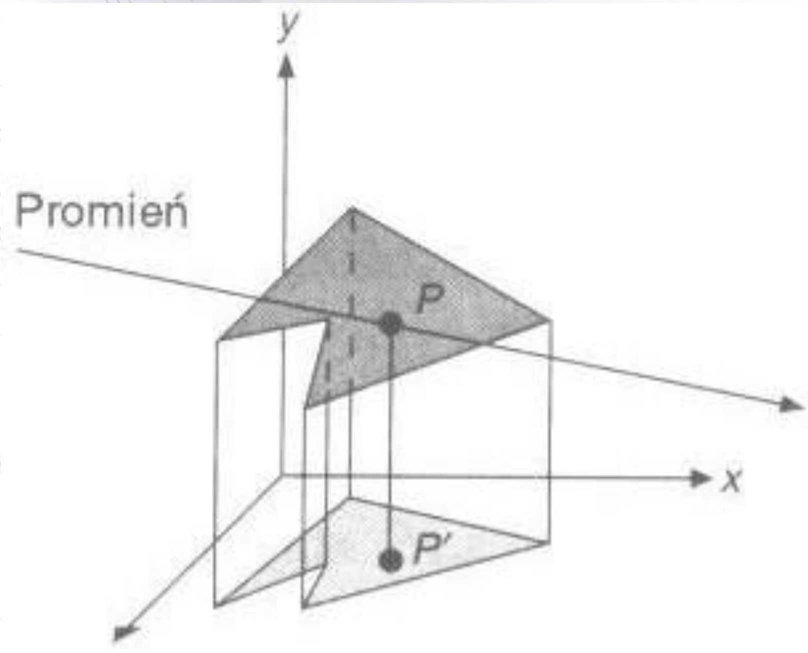
- Po uporządkowaniu ze względu na parametr

$$t(A\Delta x + B\Delta y + C\Delta z) + (Ax_0 + By_0 + Cz_0 + D) = 0$$

- Można policzyć jego wartość:

$$t = \frac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}$$

- Jeżeli mianownik równania jest równy 0, to promień i płaszczyzna są równoległe i nie przecinają się.
- Łatwy sposób sprawdzenia, czy punkt przecięcia leży wewnątrz wielokąta, polega na zrzutowaniu wielokąta i punktu prostopadle na jedną z trzech płaszczyzn układu współrzędnych jak na rysunku.



Efektywność metody śledzenia promieni dla powierzchni widocznych

- Algorytm *z*-bufora, korzystając ze spójności, przetwarza dla każdego piksela informację tylko o tych obiektach, których rzut znajduje się w danym pikselu.
- W omówionej, prostej, ale drogiej wersji algorytmu wyznaczania powierzchni widocznych metodą śledzenia promieni każdy promień wychodzący z oka jest przecinany z każdym obiektem w scenie.
- Obraz o rozdzielczości 1024 x 1024 zawierający 100 obiektów wymagałby więc wykonania 100 mln. przecięć.
- Nie jest zaskoczeniem, że dla typowych scen 75-95% i więcej czasu systemu byłoby zużywane na wyznaczanie przecięć.
- Konsekwencją tego jest to, że omawiane dalej podejścia do poprawy efektywności metody śledzenia przy wyznaczaniu powierzchni widocznych dążą do przyspieszenia obliczania jednego przecięcia albo w ogóle do unikania takich obliczeń.

Optymalizacja obliczeń związanych z wyznaczaniem przecięć

- Wiele z elementów w równaniach do wyznaczania przecięć zawiera wyrażenia, które są stałe albo w całym obrazie, albo dla określonego promienia.
- Elementy te mogą być obliczone wcześniej, tak jak na przykład rzut prostokątny wielokąta na płaszczyznę.
- Przy odpowiedniej analizie można opracować szybkie metody znajdowania przecięć – nawet prosta zależność przecięcia z kulą może być ulepszona.
- Jeżeli dokona się takiego przekształcenia promieni, żeby biegły wzdłuż osi ***z***, to to samo przekształcenie może być użyte do każdego obiektu i każde przecięcie występuje dla $x = y = 0$.

Optymalizacja obliczeń związanych z wyznaczaniem przecięć

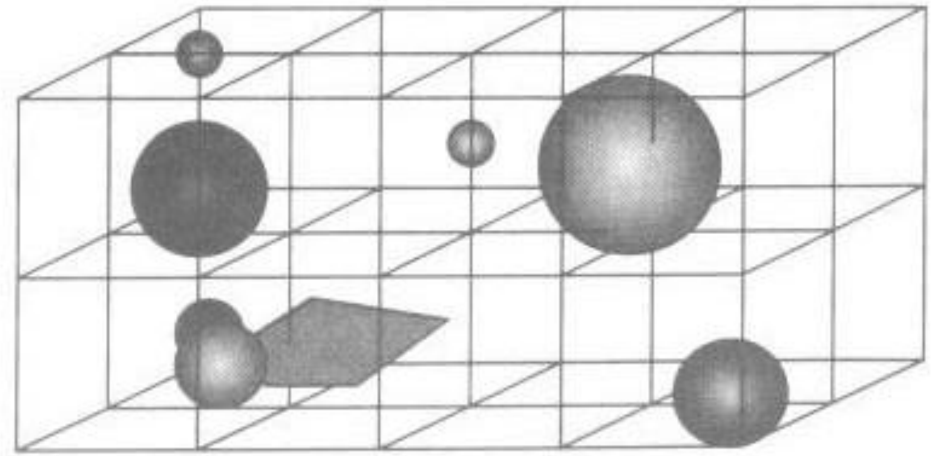
- Krok ten upraszcza obliczanie przecięć i umożliwia określanie najbliższego obiektu w wyniku sortowania względem z.
- Następnie za pomocą odwrotnego przekształcenia można punkt przecięcia przekształcić z powrotem i dalej wykonywać obliczenia związane z wyznaczaniem barwy.
- Bryły ograniczające umożliwiają w sposób szczególnie atrakcyjny zmniejszanie ilości czasu potrzebnego do wyznaczania przecięć.
- Obiekt, który jest względnie drogi, jeśli chodzi o wyznaczanie przecięć, może zostać otoczony przez bryłę ograniczającą, dla której taki test jest znacznie prostszy, na przykład przez:
  - kulę,
  - elipsoidę albo
  - prostopadłościan
- Jeżeli promień nie przecina bryły ograniczającej, to nie trzeba wykonywać testu dla obiektu.

Hierarchie

- Chociaż bryły ograniczające same nie określają kolejności albo częstotliwości testów przecięć, mogą być zorganizowane w struktury hierarchiczne z obiektami w liściach i wewnętrznymi węzłami, które otaczają swoich potomków.
- Mamy pewność, że bryła potomka nie jest przecinana przez promień, jeżeli nie jest przecinana bryła przodka.
- Dlatego jeżeli zaczniemy testy przecinania od korzenia, to możemy w trywialny sposób odrzucić wiele gałęzi tej hierarchii (a co za tym idzie wiele obiektów).

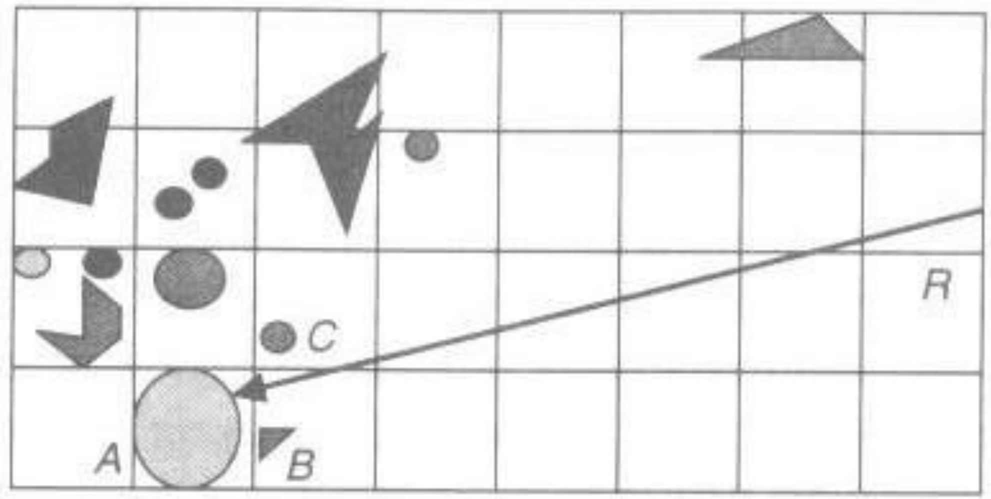
Algorytmy z podziałem przestrzennym

Podział przestrzenny



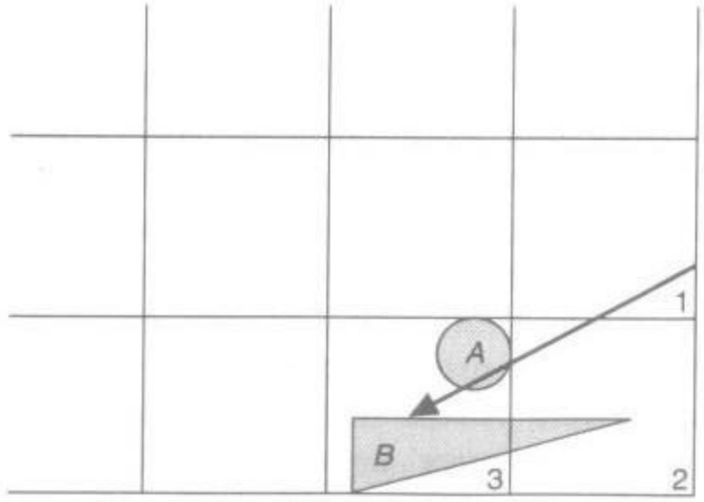
- Hierarchia brył otaczających organizuje obiekty w porządku wstępującym – podział przestrzenny natomiast dzieli przestrzeń w porządku zstępującym.
- Najpierw oblicza się prostopadłościan otaczający scenę.
- W jednym z podejść ten otaczający prostopadłościan jest dzielony na regularną siatkę jak na rysunku.
- Z każdym elementem podziału jest związana lista obiektów, które się w nim zawierają, całkowicie albo częściowo.
- Listy są wypełniane na zasadzie przypisania każdego obiektu do jednego lub więcej elementów podziału, do których ten obiekt należy.

Podział przestrzenny



- Teraz, jak pokazano na rysunku dla 2D, promień musi być przecięty tylko z tymi obiektami, które są zawarte w elementach podziału, przez które ten promień przechodzi.
- Ponadto elementy podziału trzeba sprawdzać w kolejności przechodzenia promienia
- Gdy tylko zostanie znaleziony element podziału, w którym jest przecięcie, nie trzeba sprawdzać dalszych elementów podziału.
- Zauważmy, że musimy uwzględnić wszystkie pozostałe obiekty w elemencie podziału po to, żeby sprawdzić, które przecięcie jest najbliżej.
- Ponieważ elementy podziału są utworzone przez regularną siatkę, każdy kolejny element podziału leżący wzdłuż promienia można obliczyć korzystając z wersji 3D algorytmu rysowania odcinka z punktem środkowym, zmodyfikowanego w taki sposób, żeby określać listę wszystkich elementów podziału, przez które promień przechodzi

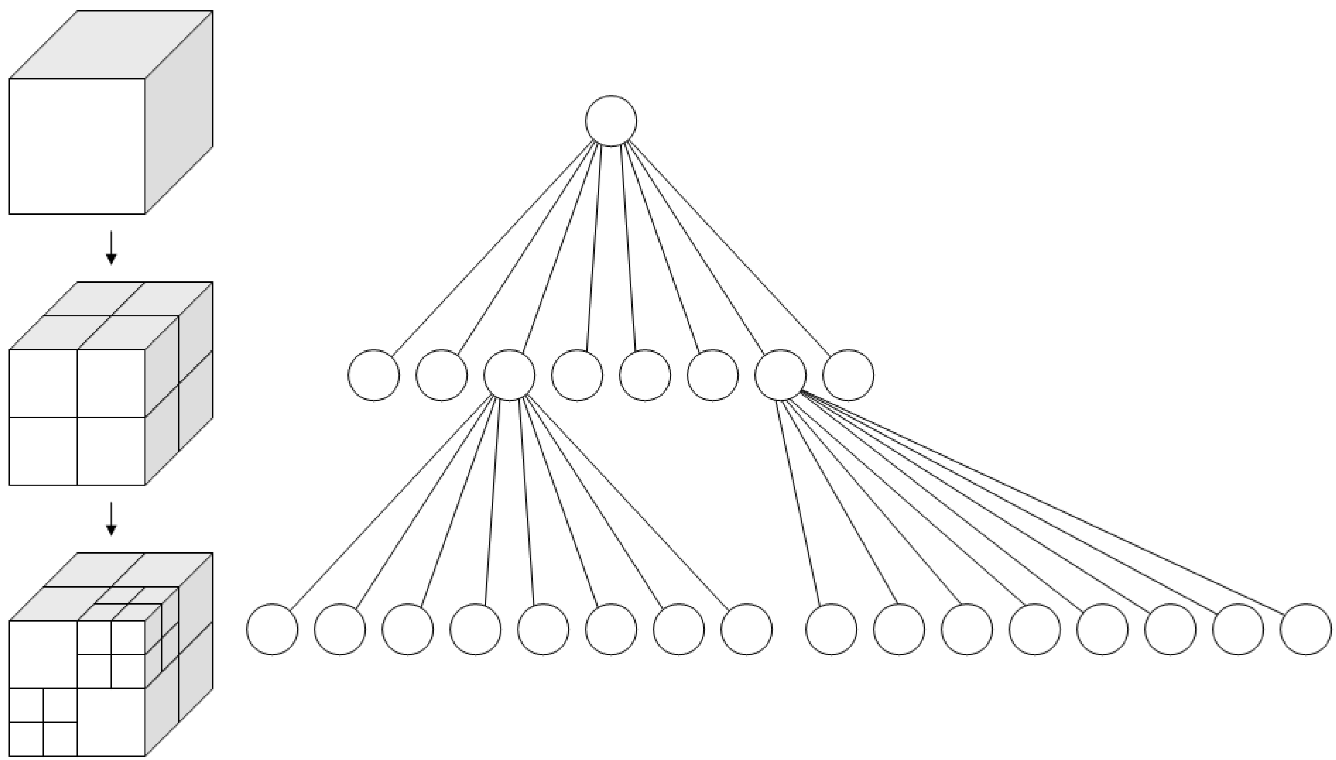
Podział przestrzenny



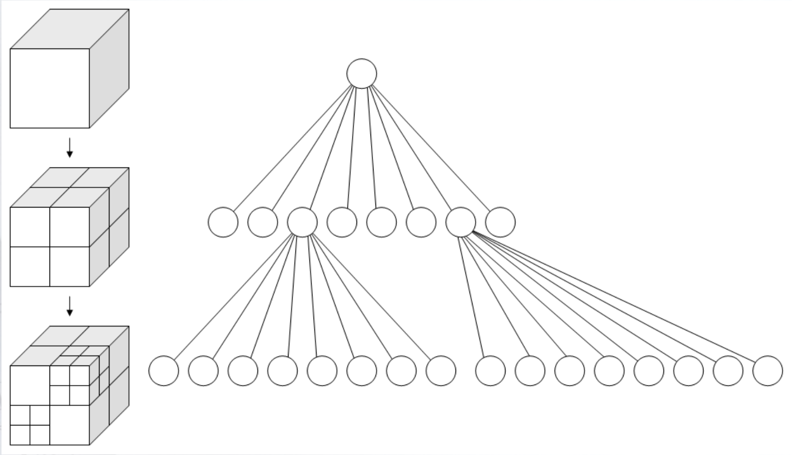
- Jeżeli promień przecina obiekt w danym elemencie podziału, to trzeba sprawdzić, czy samo przecięcie należy do elementu podziału
- Może się zdarzyć, że znalezione przecięcie jest już w innym elemencie podziału i że może istnieć bliższe przecięcie dla innego obiektu.
- Na przykład na rysunku obiekt *B* jest przecinany w elemencie podziału 3, chociaż został napotkany w elemencie podziału 2.
- Musimy kontynuować przeglądanie elementów podziału dopóty, dopóki nie znajdziemy przecięcia w bieżąco przeglądanym elemencie podziału, w tym przypadku *A* w elemencie podziału 3.
- W celu uniknięcia ponownego przeliczania przecięć promienia z obiektem, **który jest w kilku elementach podziału**, po pierwszym napotkaniu punkt przecięcia i numer identyfikacyjny mogą być wraz z obiektem zapamiętane w pomocniczej pamięci.

Podział przestrzenny

- Dippe i Swenson przedstawili adaptacyjny algorytm podziału, który tworzy elementy podziału o nierównych wielkościach.
- W innym adaptacyjnym podziale przestrzennym dokonuje się podziału sceny według metody drzewa ósemkowego.

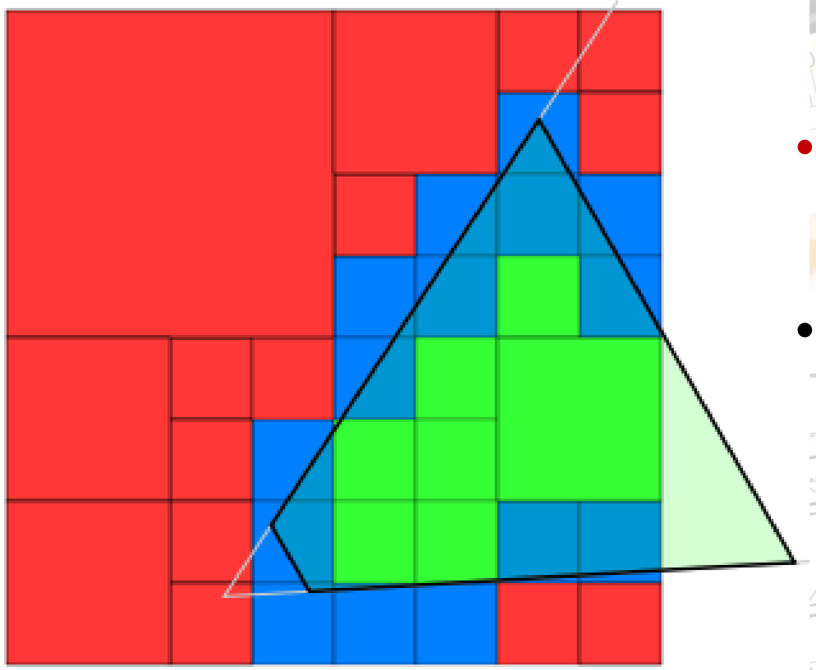


Podział przestrzenny



- W tym przypadku w celu określenia kolejnych elementów podziału leżących wzdłuż promienia można zastosować algorytm wyszukiwania sąsiadów w drzewie ósemkowym.
- Drzewa ósemkowe i inne hierarchiczne podziały przestrzenne mogą być traktowane jako specjalny przypadek hierarchii, w którym mamy pewność, ze potomkowie węzła wzajemnie nie przecinają się.
- Ponieważ takie podejście dopuszcza podział adaptacyjny, decyzja o dalszym podziale elementu podziału może zależeć od liczby obiektów w podpodziale albo od kosztu przecinania obiektów.

Drzewa ósemkowe



Na zielono zaznaczono sześciany w całości znajdujące się wewnątrz bryły widzenia, na czerwono znajdujące się poza nią, na niebiesko zawarte częściowo w bryle widzenia.

- Usprawniają odrzucanie niewidocznych obiektów, znajdujących się poza bryłą widzenia (view frustum culling) - można bardzo szybko odrzucać (lub akceptować) całe gałęzie drzewa bez zagłębianie się w nie, tj. bez dodatkowych, czasochłonnych testów.
- Testowanie widoczności ma charakter rekurencyjny i zaczyna się od korzenia drzewa ósemkowego.
- Sprawdzana jest widoczność sześcianu zapisanego w węźle:
  - Jeśli jest on niewidoczny, wiadome jest, że nie są widoczne także wszystkie obiekty, które się w nim znajdują
  - podobnie jeśli jest w całości widoczny - widoczne są także wszystkie obiekty w nim zawarte.
  - Jeśli natomiast widoczność sześcianu jest częściowa, przechodzi się w głąb drzewa i testuje widoczność jego ośmiu dzieci - mniejszych sześcianów.

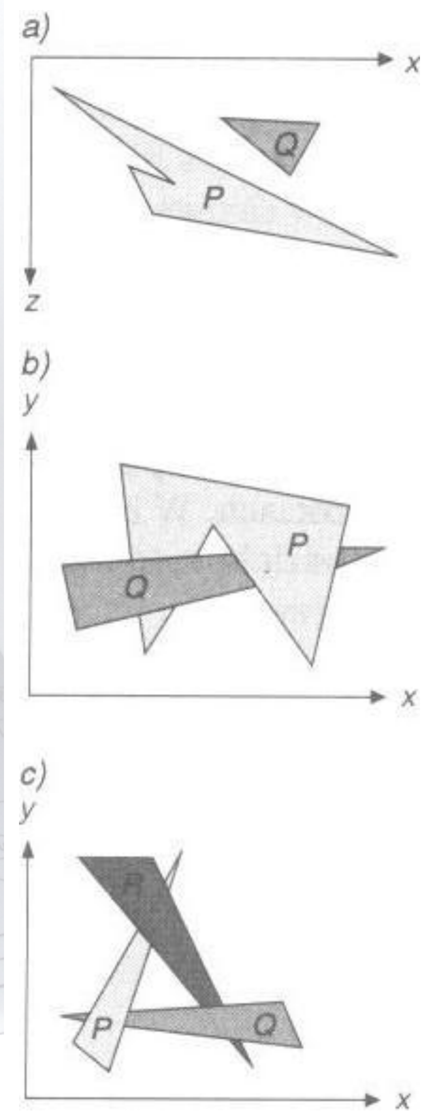
Inne rozwiązania

- Algorytmy z listą priorytetów
  - Algorytm sortowania ze względu na głębokość
  - Algorytm drzewa binarnego podziału przestrzeni
- Algorytmy podziału powierzchni
  - Algorytm Warnocka
- Algorytmy wyznaczania powierzchni krzywoliniowych

Algorytm z listą priorytetów

Algorytmy z listą priorytetów

- W *algorytmach z listą priorytetów* określa się kolejność wyświetlania obiektów, przy czym zapewnić należy, że jeżeli rendering obiektów będzie następował w tej kolejności, to uzyska się poprawne obrazy.
  - Na przykład, jeżeli żaden obiekt nie nakłada się na inny wzdłuż współrzędnej ***z***, to musimy jedynie wykonać segregowanie obiektów ze względu na rosnące wartości **z** i wykonywać rendering w tej kolejności.
  - Dalsze obiekty są przesłaniane przez bliższe obiekty, ponieważ piksele z bliższych wielokątów przykrywają te pochodzące z dalszych wielokątów.
- Jeżeli wystąpi nakładanie się obiektów względem współrzędnej ***z***, to nadal możemy określić poprawny porządek, tak jak na rysunku a.
- Jeżeli obiekty cyklicznie nakładają się tak jak na rysunku b i c albo przecinają się, to nie istnieje poprawna kolejność.
  - W takich przypadkach będzie konieczne podzielenie jednego lub więcej obiektów po to, żeby umożliwić uzyskanie porządku liniowego.



Algorytmy z listą priorytetów

- Algorytmy z listą priorytetów są hybrydami, które łączą operacje z precyzją obiektową i operacje z precyzją obrazową.
- Porównywanie głębokości i podział obiektów są wykonywane z precyzją obiektową.
- Jedynie konwersja wierszowa przy przeglądaniu, która zależy od zdolności urządzenia graficznego do zapisywania pikseli na miejscu pikseli należących do poprzednio narysowanych obiektów, jest wykonywana z precyzją obrazową.
- Ponieważ jednak lista posortowanych obiektów jest tworzona z precyzją obiektową, może być ponownie wyświetlona poprawnie z dowolną rozdzielczością.
- Algorytmy z listą priorytetową różnią się:
  - sposobem określania kolejności w wyniku sortowania,
  - sposobem wyboru obiektu do podziału
  - czasem dokonywania podziału

Algorytm sortowania ze względu na głębokość

Algorytm sortowania ze względu na głębokość

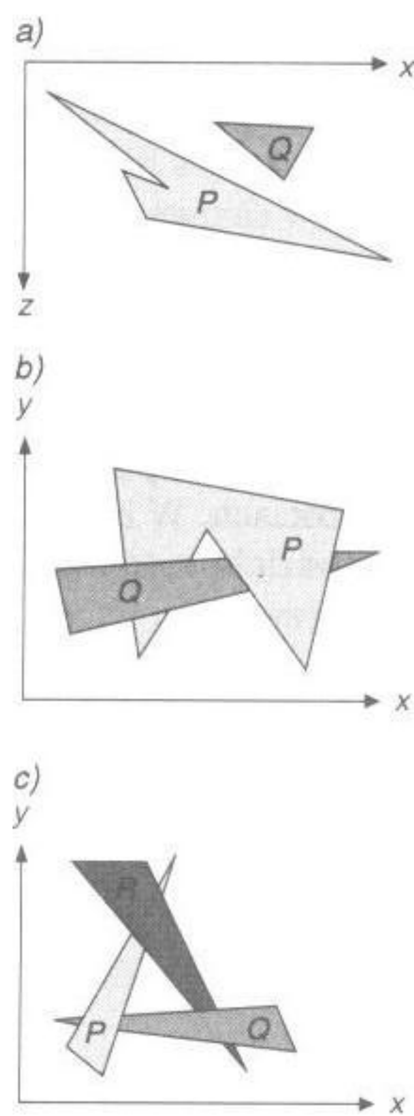
- Podstawowa idea algorytmu sortowania ze względu na głębokość, opracowanego przez Newella i Sancha, polega na rysowaniu wielokątów w pamięci obrazu w kolejności malejącej odległości od punktu obserwacji.
- Są wykonywane trzy koncepcyjne kroki:
  1. sortowanie wielokątów ze względu na najmniejszą (największą) współrzędną **z** każdego wielokąta
  2. rozstrzygnięcie wszystkich niejednoznaczności, które mogą powstawać w wyniku sortowania ze względu na nakładanie się przedziałów wartości z w wielokątach - w razie potrzeby wykonuje się podział wielokątów
  3. przeglądanie wierszami każdego wielokąta według rosnącej kolejności najmniejszej współrzędnej **z** (to znaczy od tyłu do przodu).

Algorytm sortowania ze względu na głębokość

- Rozważmy wykorzystanie bezpośredniego priorytetu.
- Bezpośredni priorytet zajmuje miejsce minimalnej wartości **z** i nie może być żadnych niejednoznaczności ze względu na głębokość
- O każdym priorytecie zakłada się, że może odpowiadać różnym płaszczyznom stałego ***z***.
- Uproszczona wersja algorytmu sortowania ze względu na głębokość jest często określana jako ***algorytm malarski***, z uwagi na podobieństwo do tego, jak malarz może malować bliższe obiekty na wcześniej namalowanych bardziej odległych obiektach.
- Algorytm malarski może być wykorzystywany do sceny, w której wielokąty znajdują się na płaszczyznach o różnych ***z***, jeżeli posortuje się wielokąty ze względu na minimalną współrzędną **z** albo ze względu na współrzędną **z** ich środka ciężkości, z pominięciem kroku 2.
- Chociaż taki sposób umożliwia konstruowanie scen, w ogólnym przypadku nie ma gwarancji, że uzyska się poprawne uporządkowanie.

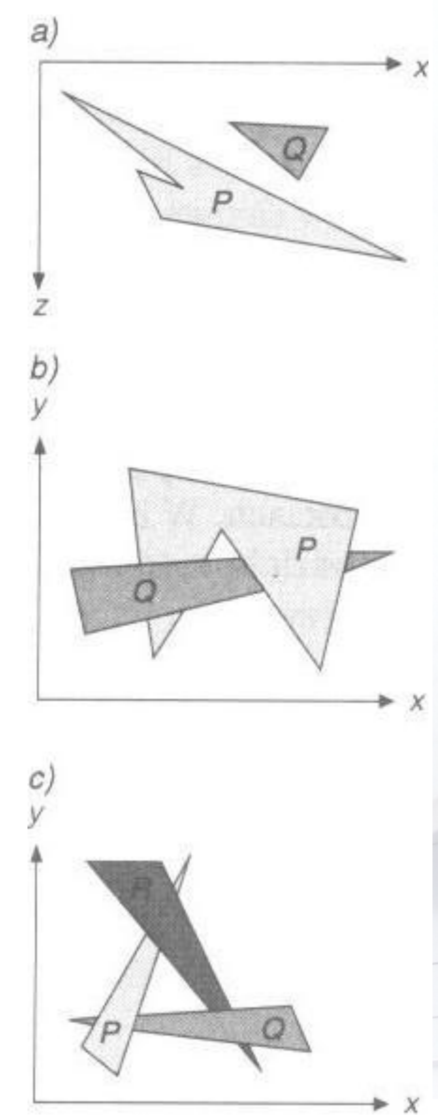
Algorytm sortowania ze względu na głębokość

- Na rysunku pokazano kilka rodzajów niejednoznaczności, które trzeba rozwiązywać w ramach kroku 2.
- Jak się to robi? Oznaczmy przez *P* wielokąt będący w danym momencie na końcu posortowanej listy.
- Zanim dokonamy rasteryzacji tego wielokąta do pamięci obrazu, trzeba go sprawdzić z każdym wielokątem *Q*, którego zakres wartości ***z*** przecina się z zakresem wartości ***z*** w wielokącie *P*
- Następuje sprawdzenie, że *P* nie zasłania *Q* i że *P* może wobec tego być zapisane przed *Q*.



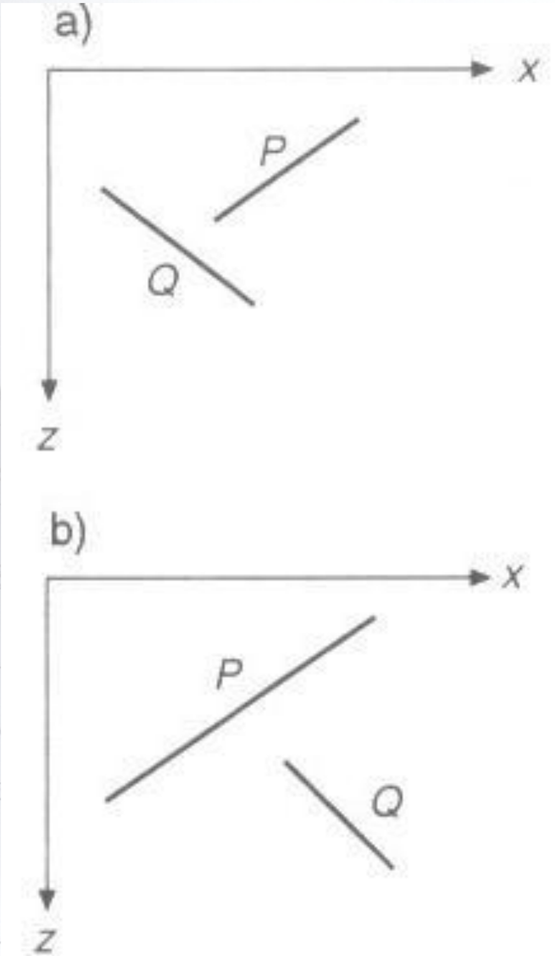
Algorytm sortowania ze względu na głębokość

- Wykonuje się do pięciu testów o rosnącej złożoności.
- Jeżeli jeden z tych testów uda się, to *P* nie zasłania *Q* i testuje się następny wielokąt *Q*, nakładający się z *P* względem współrzędnej ***z*.**
- Jeżeli wszystkie takie wielokąty przejdą przez testy, to wykonuje się rasteryzację wielokąta *P* i kolejny wielokąt na liście staje się nowym wielokątem *P*.



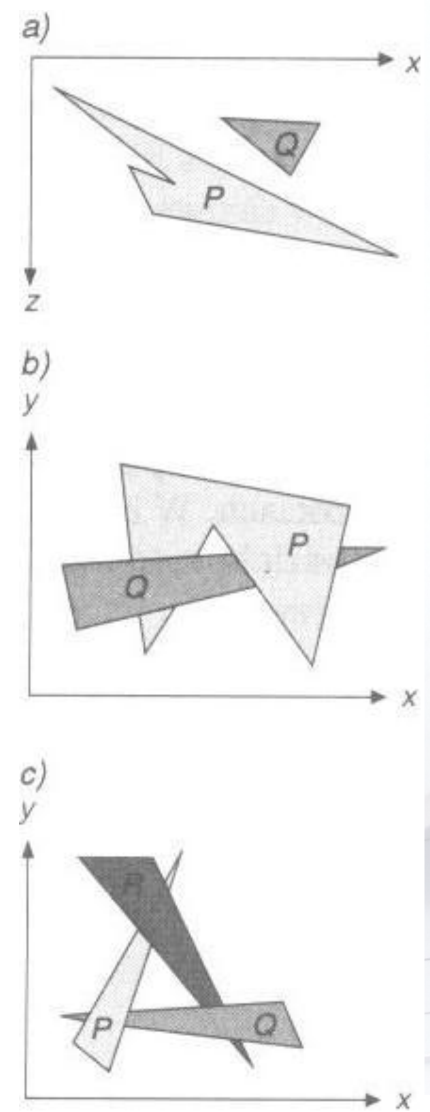
Algorytm sortowania ze względu na głębokość

- Pięć testów, o których była mowa, wygląda następująco:
  1. Czy zakresy współrzędnych **x** wielokątów nie przecinają się?
  2. Czy zakresy współrzędnych **y** wielokątów nie przecinają się?
  3. Czy P jest całkowicie po stronie płaszczyzny Q niewidocznej dla obserwatora, co jest prawdą dla rysunku a) z prawej strony
  4. Czy Q jest całkowicie po tej samej stronie płaszczyzny P co obserwator? (tak jak dla rysunku b)
  5. Czy rzuty wielokątów na płaszczyznę nakładają się? Można to określić porównując krawędzie jednego wielokąta z krawędziami drugiego.



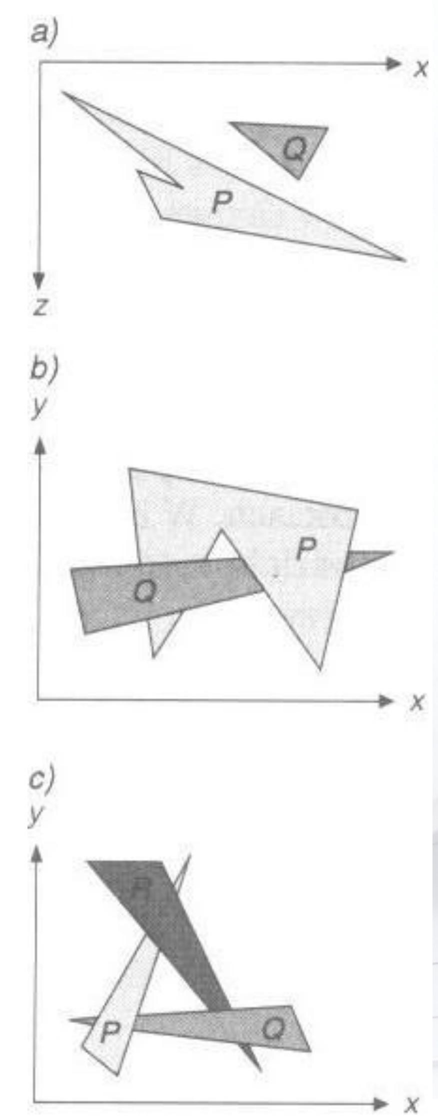
Algorytm sortowania ze względu na głębokość

- Jeżeli żaden z pięciu testów nie będzie spełniony, to zakładamy na moment, że *P* rzeczywiście zasłania *Q*, i sprawdzamy, czy można dokonać rasteryzacji *Q* przed *P.*
- Testów 1, 2 i 5 nie trzeba powtarzać, korzysta się natomiast z nowych wersji testów 3 i 4 z zamienionymi wielokątami:
  - 3'. Czy Q jest całkowicie po stronie płaszczyzny P niewidocznej dla obserwatora?
  - 4'. Czy P jest całkowicie po tej samej stronie płaszczyzny Q co obserwator?
- W przypadku z rysunku a) test 3' udaje się.
- Dlatego przesuwamy Q na koniec listy i staje się on nowym *P*.



Algorytm sortowania ze względu na głębokość

- W przypadku z rysunku b) testy w dalszym ciągu nie są rozstrzygające –nie istnieje kolejność, w jakiej można by dokonać poprawnej rasteryzacji.
- Dlatego trzeba dokonać podziału *P* albo *Q* płaszczyzną drugiego.
- Oryginalny wielokąt jest odrzucany, a jego części są umieszczane na liście we właściwej kolejności ze względu na ***z*** i algorytm jest wykonywany jak poprzednio.



Algorytm drzewa BSP

Algorytm drzewa binarnego podziału przestrzeni

- Algorytm drzewa binarnego podziału przestrzeni (BSP) był opracowany przez Fuchsa, Kedema i Naylora, którzy korzystali z prac Schumackera.
- Algorytm drzewa BSP jest bardzo wydajną metodą obliczania zależności widoczności wśród statycznej grupy wielokątów 3D oglądanych z dowolnego punktu obserwacji.
- Zapewnia on kompromis między:
  - początkowym czasochłonnym i pamięciochłonnym przetwarzaniem wstępnym
  - liniowym algorytmem wyświetlania, który jest wykonywany za każdym razem, gdy jest potrzebna nowa specyfikacja rzutowania.
- Dlatego algorytm dobrze się nadaje do zastosowań, w których zmienia się punkt obserwacji, natomiast same obiekty nie zmieniają się.

Algorytm drzewa binarnego podziału przestrzeni

- W algorytmie drzewa BSP skorzystano ze spostrzeżenia, że rasteryzacja wielokąta zostanie wykonana poprawnie (nie będzie zasłaniał niepoprawnie ani nie będzie zasłaniany niepoprawnie), jeżeli:
  – wszystkie wielokąty znajdujące się po stronie niewidocznej dla obserwatora zostaną poddane konwersji wcześniej,
  – potem zostanie wykonana konwersja tego wielokąta,
  – na końcu wszystkich wielokątów znajdujących się po tej samej stronie co obserwator.
- Musimy zapewnić to dla każdego wielokąta.
- Algorytm zapewnia łatwe określenie poprawnej kolejności rasteryzacji dzięki tworzeniu drzewa binarnego wielokątów (**drzewo BSP**).

Algorytm drzewa binarnego podziału przestrzeni

- W korzeniu drzewa BSP jest wielokąt wybrany spośród wielokątów, które mają być wyświetlone
- Algorytm działa poprawnie niezależnie od pierwotnego wyboru wielokąta.
- Wielokąt znajdujący się w korzeniu jest wykorzystywany do podziału środowiska na dwie półprzestrzenie.
- Jedna półprzestrzeń zawiera wszystkie pozostałe wielokąty znajdujące się przed wielokątem korzenia (zależy to od zwrotu normalnej wielokąta korzenia)
- W drugiej półprzestrzeni są umieszczane wszystkie wielokąty znajdujące się za wielokątem korzenia.
- Każdy wielokąt, który leży po obu stronach płaszczyzny wielokąta będącego w korzeniu, jest dzielony przez tę płaszczyznę, a jego przednie i tylne części są przypisane do odpowiednich półprzestrzeni.

Algorytm drzewa binarnego podziału przestrzeni

- Jeden wielokąt w każdej półprzestrzeni, przedniej i tylnej względem wielokąta będącego w korzeniu, staje się jego przednim albo tylnym potomkiem
- Każdy potomek jest rekursywnie wykorzystywany do przydzielenia pozostałych wielokątów do jego półprzestrzeni w podobny sposób.
- Algorytm kończy się, gdy każdy węzeł zawiera tylko jeden wielokąt.

Algorytm drzewa binarnego podziału przestrzeni

- Drzewo BSP może być przeglądane w zmodyfikowany sposób tak, żeby uzyskać poprawnie uporządkowaną według priorytetów listę dla dowolnego punktu obserwacji.
- Weźmy pod uwagę wielokąt będący w korzeniu.
- Dzieli on pozostałe wielokąty na dwa zbiory, z których każdy leży całkowicie po jednej stronie płaszczyzny wielokąta znajdującego się w korzeniu.
- Algorytm musi gwarantować wyświetlanie zbiorów w poprawnym, względnym porządku po to, żeby zapewnić:
  - brak możliwości mieszania się wielokątów z jednego zbioru z innymi,
  - wielokąt będący korzeniem był wyświetlany poprawnie i we właściwej kolejności względem innych.

Algorytm drzewa binarnego podziału przestrzeni

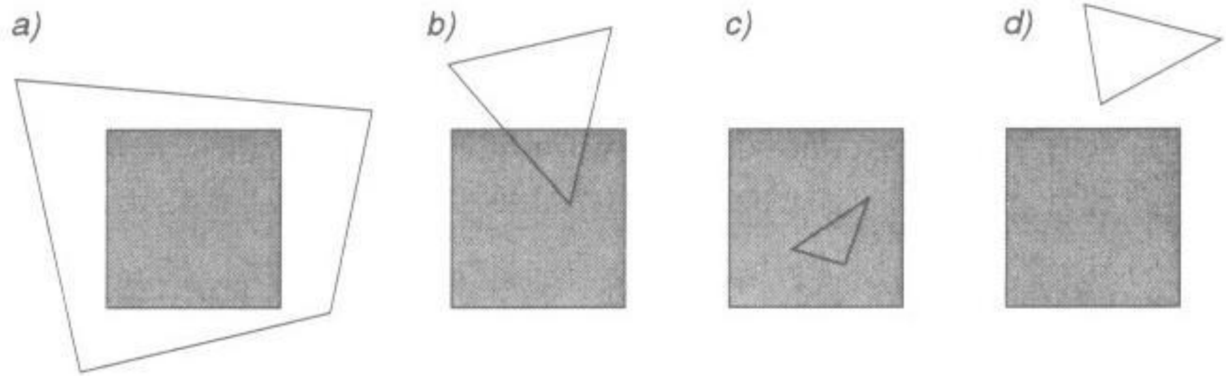
- Jeżeli obserwator jest w przedniej półprzestrzeni względem wielokąta znajdującego się w korzeniu, to algorytm musi
  - najpierw wyświetlić wszystkie wielokąty tylnej półprzestrzeni względem korzenia (te, które mogłyby zostać zasłonięte przez wielokąt znajdujący się w korzeniu),
  - potem wielokąt znajdujący się w korzeniu
  - i wreszcie wszystkie wielokąty z przedniej półprzestrzeni (te, które mogłyby zasłonić wielokąt znajdujący się w korzeniu)
- W przypadku gdy obserwator jest w tylnej półprzestrzeni, algorytm musi odwrócić kolejność wyświetlania
- Każdy z potomków korzenia jest przetwarzany rekursywnie za pomocą tego algorytmu.
- Podobnie jak w algorytmie sortowania ze względu na głębokość, algorytm z drzewem BSP:
  - wykonuje przecinanie i sortowanie z precyzją obiektową
  - wykorzystuje możliwości urządzenia rastrowego do zapisywania na poprzedniej zawartości z precyzją obrazową.

Algorytm Warnocka podziału powierzchni

Algorytmy podziału powierzchni

- Wszystkie *algorytmy podziału powierzchni* wykorzystują strategię dziel i zwyciężaj podziału przestrzennego na rzutni.
- Sprawdza się powierzchnię zrzutowanego obrazu.
- Jeżeli można łatwo zdecydować, które wielokąty w tym obszarze są widoczne, to są one wyświetlane.
- W przeciwnym przypadku powierzchnia jest dzielona na mniejsze powierzchnie, do których rekursywnie stosuje się proces decyzji logicznych.
- Gdy powierzchnia maleje, wówczas mniej wielokątów pokrywa każdą powierzchnię i w końcu staje się możliwe podjęcie decyzji.
- W tym podejściu wykorzystuje się spójność powierzchniową, ponieważ odpowiednio małe obszary obrazu będą zawarte w najwyżej jednym widocznym wielokącie.

Algorytm Warnocka



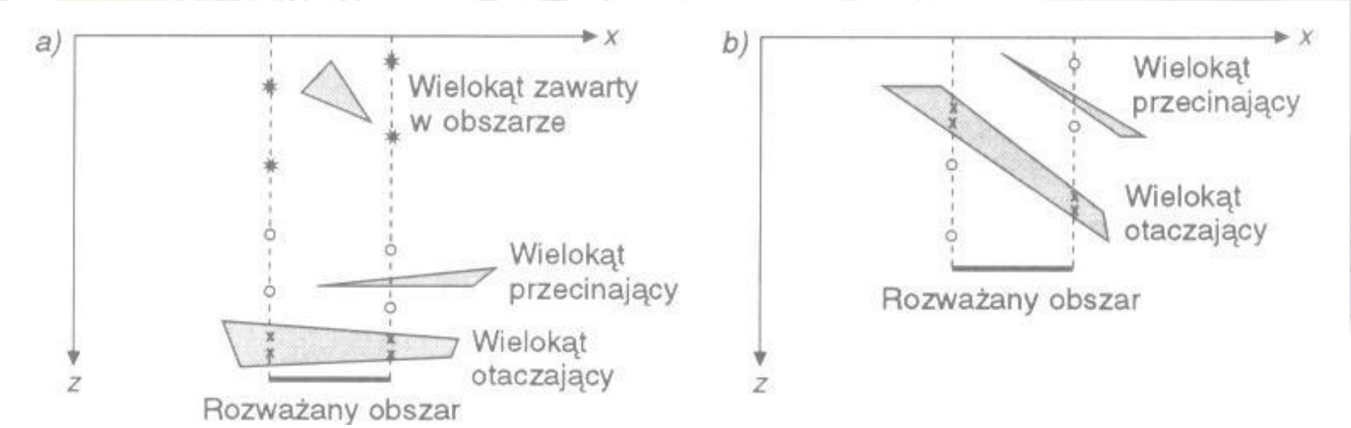
- Algorytm dzieli każdy obszar na cztery jednakowe kwadraty.
- W każdym kroku rekursywnego procesu podziału rzut każdego wielokąta może się znaleźć w jednej z czterech relacji względem odpowiedniego obszaru:
  - wielokąty **otaczające** całkowicie zawierają rozpatrywany obszar (rys. a)
  - wielokąty **przecinające** przecinają obszar (rys. b).
  - wielokąty **zawarte** są całkowicie wewnątrz obszaru (rys. c)
  - wielokąty **rozłączne** są całkowicie na zewnątrz obszaru (rys. d).
- Wielokąty rozłączne oczywiście nie mają żadnego wpływu na rozważany obszar.
- Część wielokąta przecinającego, która jest na zewnątrz obszaru, również nie jest istotna
- Część wielokąta przecinającego, która jest wewnątrz obszaru, powinna być traktowana tak samo jak wielokąt zawarty w obszarze.

Algorytm Warnocka

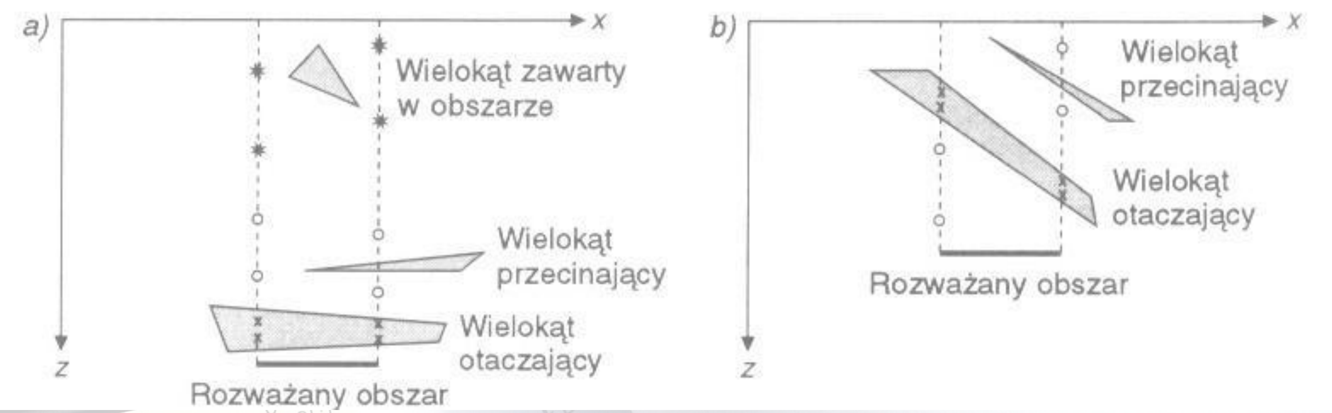
- W czterech przypadkach można łatwo podjąć decyzję co do obszaru i nie musi on być dalej dzielony:
  1. Wszystkie wielokąty są rozłączne względem obszaru – obszar można wypełnić barwą tła.
  2. Jest tylko jeden wielokąt przecinający lub tylko jeden wielokąt zawarty
     - obszar jest najpierw wypełniany barwą tła,
     - potem wykonuje się rasteryzację wielokąta zawartego lub części wielokąta przecinającego
  3. Jest jeden wielokąt otaczający i nie ma wielokątów przecinających ani zawartych. – obszar jest wypełniany barwą wielokąta otaczającego.
  4. Jest więcej niż jeden wielokąt przecinający, zawarty albo otaczający obszar, ale tylko jeden wielokąt otaczający jest przed wszystkimi innymi wielokątami – obszar jest wypełniany barwą wielokąta otaczającego.

Algorytm Warnocka

- Sprawdzenie, czy wielokąt otaczający jest z przodu, wykonuje się na zasadzie obliczania współrzędnych ***z*** we wszystkich czterech rogach obszaru wykorzystując równania płaszczyzn wszystkich wielokątów otaczających, przecinających i zawartych
- jeżeli jest wielokąt otaczający, dla którego współrzędne ***z*** czterech rogów są większe (bliższe obserwatorowi) niż dla każdego innego wielokąta, to cały obszar może być wypełniony barwą tego wielokąta otaczającego.
- Przypadki 1, 2 i 3 są łatwe do zrozumienia.
- Przypadek 4 jest dodatkowo zilustrowany na rysunku



Algorytm Warnocka



- W części a) wszystkie cztery przecięcia wielokąta otaczającego są bliższe punktu obserwacji (który jest w nieskończoności na osi ***z*** niż jakiekolwiek inne przecięcia.
- W konsekwencji cały obszar jest wypełniony barwą wielokąta otaczającego.
- W części b) nie można podjąć decyzji, chociaż wielokąt otaczający wydaje się być z przodu wielokąta przecinającego, ponieważ z lewej strony płaszczyzna wielokąta przecinającego jest przed płaszczyzną wielokąta otaczającego.
- Algorytm Warnocka zawsze dzieli obszar w celu uproszczenia problemu.
- Po podziale trzeba ponownie sprawdzić jedynie wielokąty zawarte i przecinające
- Wielokąty otaczające i rozłączne względem oryginalnego obszaru są wielokątami otaczającymi i rozłącznymi każdego obszaru otrzymanego w wyniku podziału.

Algorytm Warnocka

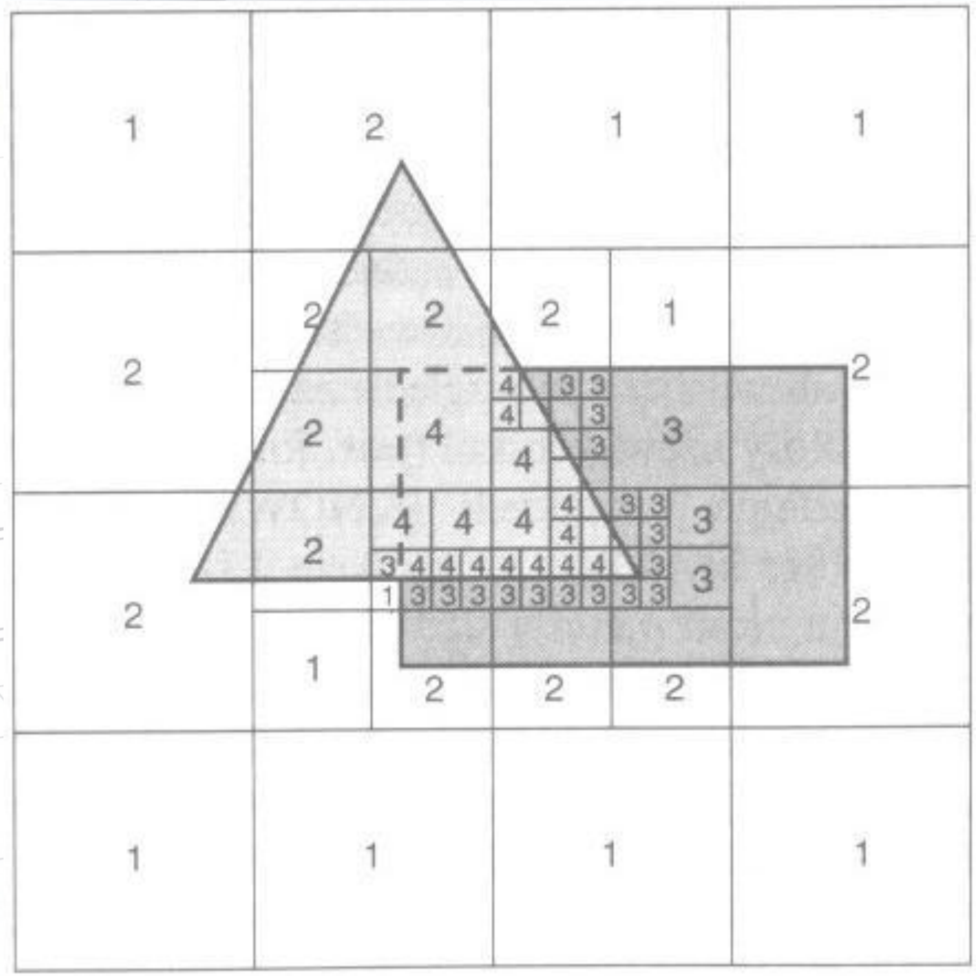
- Do tego momentu algorytm działał z precyzją obiektową, z wyjątkiem rasteryzacji tła i wielokątów obciętych w czterech przypadkach.
- Jednak te operacje rasteryzacji z precyzją obrazową można zastąpić operacjami z precyzją obiektową, które dają dokładną reprezentację widocznych powierzchni:
  - albo kwadrat o wielkości obszaru (przypadki 1, 3 i 4),
  - albo wielokąt obcięty do obszaru, wraz z boolowskim uzupełnieniem względem obszaru reprezentującego widzialną część tła (przypadek 2).
- A co z przypadkami innymi niż te cztery?
- Jedno z rozwiązań polega na zatrzymaniu podziału po osiągnięciu rozdzielczości urządzenia wyświetlającego.

Algorytm Warnocka

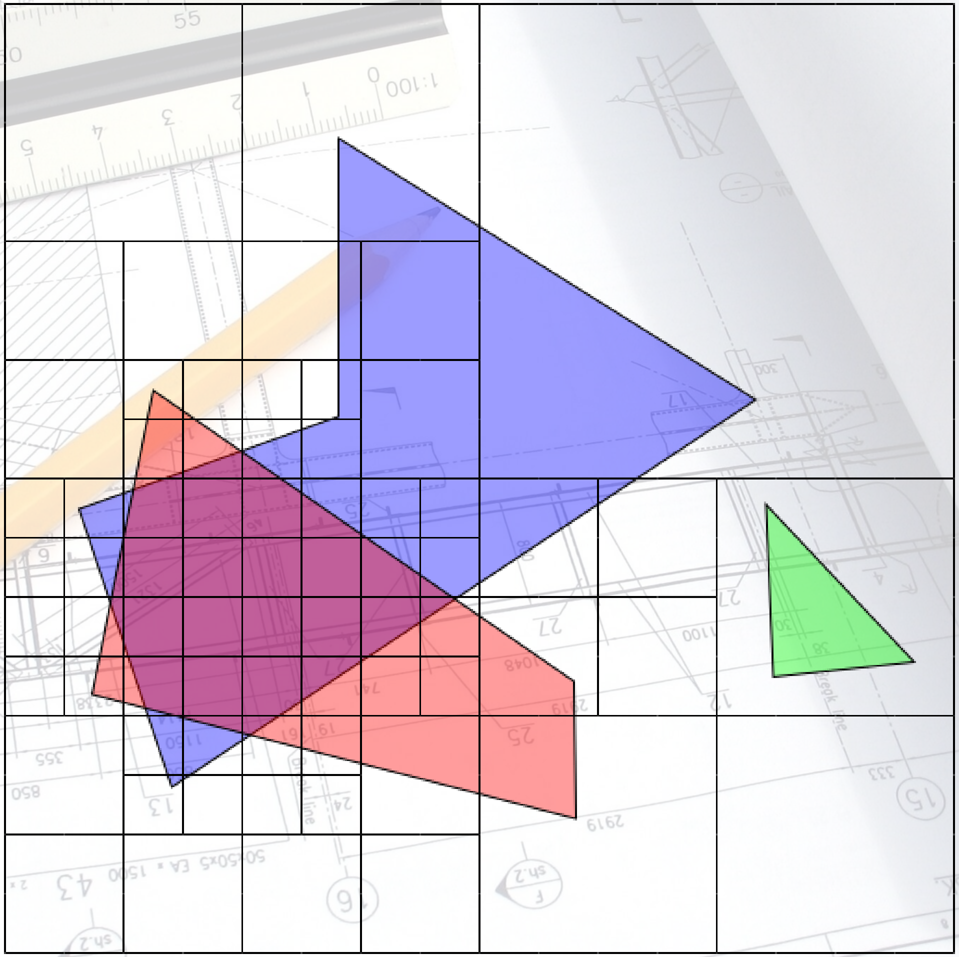
- Dlatego w urządzeniu wyświetlającym o rozdzielczości rastra 1024 x 1024 trzeba najwyżej 10 poziomów podziału.
- Jeżeli po tej maksymalnej liczbie podziałów nie wystąpi żaden z przypadków od 1 do 4, to oblicza się głębokość wszystkich odpowiednich wielokątów w środku tego niepodzielonego obszaru o wielkości piksela.
- Wielokąt o najbliższej współrzędnej ***z*** określa barwę obszaru.
- Można też ze względu na problem eliminacji zakłóceń użyć kilku dalszych poziomów podziału w celu określenia barwy piksela na zasadzie przypisania każdemu obszarowi podpikselowemu wag zależnych od wielkości obszaru.
- Ta właśnie operacja wykonywana wówczas, gdy dla obszaru nie zachodzi jeden z prostych przypadków
- W efekcie decyduje o tym, że jest to podejście z precyzją obrazową.

Algorytm Warnocka

- Na rysunku pokazano prostą scenę i podział potrzebny do jej wyświetlenia.
- Liczba w każdym obszarze podziału odpowiada jednemu z czterech przypadków
- Tam gdzie nie jest wpisana żadna liczba, nie zachodzi żaden z tych czterech przypadków.



Algorytm Warnocka



Algorytmy wyznaczania powierzchni krzywoliniowych

Algorytmy wyznaczania powierzchni krzywoliniowych

- Wszystkie dotychczas prezentowane algorytmy, z wyjątkiem z-bufora, były zdefiniowane dla ścian wielokątowych.
- Obiekty takie jak powierzchnie krzywoliniowe muszą być najpierw aproksymowane za pomocą wielu małych ścianek, zanim można użyć wielokątowej wersji któregoś z algorytmów.
- Chociaż można zrobić taką aproksymację, często lepiej jest bezpośrednio wykonać rasteryzację powierzchni krzywoliniowej, eliminując zakłócenia wielokątowe i dodatkową pamięć potrzebną do aproksymacji wielokątowej.
- W grafice komputerowej są popularne powierzchnie drugiego stopnia – tzw. **kwadryki**.
- Algorytmy wyznaczania powierzchni widocznych dla powierzchni drugiego stopnia zostały opracowane przez Weissa, Woona, Mahla, i Sarraga.

Algorytmy wyznaczania powierzchni krzywoliniowych

- Znajdują oni przecięcia dwóch powierzchni drugiego stopnia, co prowadzi do równań czwartego stopnia dla zmiennych *x, y* i *z*, których pierwiastki trzeba znajdować numerycznie.
- Levin redukuje to do problemu drugiego stopnia przez parametryzowanie krzywych przecięcia.
- Kule, będące specjalnym przypadkiem powierzchni drugiego stopnia, są najłatwiejsze z tego punktu widzenia
- Są one szczególnie interesujące ze względu na to, że cząsteczki są często wyświetlane jako zbiory barwnych kul.
- Opracowano wiele algorytmów wyświetlania cząsteczek.
- Rendering kuli można wykonać metodą śledzenia promieni.

Algorytmy wyznaczania powierzchni krzywoliniowych

- Jeszcze większą elastyczność można osiągnąć w przypadku parametrycznych powierzchni sklejanych, ponieważ są one bardziej ogólne i umożliwiają zapewnienie ciągłości stycznej na granicach płatów.
- Catmull jako pierwszy opracował algorytm wyświetlania dla powierzchni bikubicznych.
- Zgodnie z algorytmem Warnocka płat jest rekursywnie dzielony w kierunkach *s* i *t* na cztery płaty dopóty, dopóki ich rzuty pokrywają więcej niż 1 piksel.
- Algorytm z-bufora określa, czy płat jest widoczny w tym pikselu.
- Jeżeli tak, to oblicza się barwę i umieszczają w pamięci obrazu.

Algorytmy wyznaczania powierzchni krzywoliniowych

- W innym podejściu wykorzystuje się adaptacyjny podział każdego płata bikubicznego dopóty, dopóki dzielony płat nie znajdzie się w zakresie przyjętej tolerancji płaskości.
- Ta tolerancja zależy od rozdzielczości urządzenia wyświetlającego i orientacji dzielonego obszaru względem rzutni, tak że jest eliminowane niepotrzebne dzielenie.
- Płat może być dzielony tylko w jednym kierunku, jeżeli jest już wystarczająco płaski w drugim kierunku.
- Po dostatecznym podziale płat może być traktowany jak czworokąt.
- Małe wielokąty określone przez cztery rogi każdego płatu są przetwarzane za pomocą algorytmu przeglądania wierszami,
- Umożliwia to mieszanie powierzchni wielokątowych i bikubicznych.
- Algorytmy wykorzystujące tę podstawową ideę zostały opracowane przez Lane'a i Carpentera oraz Clarka.

Względne oszacowanie wydajności czterech algorytmów

Algorytm	Liczba wielokątów w scenie		
	100	2500	60000
Sortowanie ze względu na głębokość	1*	10	507
z-bufor	54	54	54
Przeglądanie wierszy	5	21	100
Podział obszaru Warnocka	11	64	307

* Elementy w tablicy zostały tak znormalizowane, żeby w tym miejscu było 1