Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Scena i kolizje

Organizacja sceny. Techniki optymalizacji renderowania. Wykrywanie i reakcja na kolizje.

Wydział
Informatyki

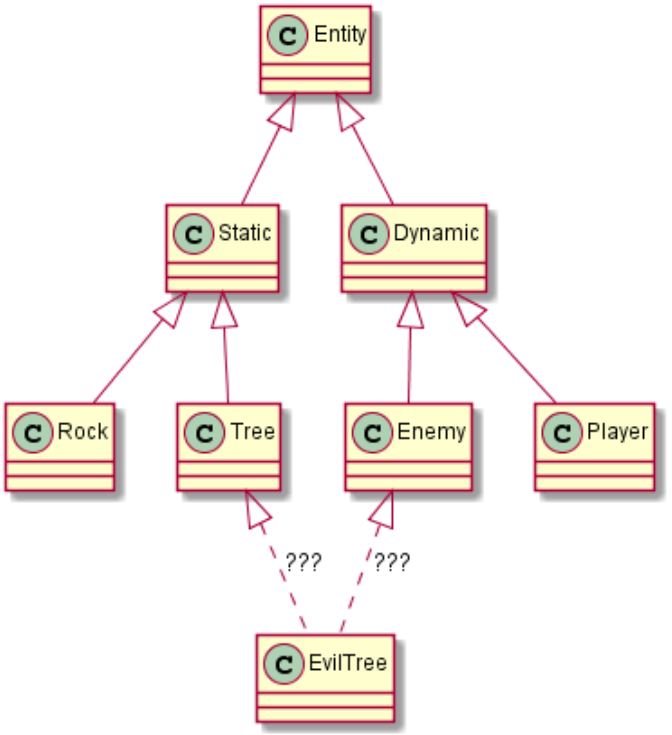
Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok

ORGANIZACJA SCENY

- W jaki sposób **reprezentować** obiekty na scenie (***encje***, ang. *entities*)?
  - Elementy sceny posiadają swoje **atrybuty**, np.:
    - **Położenie**
    - **Prędkość**
    - **Poziom życia**
    - **Stan** (*w trakcie odpoczynku*, *w trakcie biegu*, *w trakcie strzelania*)
  - Posiadają one też zestaw charakterystycznych **operacji logicznych**, np.:
    - Pocisk realizuje ***ruch***
    - Przeciwnik ***szuka ścieżki*** do gracza, którego goni
    - Pojazd ***zmienia swoją przyczepność*** zależnie od nawierzchni na której się znajduje i prędkości
    - Futurystyczna broń plazmowa ***ładuje się***
- Modele reprezentacji encji
  - Model **obiektowy**
  - Model **komponentowy**

ORGANIZACJA SCENY

- **Model obiektowy** reprezentacji sceny
  - Encje jako **instancje wyspecjalizowanych klas** dziedziczących po **wspólnym przodku**
  - **Hierarchia dziedziczenia** może uwzględniać **logiczną charakterystykę elementów sceny**, np.:
    - Elementy, które mogą się **poruszać**
    - Elementy, z którymi mogą **zachodzić kolizje**
    - Elementy, które są **przeciwnikami**



Źródło obrazu:
Boreal Games, gamedev.net

ORGANIZACJA SCENY

- **Model obiektowy** reprezentacji sceny
  - Hierarchia klas wykorzystuje **polimorfizm**
    - kolejne poziomy coraz bardziej wyspecjalizowanych klas oferują **wirtualne metody**, które mogą zostać *przysłonięte* przez implementacje w klasach jeszcze bardziej wyspecjalizowanych, np. (pseudokod):

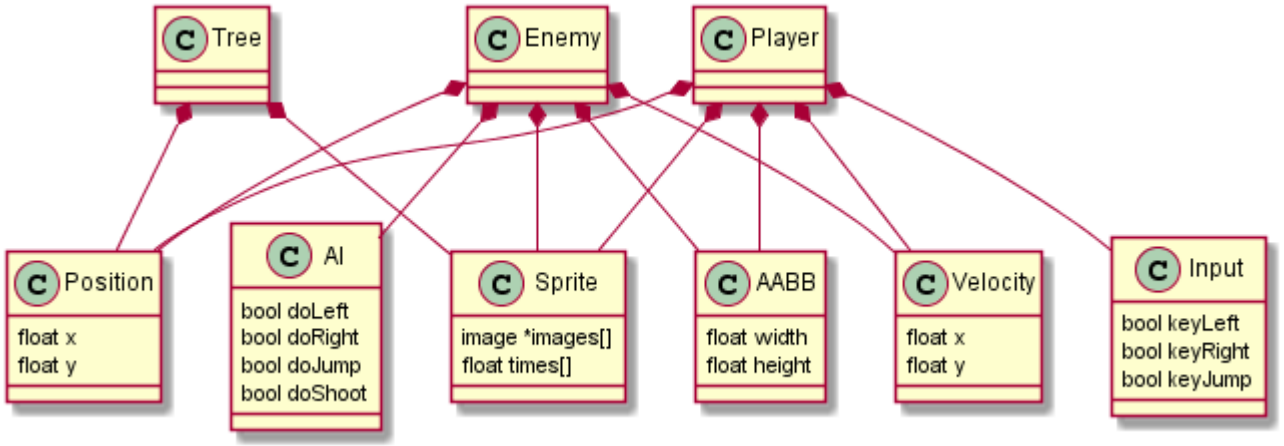
```
abstract class Node {
        vec3 Position;
        virtual void Render() {};
        virtual void Update(double T) {};
}

abstract class MovableNode : Node {
        vec3 Direction;
        float Speed;
        void Update(double T) {
                Position = Position + Direction * Speed * T;
        }
}

class Zombie : MovableNode {
        void Render() {
                DrawUglyZombie();
        }
}
```

Inna klasa dziedzicząca po *MovableNode* może do metody *Update()* dodać np. odwołanie do sztucznej inteligencji, w przeciwieństwie do klasy *Zombie*.

Dodatkowo pośrednio mogłaby się np. znaleźć klasa ***DestroyableNode***, która dodawałaby liczbę punktów życia (*hit points*) i sprawdzenie w *Update()*, czy przypadkiem nie spadły one poniżej zera.

ORGANIZACJA SCENY

- **Model komponentowy** reprezentacji sceny
  - **Komponenty** przechowują dane
    - Cząstki logicznie powiązanych ze sobą atrybutów
  - **Encje** są faktycznymi elementami sceny
    - Są do nich dołączane komponenty
  - **Systemy** rozwiązują logikę gry
    - Na podstawie encji, na które składają się komponenty



Źródło obrazu:
Boreal Games, gamedev.net

ORGANIZACJA SCENY

Nie należy mylić tego problemu z reprezentacją obiektową bądź komponentową – to coś niezależnego!

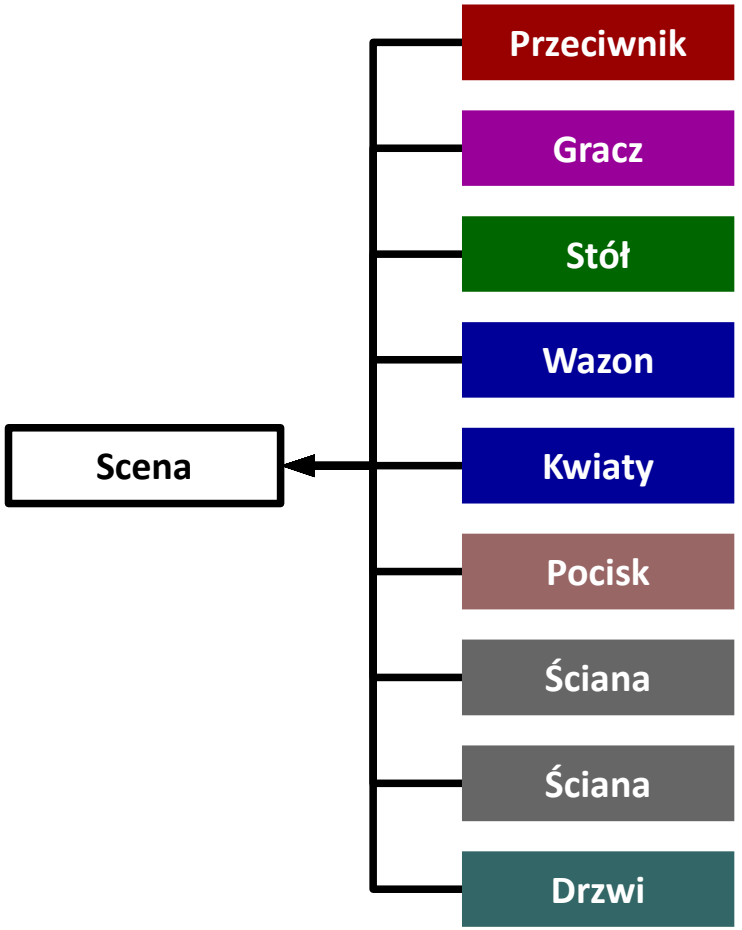
- **Kolekcje** elementów na scenie
  - Potrzebna jest możliwość łatwego **zgrupowania** i **przechodzenia** kolekcji wszystkich elementów
  - Różne **konfiguracje** reprezentujące hierarchię na scenie:
    - **Lista** elementów sceny
    - **Drzewo** sceny
    - **Graf** sceny

Źródło obrazu:
Fabehome.com

ORGANIZACJA SCENY

Jako że każda klasa posiada własną implementację wirtualnych metod uwzględniającą jej logikę, można łatwo wywołać daną metodę dla wszystkich obiektów w kolekcji – a każdy z obiektów zrobi wtedy to, co jest dla niego charakterystyczne.

- **Lista** elementów sceny
  - **Brak hierarchii** obiektów, przydatna do prostych scen



ORGANIZACJA SCENY

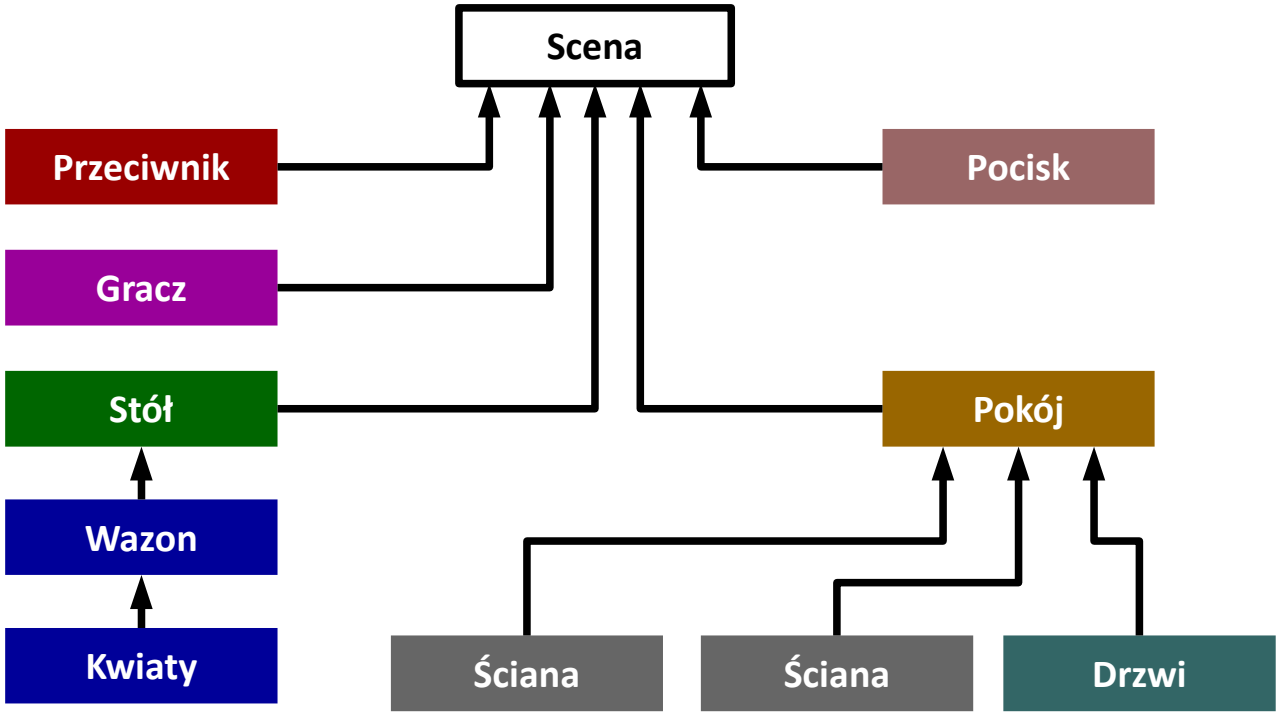
Jako że jest to **metoda wirtualna**, zostanie wywołana implementacja *Render()* charakterystyczna dla konkretnego, najbardziej wyspecjalizowanego typu danej instancji.

- **Lista** elementów sceny
  - **Brak hierarchii** obiektów, przydatna do prostych scen
  - **Łatwe przechodzenie**: prosta iteracja

```
vector<Node *> sceneNodes;

sceneNodes.Add(new Zombie());
sceneNodes.Add(new Player());
sceneNodes.Add(new Wall());
sceneNodes.Add(new Bullet());

// ...

for (int i = 0; i < sceneNodes.Count; ++i) {
        sceneNodes[i]→Render();
}
```

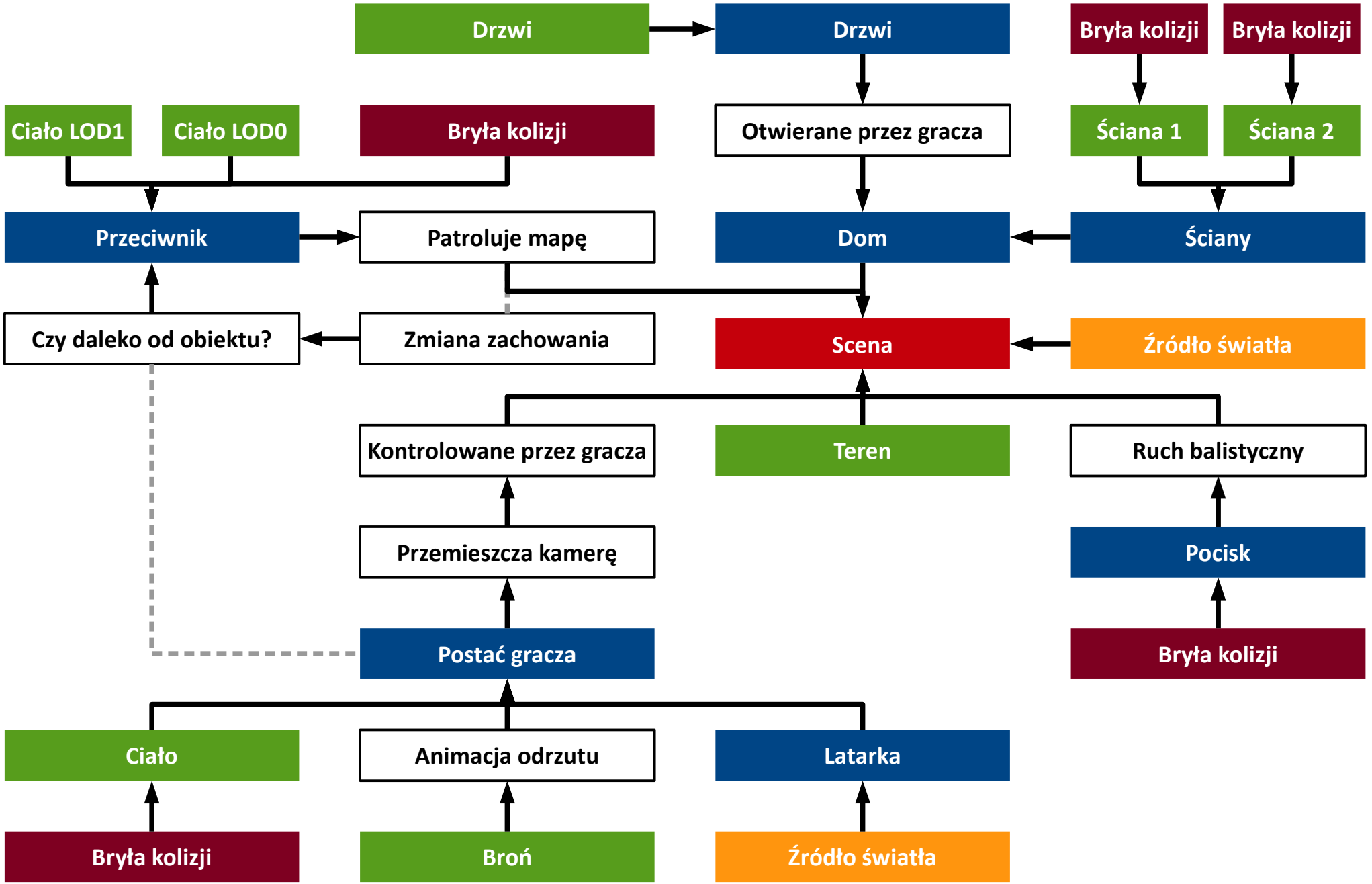
ORGANIZACJA SCENY

- **Drzewo** elementów sceny
  - **Grupowanie** elementów
  - **Lokalne układy współrzędnych**
    - **Transformacja rodzica** wpływa też **na jego dzieci**
  - Przechodzenie drzewa **od korzenia do liści**, *pre-order*

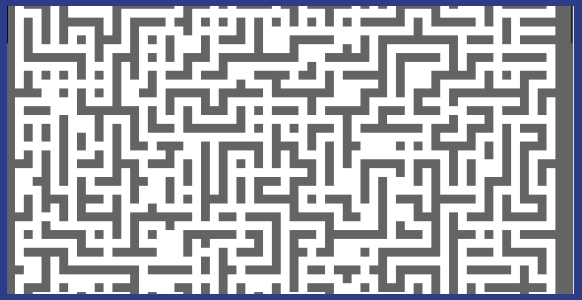


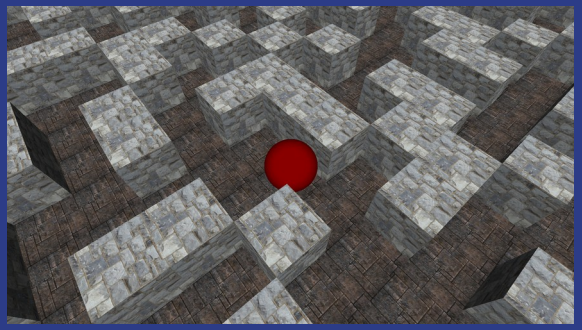
ORGANIZACJA SCENY

- Bardziej złożone struktury danych o obiektach tworzących scenę i powiązań logicznych między nimi nazywa się **grafem sceny** (ang. *scene graph*)
  - Jest to najczęściej **acykliczny graf skierowany** (ang. *directed acyclic graph*), co czyni go znów w zasadzie **drzewem**
  - Różnica polega na tym, że **węzły grafu** nie muszą być *stricte* obiektami – **mogą być np. transformacjami, zachowaniami lub specyficznymi akcjami**
    - Każdy z tych węzłów i tak dziedziczy po **bazowej klasie węzła**
    - Graf może zawierać też np. **mniej szczegółowe reprezentacje obiektów**, **bryły kolizji**, **kamery**, metody **sterowania**, odwołania do **sztucznej inteligencji**...


Drzwi
Drzwi
Bryła kolizji
Bryła kolizji
Ciało LOD1
Ciało LOD0
Bryła kolizji
Otwierane przez gracza
Ściana 1
Ściana 2
Przeciwnik
Patroluje mapę
Dom
Ściany
Czy daleko od obiektu?
Zmiana zachowania
Scena
Źródło światła
Kontrolowane przez gracza
Teren
Ruch balistyczny
Przemieszcza kamerę
Pocisk
Postać gracza
Bryła kolizji
Ciało
Animacja odrzutu
Latarka
Bryła kolizji
Broń
Źródło światła

ORGANIZACJA SCENY



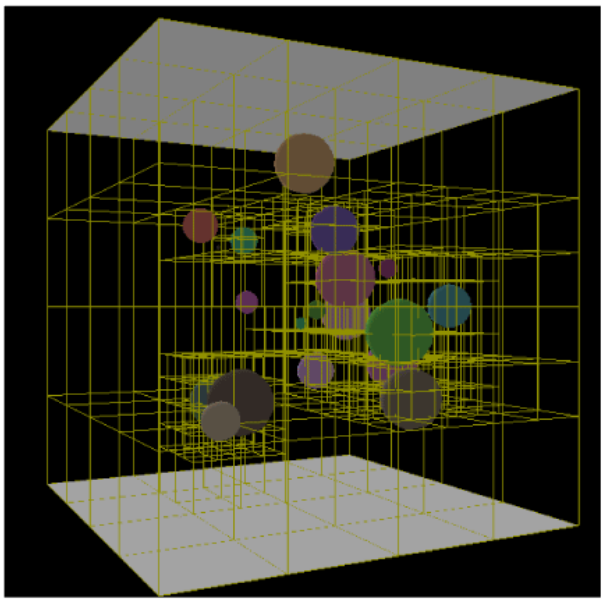


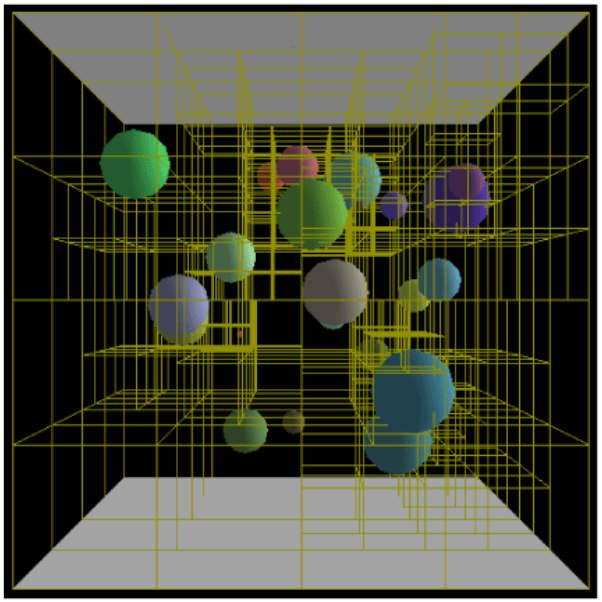
Przykład reprezentacji świata jako **macierzy obiektów** (np. *bitmapy*).

Poszczególne komórki macierzy zawierają informacje o znajdujących się w nich blokach.

Źródła obrazów:
Martin, Ludumdare;
Sergey Tihon;
G. Qin and N. Jiang.
„Rendering for 3D Animation Based on Octree”

- Innym zagadnieniem jest **reprezentacja przestrzenna** elementów sceny
  - Pozwala na **podział przestrzeni** i przyporządkowanie poszczególnych elementów do odpowiednich jej części
  - Łatwe **znajdowanie** obiektów z danej części przestrzeni
  - Łatwe określanie czy dwa obiekty są w **sąsiedztwie**
  - Przykłady reprezentacji:
    - **Siatki obiektów**
    - **Drzewa dwójkowe, czwórkowe, ósemkowe**



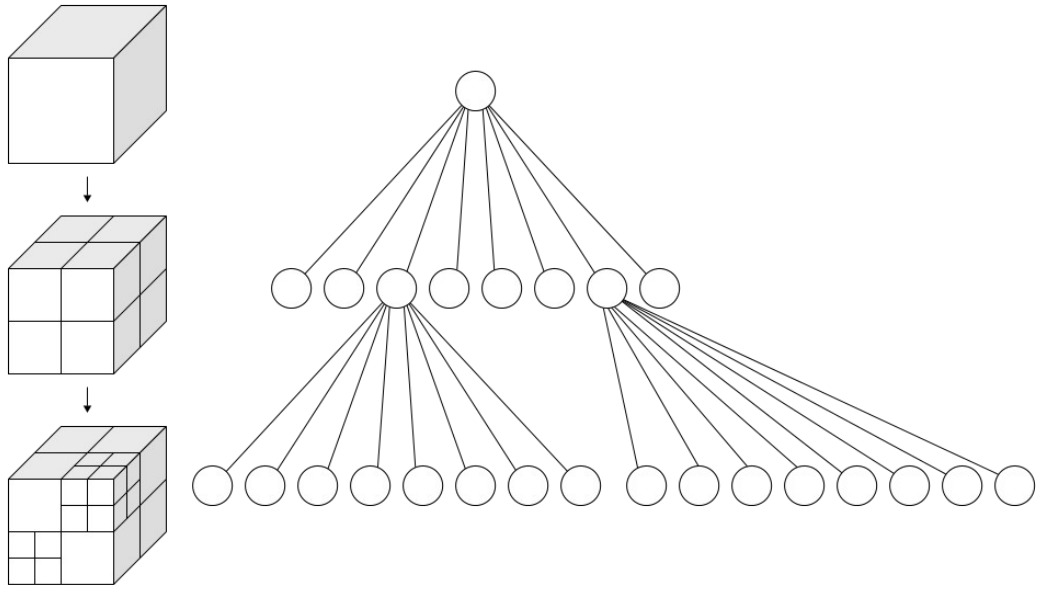


ORGANIZACJA SCENY

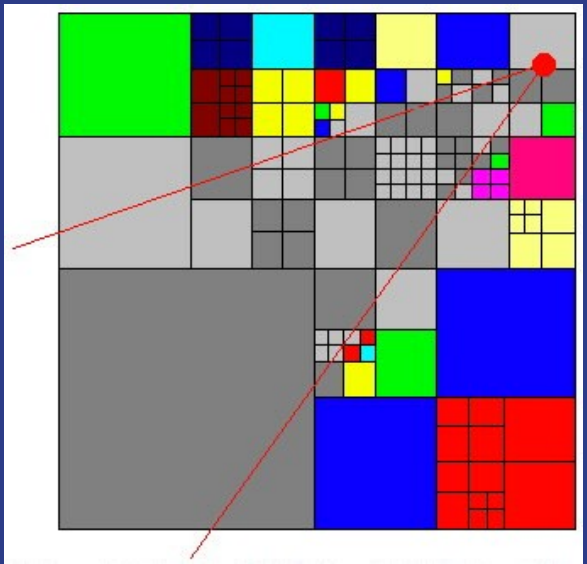
Idea opisu świata za pomocą wokseli może być też wykorzystana do samego **renderowania obrazu**. Seria gier *Comanche* oparta na silniku *Voxel Engine* oferowała w latach '90 niespotykane wcześniej zróżnicowanie terenu. Jednak karty graficzne nie są przystosowane do szybkiego renderowania tak reprezentowanej sceny, co zadecydowało o upadku tej idei.

Źródła obrazów:
Nova Logic
Wikipedia

- **Drzewa dwójkowe** (ang. *binary tree*), **czwórkowe** (ang. *quadtree*) i **ósemkowe** (ang. *octree*) to idea bazująca na **rekurencyjnym podziale przestrzeni**
  - Dzielimy przestrzeń zawsze **na dwie równe części wzdłuż każdego z wymiarów** – aż do osiągnięcia zadanego poziomu
    - Komórki wynikające z podziału nazywa się **wokselami** (ang. *voxel*)
    - Każdy kolejny poziom zawiera odpowiednio dla drzew dwójkowych/czwórkowych/ósemkowych: **dwa**, **cztery** i **osiem wokseli na każdy woksel poprzedniego poziomu**



ORGANIZACJA SCENY

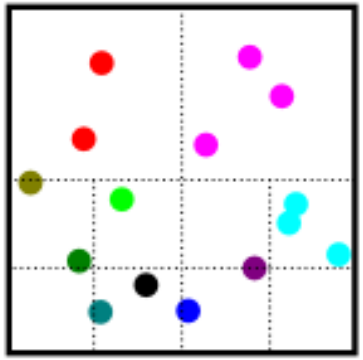


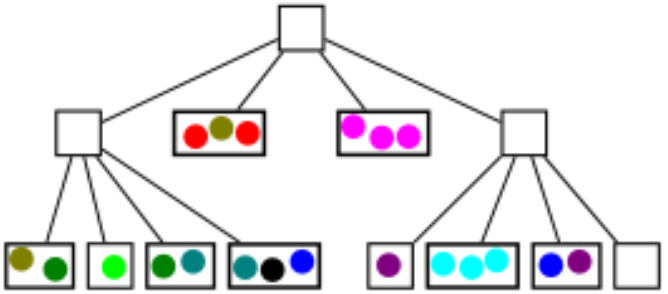
Problem sprawdzenia widzialności poszczególnych wokseli dla drzewa czwórkowego.

Rysunek pokazuje, że wystarczy z negatywnym skutkiem sprawdzić gruboziarniste woksele, by móc odrzucić wszystkie zawarte w nich woksele drobnoziarniste.

Źródła obrazów:
Jaap Suter, Flipcode
Videotutorialsrock.com

- **Drzewa dwójkowe**, **czwórkowe** i **ósemkowe** to idea bazująca na rekurencyjnym podziale przestrzeni
  - Każdy element sceny przyporządkowany jest do **jednego lub więcej wokseli** powstałych z najbardziej drobnoziarnistego podziału
    - Jeżeli przynależy do woksela *A*, to przynależy też do wszystkich wokseli **poziom wyżej** w hierarchii
  - Łatwe optymalizowanie **sprawdzania kolizji**, **widzialności**:
    - *Czy gracz widzi woksel najniższego poziomu*?
      - ***Tak*** – sprawdzamy każdy z zawartych w nim wokseli następnego poziomu
      - ***Nie*** – to znaczy, że nie widzi też żadnego obiektu znajdującego się w wokselach następnych poziomów





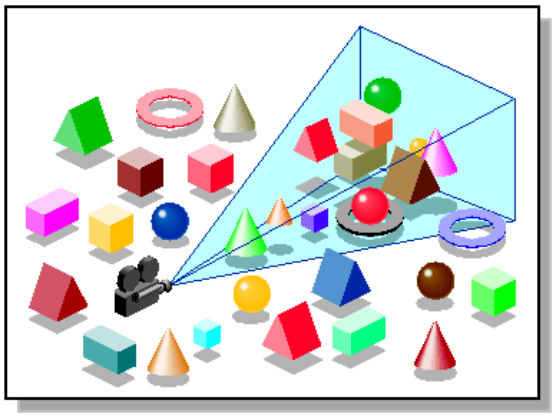
Źródło obrazu:
Bethesda

OPTYMALIZACJA RENDEROWANIA

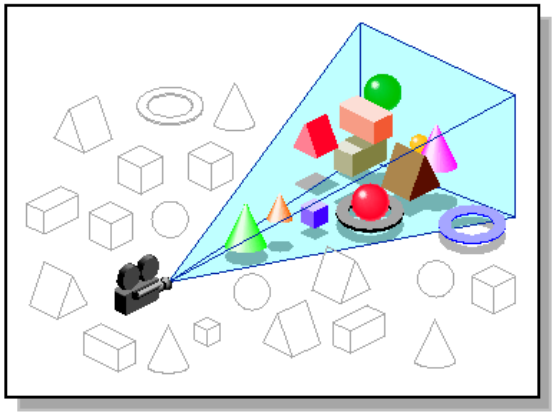
Jeśli przekazujemy do wyrenderowania całą scenę, wówczas **przetwarzane są wszystkie wierzchołki** ze zdefiniowanego zbioru.

Także te, które **nie są widziane przez kamerę**. Dlatego warto ograniczyć po stronie klienta zbiór wierzchołków tylko do tych, które opisują obiekty faktycznie **widziane**.

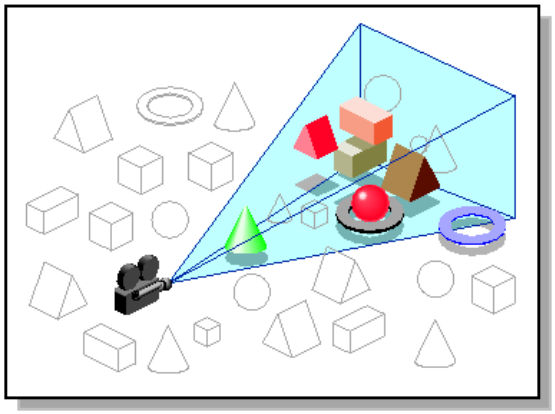
- Techniki **pomijania** podczas renderowania tych obiektów, które **nie są widziane** przez kamerę:



- **Frustum culling**
  - Wykorzystanie **bryły widzenia** i renderowanie tylko tych obiektów, które znajdują się w jej obrębie

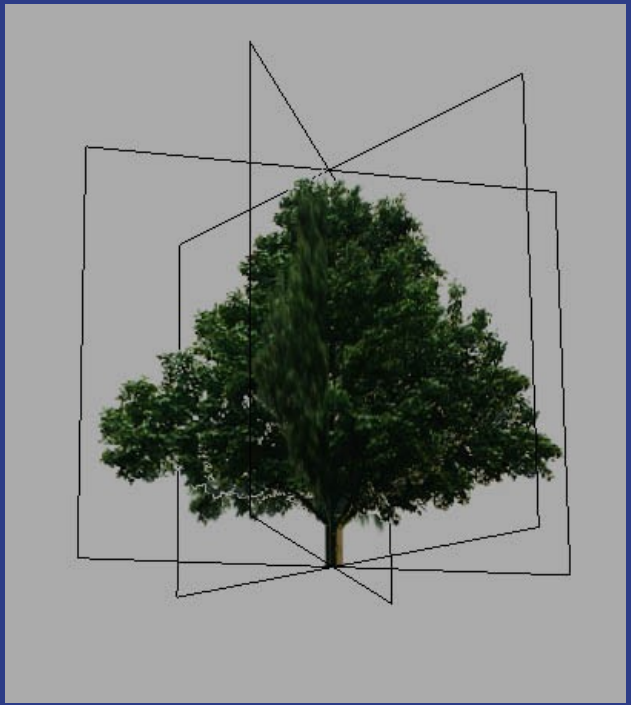


- **Occlusion culling**
  - Znalezienie obiektów, które **są zasłonięte** przez inne obiekty i ich pominięcie



Źródło obrazów:
OpenGL Optimizer Programmer's Guide

OPTYMALIZACJA RENDEROWANIA



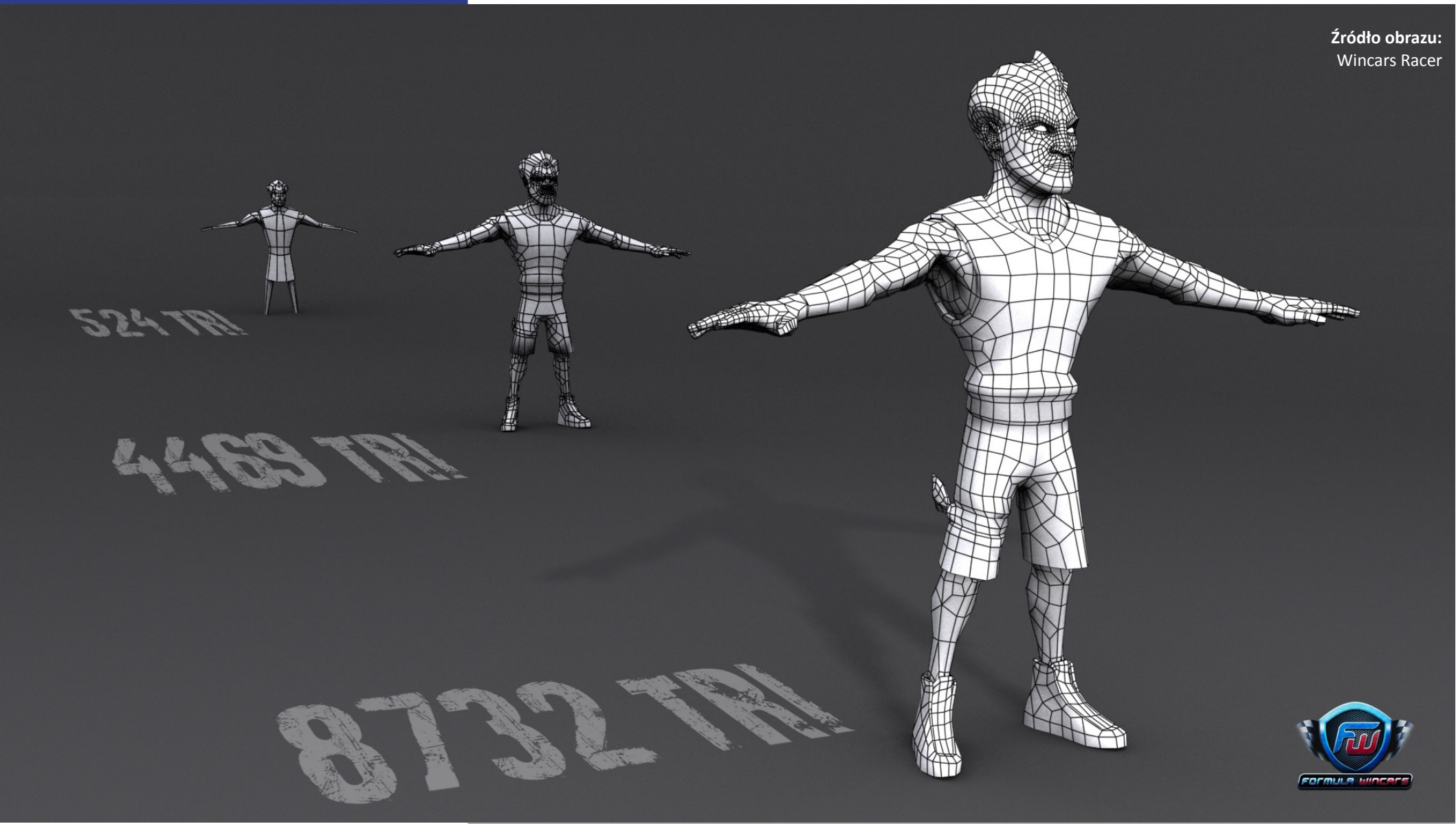
Odległe obiekty, np. drzewa, mogą być zastępowane *impostorami* (*sprite'ami*).

- Innym sposobem optymalizującym scenę na potrzeby renderowania są podejścia nazywane ***contribution culling***
  - **Pomijane** lub **ograniczane** są te elementy sceny, których **wpływ na wygląd klatki jest znikomy**, np.:
    - **Pomijanie odległych** obiektów (zasięg widoczności)
    - **Zmniejszenie liczby źródeł światła** podczas renderowania, jeśli nie wszystkie mają istotny wpływ na wygląd obiektu
    - **Ograniczenie szczegółowości** obiektu gdy ten znajduje się daleko od kamery

Źródła obrazów:
Bethesda
Sylexer, Unity3D Forums

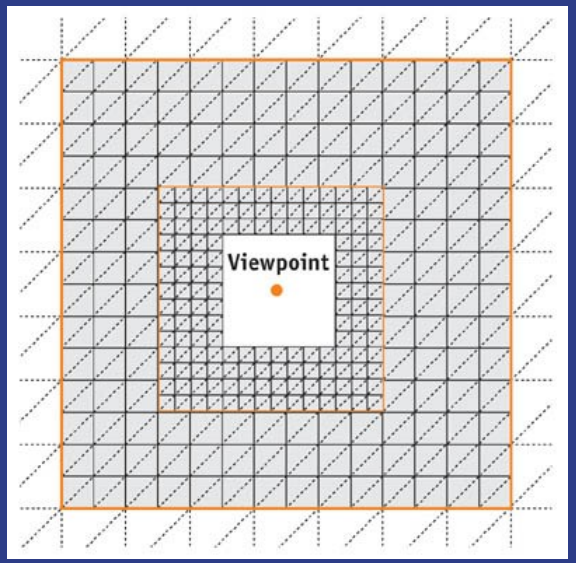
OPTYMALIZACJA RENDEROWANIA

- ***Level of detail*** - technika optymalizacji z gatunku *contribution culling* polegająca na wyświetlaniu **mniej szczegółowych brył** gdy znajdują się one daleko od kamery



Źródło obrazu:
Wincars Racer

OPTYMALIZACJA RENDEROWANIA

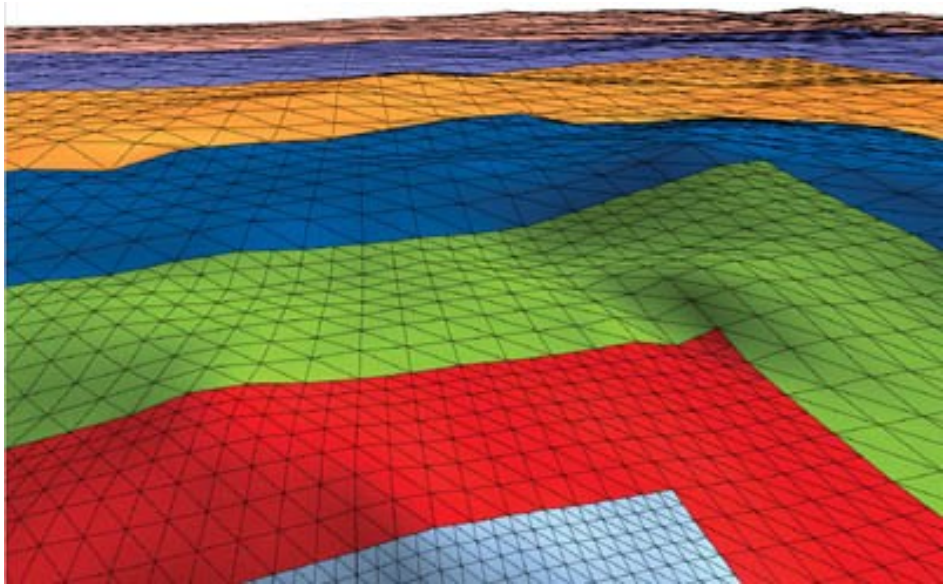


Podział mapy na segmenty, bliższe graczowi renderowane są z wyższym *LOD*.

Należy zadbać o przejścia pomiędzy segmentami, by nie powstały dziury w powierzchni na styku różnych *LOD*.

Źródło obrazów:
Microsoft Research. GPU Gems 2.

- ***Level of detail*** - technika optymalizacji z gatunku *contribution culling* polegająca na wyświetlaniu **mniej szczegółowych brył** gdy znajdują się one daleko od kamery
  - Potocznie zwana ***LOD*** (ang. *Level Of Detail*)
  - Można wykorzystać również do renderowania **powierzchni terenu**
  - Przydatne są techniki **wybiórczego renderowania**, gdzie określane są wybrane indeksy wierzchołków

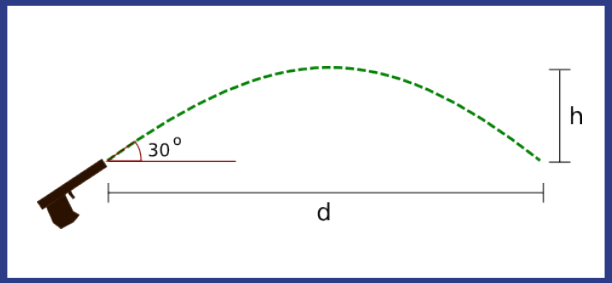


STRZELANIE

- Jak w prosty sposób zrealizować w grze **strzelanie**?

Źródło obrazu:
Rockstar

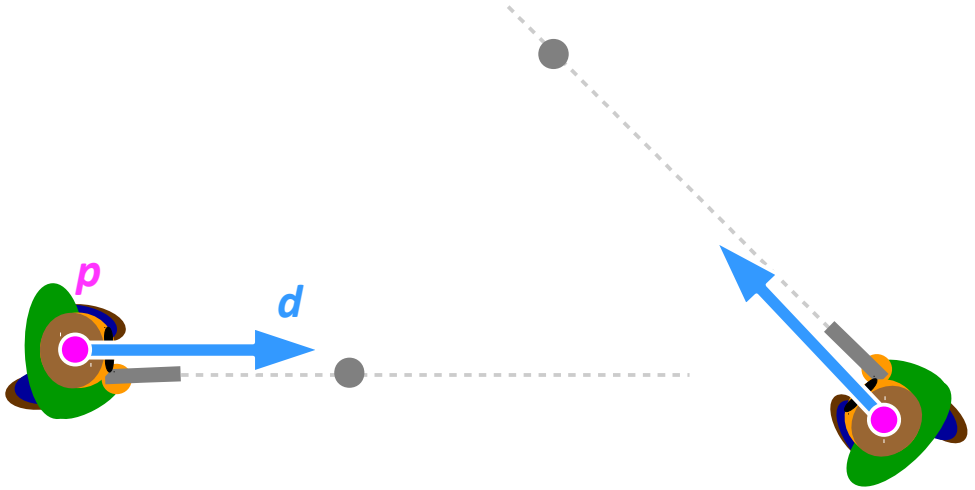
STRZELANIE



Prosta balistyka może opierać się na przykład na modyfikowaniu podczas aktualizacji stanu pocisku **współrzędnej pionowej** jego wektora kierunku.

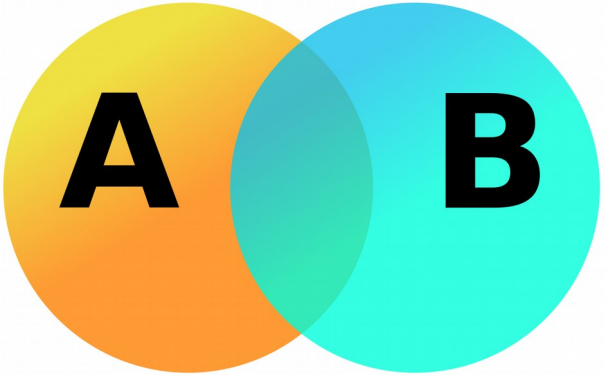
Źródło obrazu:
http://math.cornell.edu

- Jak w prosty sposób zrealizować w grze **strzelanie**?
  - Wciśnięcie przez gracza klawisza odpowiadającego za strzał powinno skutkować:
    - **Dodaniem do kolekcji obiektów sceny** pocisku
    - Nadaniem pociskowi odpowiedniego **kierunku** i **szybkości**
    - **Aktualizując stan** pocisku należy zmieniać jego położenie z uwzględnieniem kierunku i szybkości
    - Gdy pocisk trafi w coś lub znajdzie się odpowiednio daleko, powinniśmy **usunąć go z kolekcji obiektów sceny**



KOLIZJE

- **Kolizją** nazywamy sytuację w której dwa obiekty **mają część wspólną**
  - **OpenGL nie wspiera** technik związanych **z kolizjami**, jest to wyłącznie *API* graficzne
  - Implementacja sprawnie działającej obsługi kolizji **to wyzwanie programistyczne**
  - **Nie ma uniwersalnych metod** – każde rozwiązanie powinno być szyte na miarę konkretnego zastosowania
  - Rozpatrywanie kolizji jest **kosztowne obliczeniowo**
    - W naiwnym podejściu sprowadza się do sprawdzenia potencjalnych kolizji **między wszystkimi obiektami na scenie**
    - Stosuje się liczne **sztuczki i uproszczenia**



Źródło obrazu:
Wikipedia

KOLIZJE

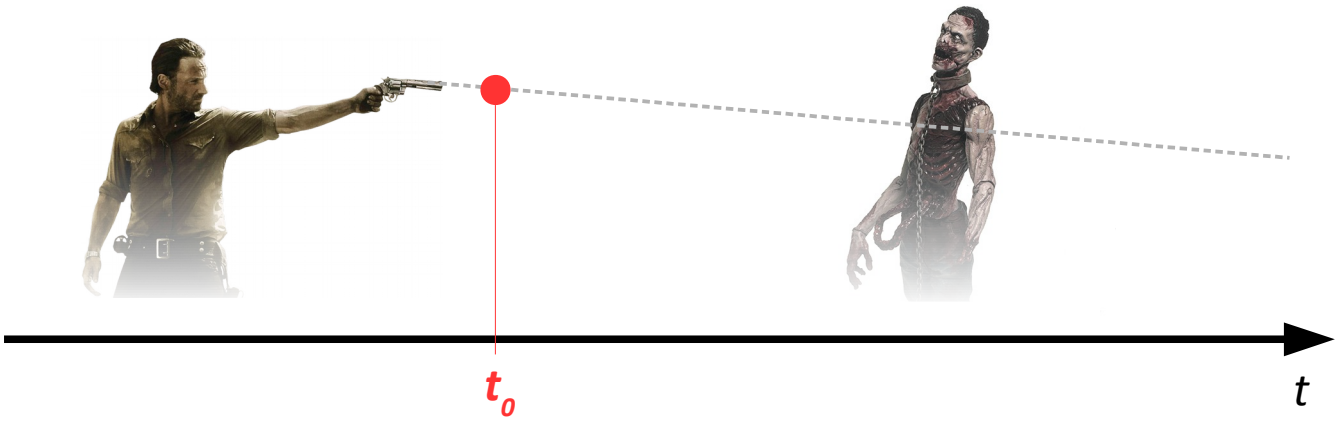
- **Problem kolizji** jest złożony i można rozpatrywać go w następujących kategoriach:
  - **Detekcja** kolizji
    - Czy **w danej chwili** zachodzi kolizja?
      - **Dyskretna detekcja kolizji** (*DCD*, ang. *Discrete Collision Detection*)
    - Czy na przestrzeni czasu doszło do kolizji i jeśli tak, to **kiedy** i **gdzie**?
      - **Ciągła detekcja kolizji** (*CCD*, ang. *Continuous Collision Detection*)
  - **Reakcja** na kolizje
    - Jak obiekty które kolidują powinny się **zachować**?

Źródło obrazu:
Rockstar Games

KOLIZJE

- **Dyskretna detekcja kolizji** (*DCD*) jest sprawdzeniem, czy **w danym momencie** zachodzi kolizja
  - Polega więc wyłącznie na sprawdzeniu czy bryły kolizji dla istniejących obiektów mają w obecnym stanie **części wspólne**

Sprawdzanie kolizji odbywa się w kolejnych **krokach aktualizacji stanu świata gry**.

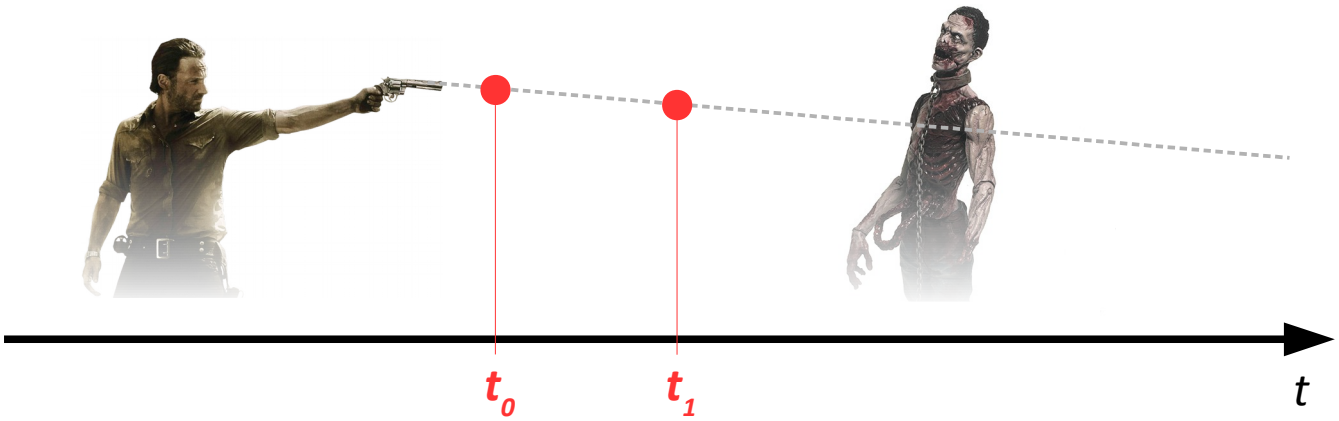


Źródło obrazów:
AMC

KOLIZJE

- **Dyskretna detekcja kolizji** (*DCD*) jest sprawdzeniem, czy **w danym momencie** zachodzi kolizja
  - Polega więc wyłącznie na sprawdzeniu czy bryły kolizji dla istniejących obiektów mają w obecnym stanie **części wspólne**

Sprawdzanie kolizji odbywa się w kolejnych **krokach aktualizacji stanu świata gry**.

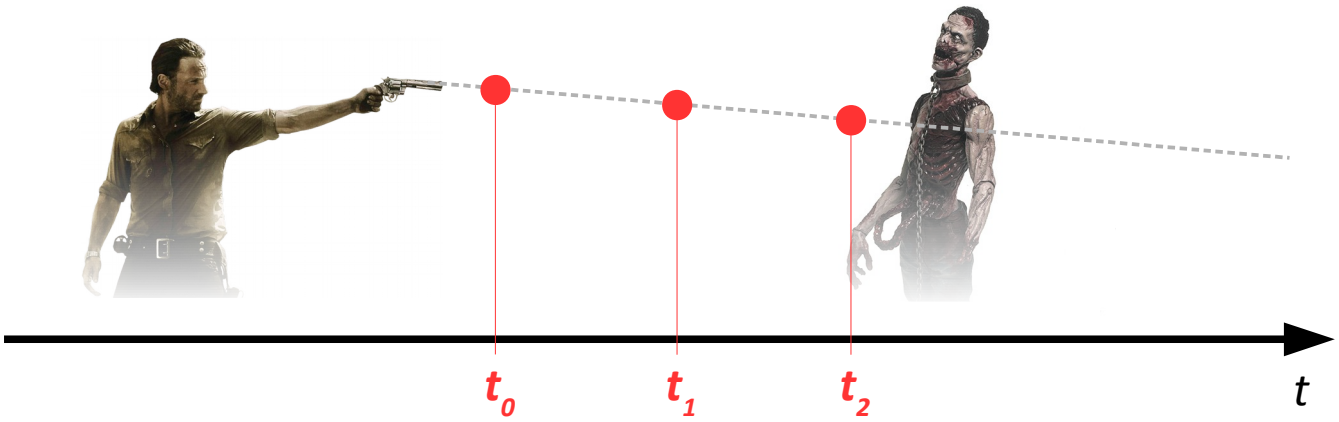


Źródło obrazów:
AMC

KOLIZJE

- **Dyskretna detekcja kolizji** (*DCD*) jest sprawdzeniem, czy **w danym momencie** zachodzi kolizja
  - Polega więc wyłącznie na sprawdzeniu czy bryły kolizji dla istniejących obiektów mają w obecnym stanie **części wspólne**

Sprawdzanie kolizji odbywa się w kolejnych **krokach aktualizacji stanu świata gry**.



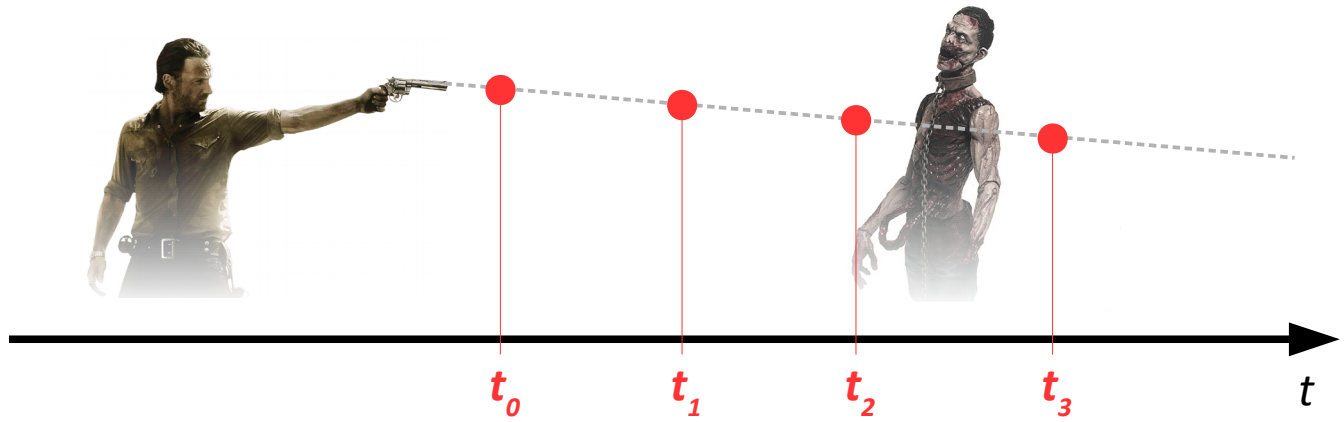
Źródło obrazów:
AMC

KOLIZJE

- **Dyskretna detekcja kolizji** (*DCD*) jest sprawdzeniem, czy **w danym momencie** zachodzi kolizja
  - Polega więc wyłącznie na sprawdzeniu czy bryły kolizji dla istniejących obiektów mają w obecnym stanie **części wspólne**

Co się stanie gdy szybko poruszający się obiekt w kolejnym kroku znajdzie się już za obiektem, z którym powinien mieć kolizję?

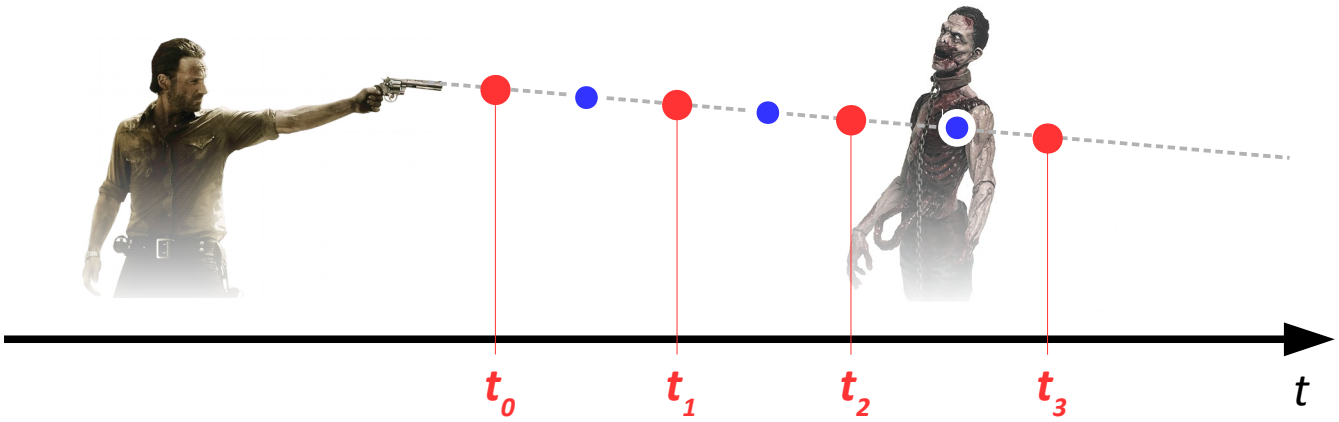
Takie zjawisko nosi nazwę ***tunelowania***.



Źródło obrazów:
AMC

KOLIZJE

- **Dyskretna detekcja kolizji** (*DCD*) jest sprawdzeniem, czy **w danym momencie** zachodzi kolizja
  - Polega więc wyłącznie na sprawdzeniu czy bryły kolizji dla istniejących obiektów mają w obecnym stanie **części wspólne**



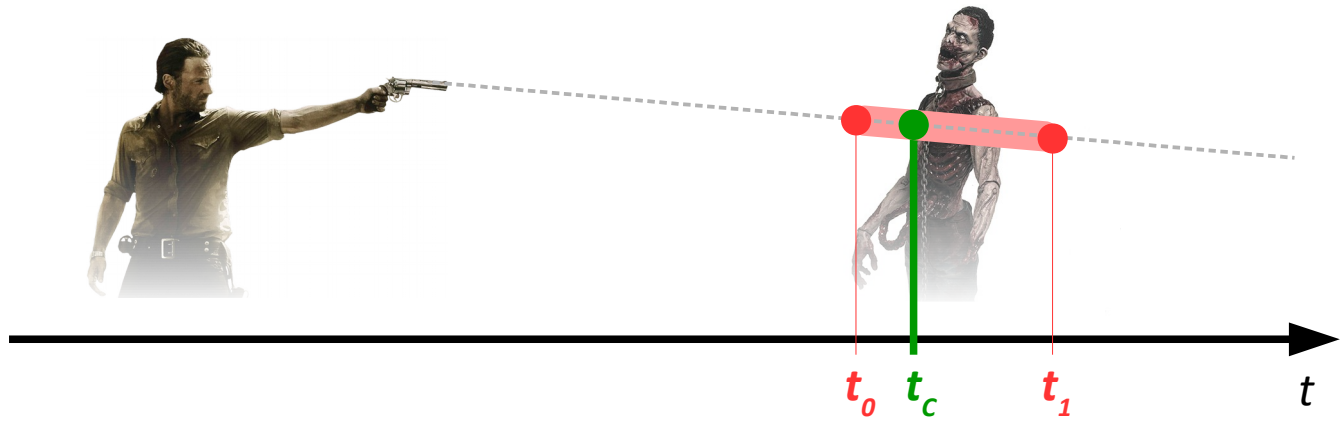
Jak poradzić sobie z tym problemem?

Wprowadzając pośrednie, **dodatkowe *podkroki***.
Ale jeśli i to nie wystarczy?
Wprowadzać kolejne podziały?

Źródło obrazów:
AMC

KOLIZJE

- **Ciągła detekcja kolizji** (*CCD*) jest sprawdzeniem, **w którym momencie i miejscu** zaszła kolizja pomiędzy dwoma krokami
  - Polega na znalezieniu przecięcia pomiędzy **bryłą otaczającą bryłę kolizji w dwóch stanach** (początkowym i końcowym), a obiektem na scenie
    - Takie tworzenie bryły otaczającej i sprawdzanie jej przecięcia z obiektami na scenie określa się mianem *sweeping*



Ciągła detekcja kolizji jest bardziej kosztowna od dyskretnej, ale tańsza od wprowadzania licznych kolejnych *podkroków*.

Źródło obrazów:
AMC

DYSKRETNA DETEKCJA KOLIZJI

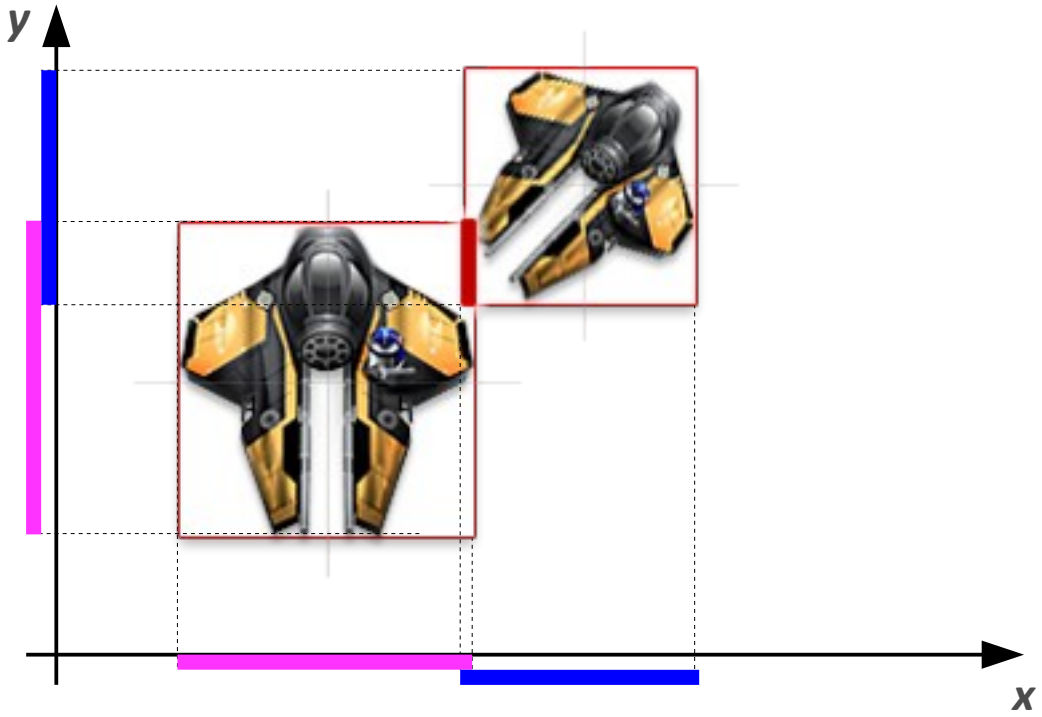
Aby zdefiniować *AABB*, wystarczy określić **najmniejszą i największą wartość każdej ze współrzędnych** – czyli określić *rozciągłość* pudełka na każdej z osi.

Kolizja między trójwymiarowymi pudełkami A i B zachodzi gdy spełnione są **wszystkie** następujące warunki:

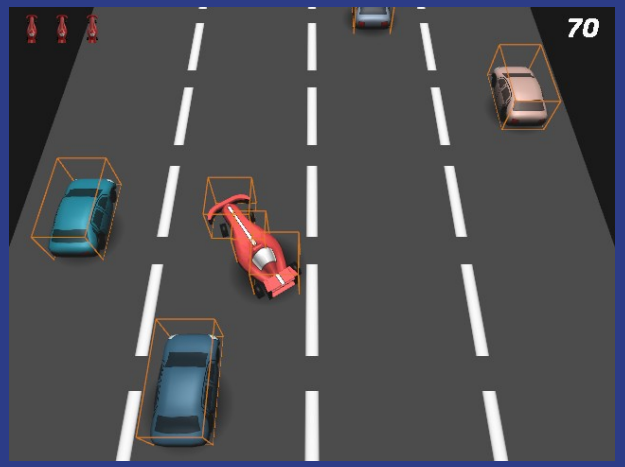
$$Ax_{MIN} < Bx_{MAX} \textbf{ AND } Bx_{MIN} < Ax_{MAX}$$
$$Ay_{MIN} < By_{MAX} \textbf{ AND } By_{MIN} < Ay_{MAX}$$
$$Az_{MIN} < Bz_{MAX} \textbf{ AND } Bz_{MIN} < Az_{MAX}$$

Źródło obrazu:
Starling Wiki

- Metoda **AABB** (ang. *Axis-Aligned Bounding Boxes*)
  - Polega na opisaniu kształtu bryły za pomocą prostopadłościanu (lub prostokąta dla problemu *2D*), którego **ściany są równoległe do osi układu współrzędnych**
    - Takie *pudełko* powinno być **jak najmniejsze**, jednocześnie zawierając w sobie **całą pierwotną bryłę**
  - Sprawdzenie czy pudełka mają **część wspólną** jest **mało kosztowne**



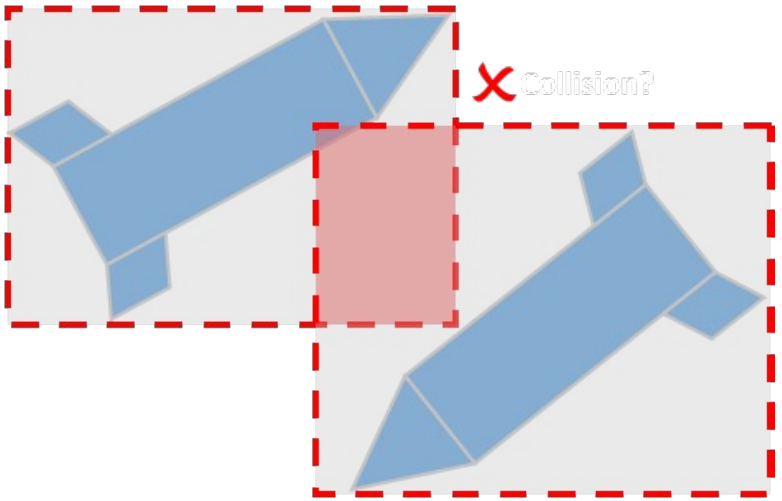
DYSKRETNA DETEKCJA KOLIZJI



Zależnie od kształtu brył i ich ułożenia na scenie, mogą one być przybliżone za pomocą więcej niż jednego *AABB*.

Źródła obrazów:
Urge to Merge
DataGenetics.com

- Metoda **AABB** (ang. *Axis-Aligned Bounding Boxes*)
  - Często dla brył które nie są zorientowanymi zgodnie z osiami układu współrzędnych sześcianami, *AABB* potrafi **wykryć kolizje gdy faktycznie do nich nie dochodzi**
  - Dlatego wykorzystuje się tę metodę często tylko jako najbardziej zgrubny, **pierwszy krok** pozwalający stwierdzić czy warto dokładniej rozpatrywać potencjalną kolizję

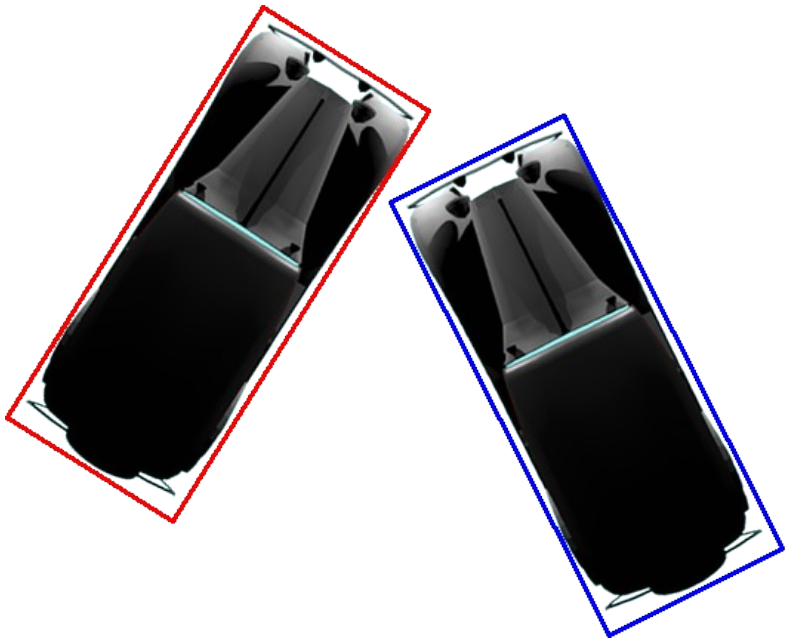


DYSKRETNA DETEKCJA KOLIZJI

Bryły które kształtem nie przypominają prostopadłościanu można przybliżyć za pomocą **więcej niż jednego *OBB***

Źródło obrazu:
Toymaker.info
Euclideanspace.com

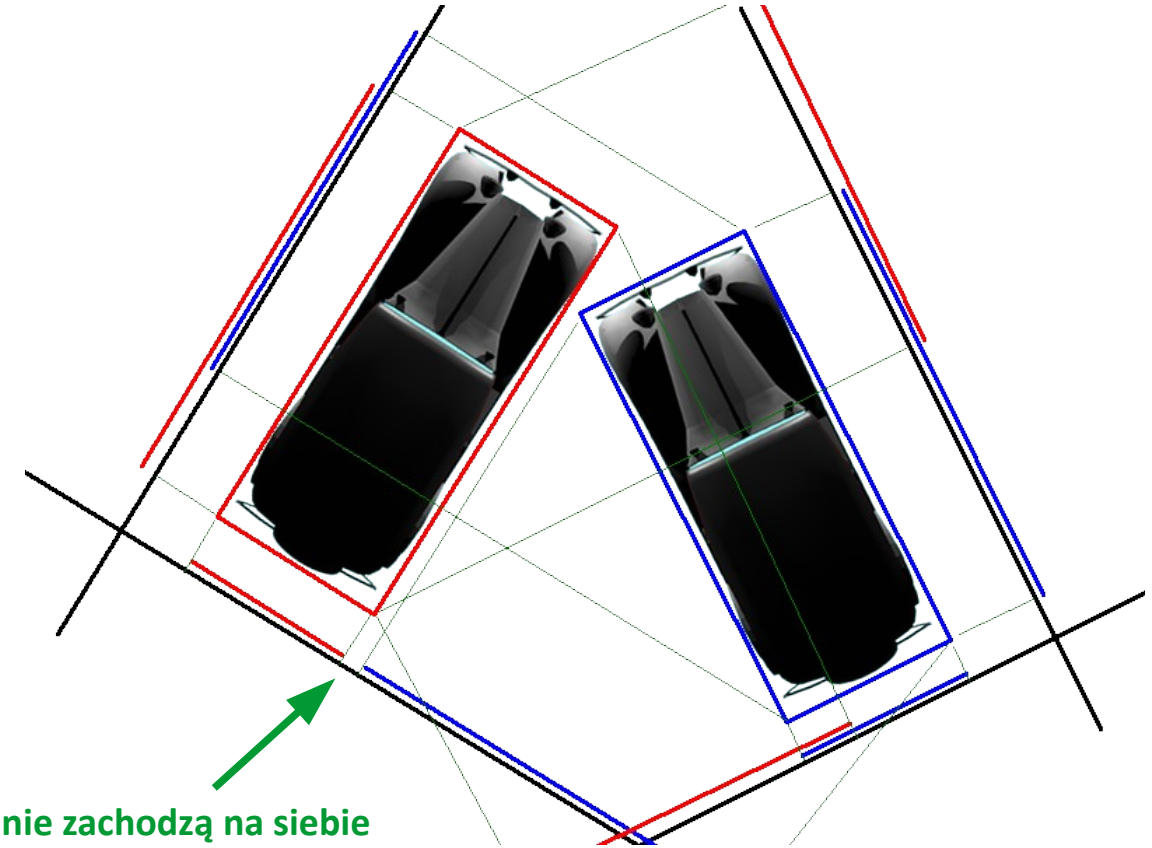
- Metoda **OBB** (ang. *Oriented Bounding Boxes*)
  - Pudełka (prostokąty lub prostopadłościany) są **orientowane** tak, by zapewnić **jak najlepsze dopasowanie do kształtu bryły** (ang. *best-fit boxes*)
    - Problem **fałszywych kolizji** znany z *AABB* jest znacząco **zredukowany** dla brył zbliżonych kształtem do prostopadłościanu



DYSKRETNA DETEKCJA KOLIZJI

- Metoda **OBB** (ang. *Oriented Bounding Boxes*)
  - Znajdujemy **rzuty każdego z prostopadłościanów** na każdą z **płaszczyzn równoległych** do ich ścian
  - Jeśli na którejkolwiek z płaszczyzn rzuty **nie mają części wspólnej**, to **nie ma kolizji**

Porównanie do świata rzeczywistego: jeśli obchodząc dwa prostopadłościenne obiekty dookoła zobaczymy chociaż z jednego miejsca, że jest pomiędzy nimi odstęp, to znaczy że **nie doszło do kolizji**.



Rzuty nie zachodzą na siebie

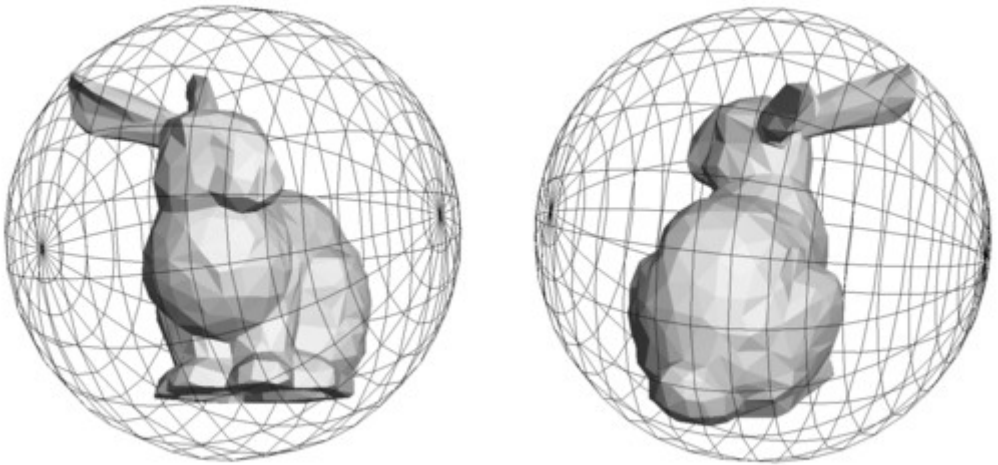
Źródło obrazu:
Euclideanspace.com

DYSKRETNA DETEKCJA KOLIZJI

Wystarczy określić **środek** kuli i jej **promień**. Jeśli dodatkowo założymy że kula zawsze ma środek w miejscu określonym przez pozycję danego obiektu, wystarczy pamiętać tylko promień.

Źródła obrazów:
Sharky's Blog
Mathforum.org

- Metoda **otaczającej kuli** (ang. *bounding sphere*)
  - Przybliża bryłę za pomocą **kuli** tak, by cała bryła była w niej **zawarta**



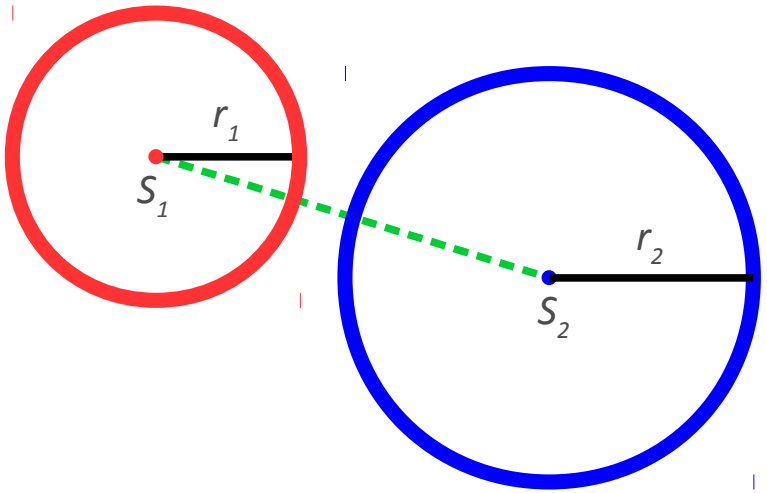
DYSKRETNA DETEKCJA KOLIZJI

- Metoda **otaczającej kuli** (ang. *bounding sphere*)
  - Przybliża bryłę za pomocą **kuli** tak, by cała bryła była w niej **zawarta**
  - Wykrycie kolizji następuje przez **porównanie odległości środków i długości promieni**

Przypadek 1:

Kule **nie mają części wspólnej**.

Odległość między środkami jest **większa** niż **suma promieni**.



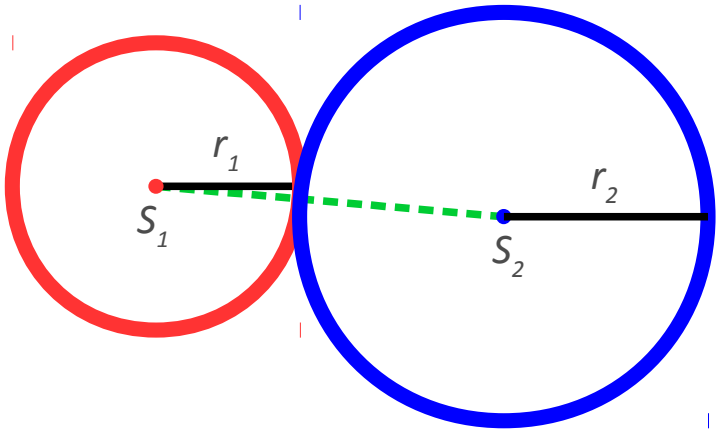
DYSKRETNA DETEKCJA KOLIZJI

Przypadek 2:

Kule **mają wspólny jeden punkt**.

Odległość między środkami jest **równa sumie promieni**.

- Metoda **otaczającej kuli** (ang. *bounding sphere*)
  - Przybliża bryłę za pomocą **kuli** tak, by cała bryła była w niej **zawarta**
  - Wykrycie kolizji następuje przez **porównanie odległości środków i długości promieni**



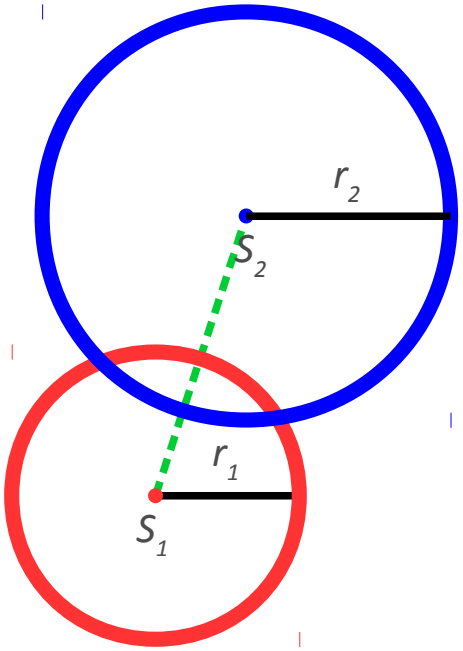
DYSKRETNA DETEKCJA KOLIZJI

- Metoda **otaczającej kuli** (ang. *bounding sphere*)
  - Przybliża bryłę za pomocą **kuli** tak, by cała bryła była w niej **zawarta**
  - Wykrycie kolizji następuje przez **porównanie odległości środków i długości promieni**

Przypadek 3:

Kule **mają część wspólną**.

Odległość między środkami jest **mniejsza** niż **suma długości promieni**.



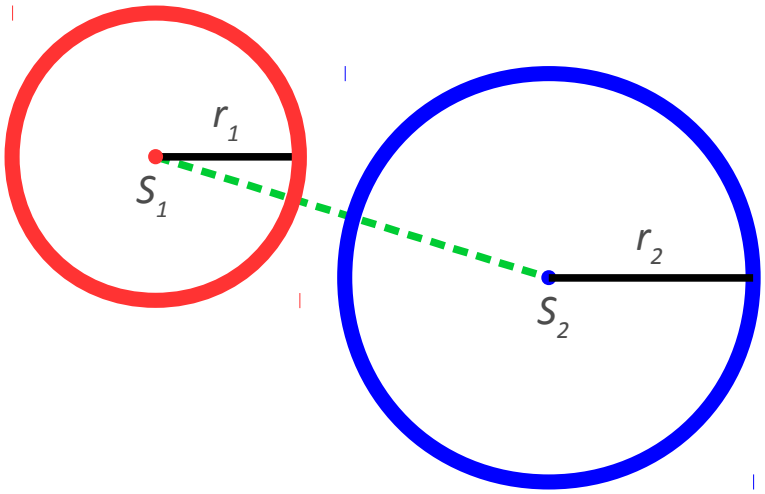
DYSKRETNA DETEKCJA KOLIZJI

- Metoda **otaczającej kuli** (ang. *bounding sphere*)
  - Przybliża bryłę za pomocą **kuli** tak, by cała bryła była w niej **zawarta**
  - Wykrycie kolizji następuje przez **porównanie odległości środków i długości promieni**

Przypadek 1:

Kule **nie mają części wspólnej**.

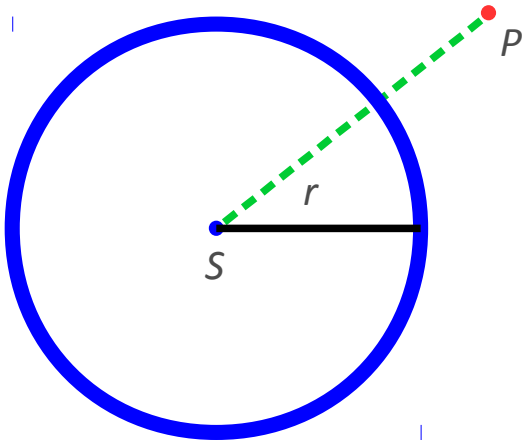
Odległość między środkami jest **większa** niż **suma promieni**.



DYSKRETNA DETEKCJA KOLIZJI

Analogicznie możemy dokonać sprawdzenia **kolizji punktu z kulą**, porównując odległość z długością promienia jednej kuli.

- Metoda **otaczającej kuli** (ang. *bounding sphere*)
  - Przybliża bryłę za pomocą **kuli** tak, by cała bryła była w niej **zawarta**
  - Wykrycie kolizji następuje przez **porównanie odległości środków i długości promieni**

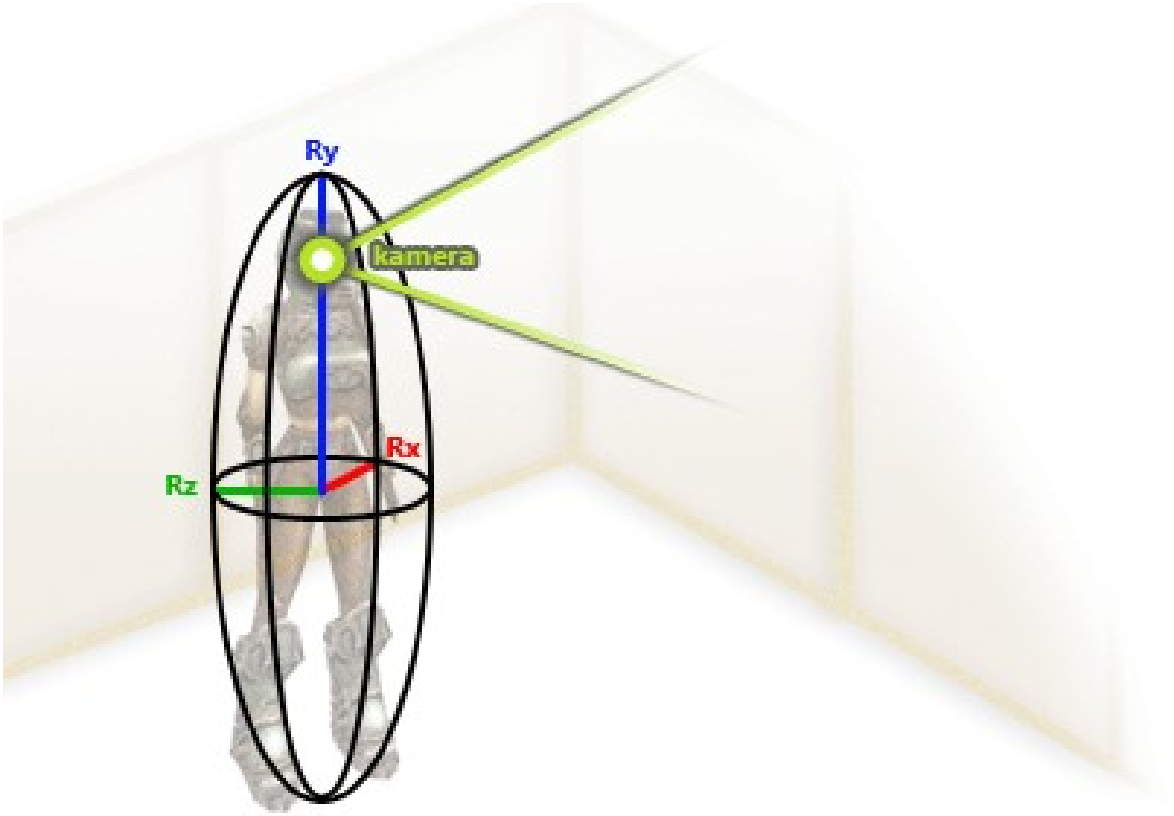


CIĄGŁA DETEKCJA KOLIZJI

- **Ciągła detekcja kolizji** jest bardzo przydatna tam, gdzie ważna jest naturalna reakcja na kolizje
  - Czyli tam, gdzie istotne jest *kiedy* nastąpiła kolizja i *gdzie*
  - Na przykład gdy **ograniczamy ruchy gracza ścianami**
- Przyjrzyjmy się technice ***swept sphere*** i ciągłym kolizjom **elipsoidy z wielokątami**

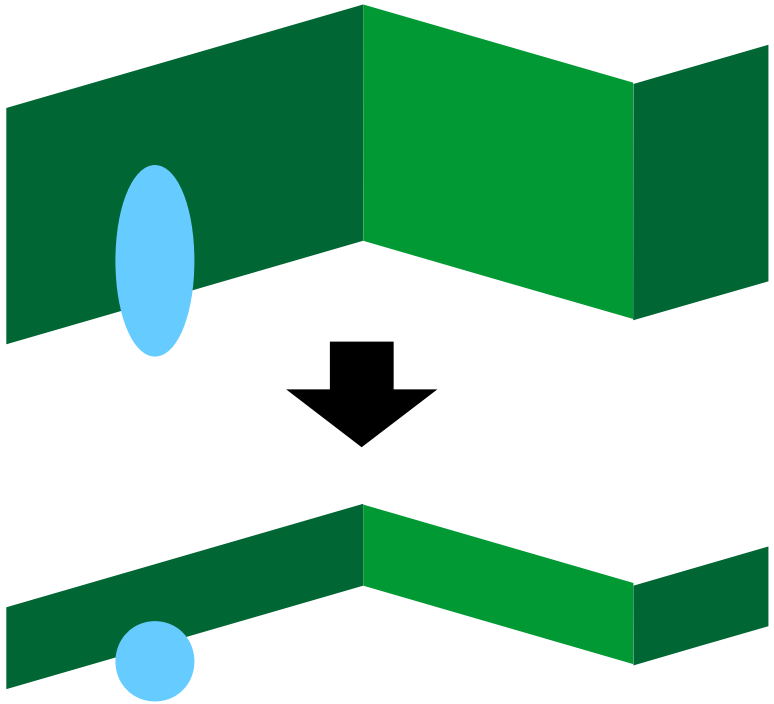
CIĄGŁA DETEKCJA KOLIZJI

- Kolizje **gracza ze ścianami**
  - Kształtem, który dobrze opisuje większość humanoidalnych postaci jest **elipsoida**
    - Elipsoida posiada **trzy promienie**: ***Rx***, ***Ry***, ***Rz***
    - Dla uproszczenia przyjmiemy, że ***Rx = Ry***, by pominąć kwestię obracania elipsoidy gdy gracz zmienia kierunek patrzenia



CIĄGŁA DETEKCJA KOLIZJI

- Kolizje **gracza ze ścianami**
  - Kształtem, który dobrze opisuje większość humanoidalnych postaci jest **elipsoida**
  - Jeśli **przeskalujemy** cały świat z mnożnikami ***(1/Rx, 1/Ry, 1/Rz)***, nie wpłynie to negatywnie na skuteczność wykrywania kolizji
    - Elipsoida stanie się wówczas **kulą**, co znacząco ułatwi dalsze obliczenia

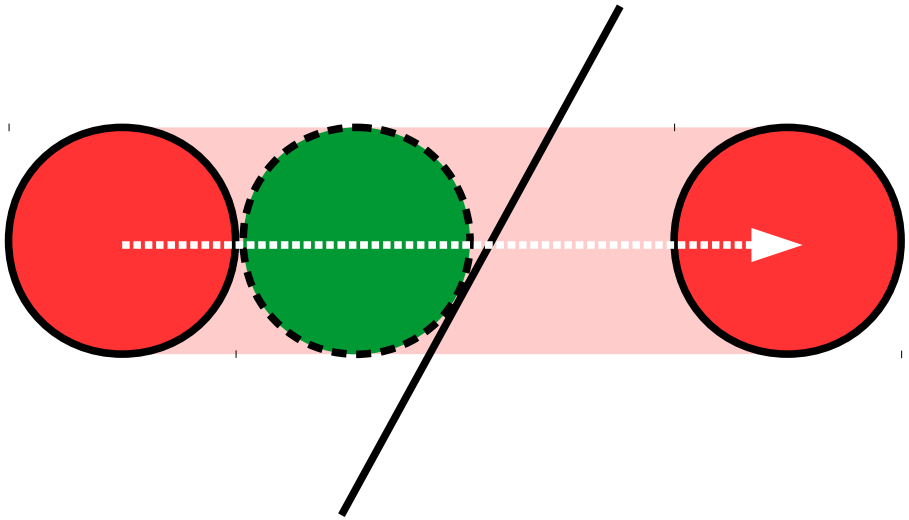


CIĄGŁA DETEKCJA KOLIZJI

- Kolizje **gracza ze ścianami**
  - Kształtem, który dobrze opisuje większość humanoidalnych postaci jest **elipsoida**
  - Jeśli **przeskalujemy** cały świat z mnożnikami ***(1/Rx, 1/Ry, 1/Rz)***, nie wpłynie to negatywnie na skuteczność wykrywania kolizji
  - Całość sprowadza się do wykrycia kolizji **pomiędzy kulą**, a **wielokątami**

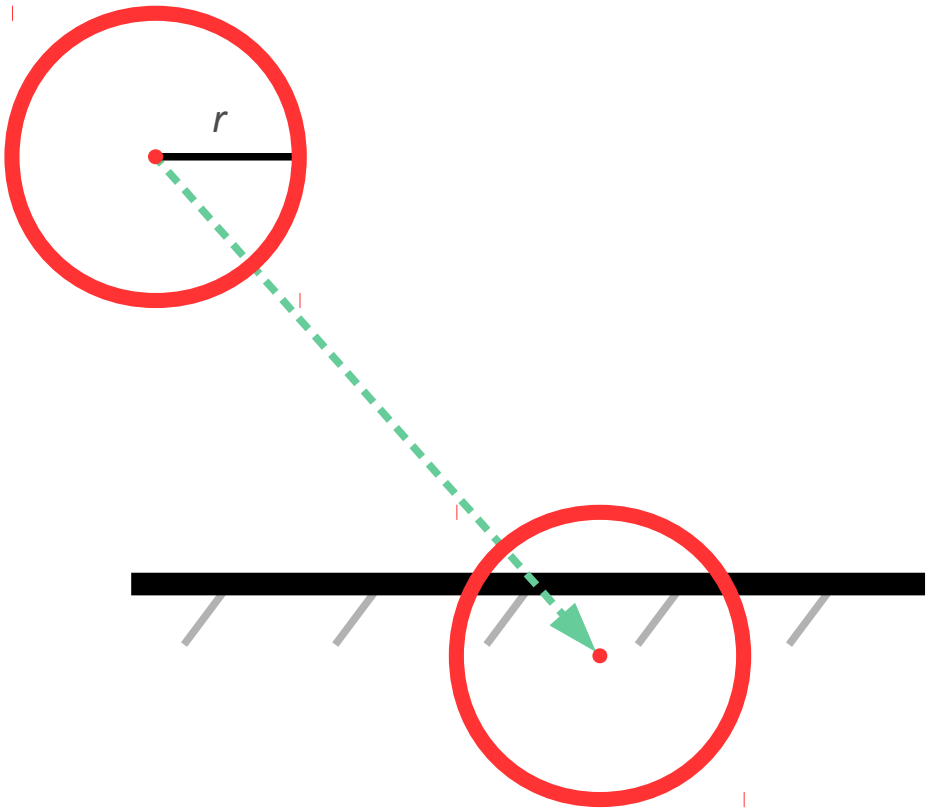
Należy rozpatrzyć **trzy przypadki** kolizji kuli z wielokątem:

Kula koliduje z **wnętrzem**
Kula koliduje z **krawędzią**
Kula koliduje z **wierzchołkiem**



REAKCJA NA KOLIZJE

- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany

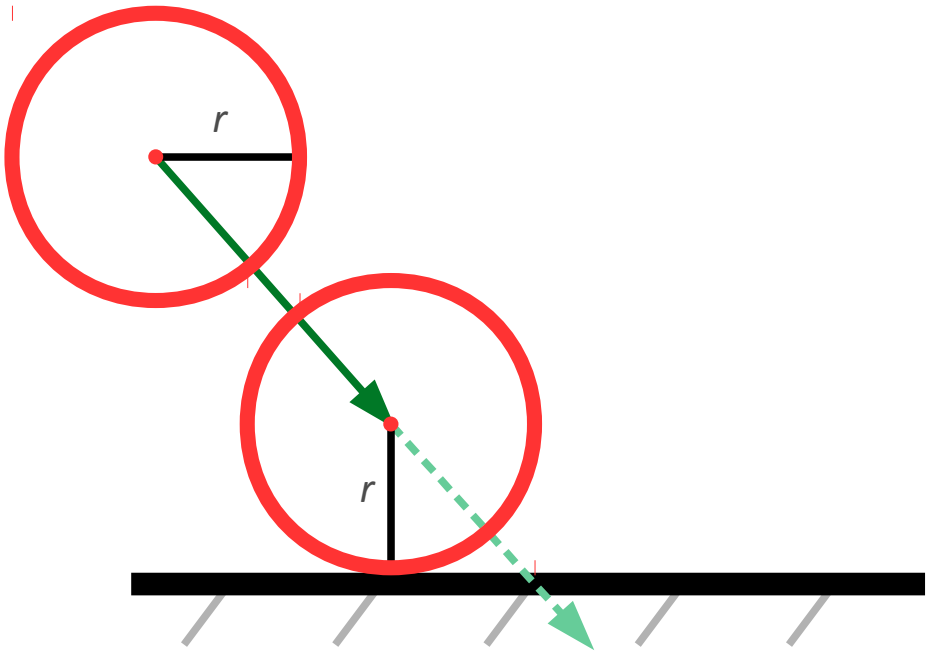


Z prędkości poruszania się i kierunku gracza wynika, że powinien przemieścić się o narysowany wektor.

Oczywistym jest, że należy uniemożliwić przeniknięcie przez ścianę.

REAKCJA NA KOLIZJE

- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany



Ruch gracza ostanie więc ograniczony tylko do tego momentu, w którym kula dotknie ściany.

Innymi słowy położenie środka kuli przemieści się wzdłuż wektora przemieszczenia do momentu, gdy jego odległość od ściany będzie równa promieniowi kuli.

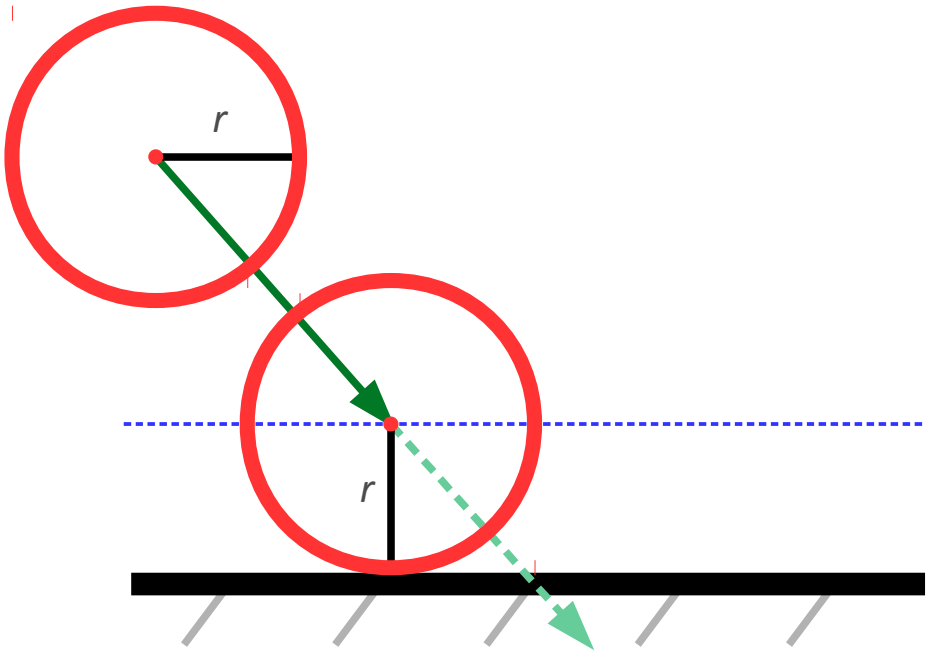
REAKCJA NA KOLIZJE

Jeśli zakończymy na tym ruch, gracz „przyklei się” do ściany.

Każda kolejna próba prouszenia się w kierunku choć trochę przybliżającym się do ściany nie wywoła żadnego ruchu.

Naturalnym zachowaniem byłoby prześlizgnięcie się wzdłuż ściany po kierunku równoległym do niej.

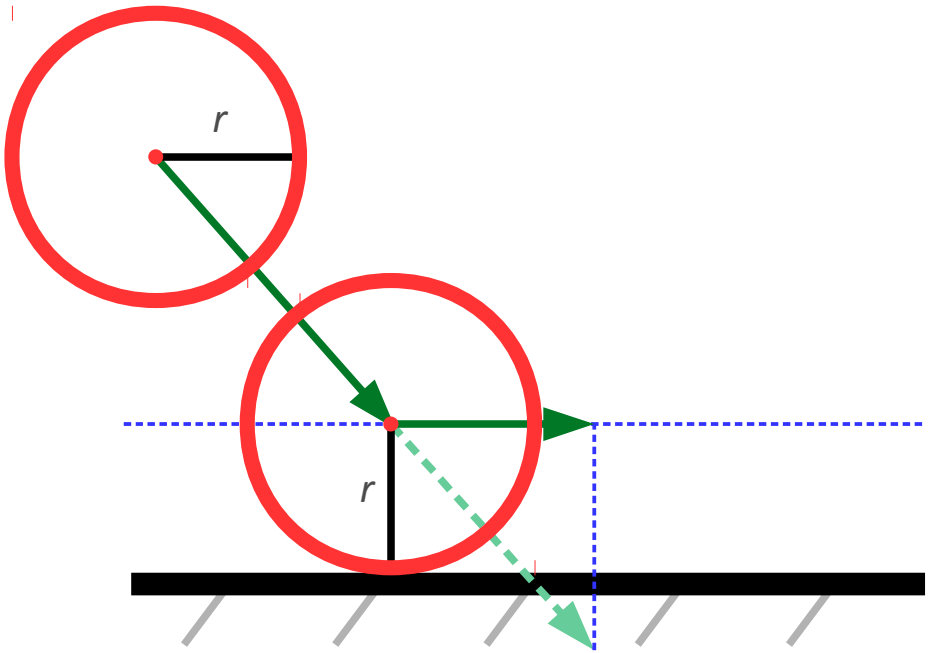
- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany



REAKCJA NA KOLIZJE

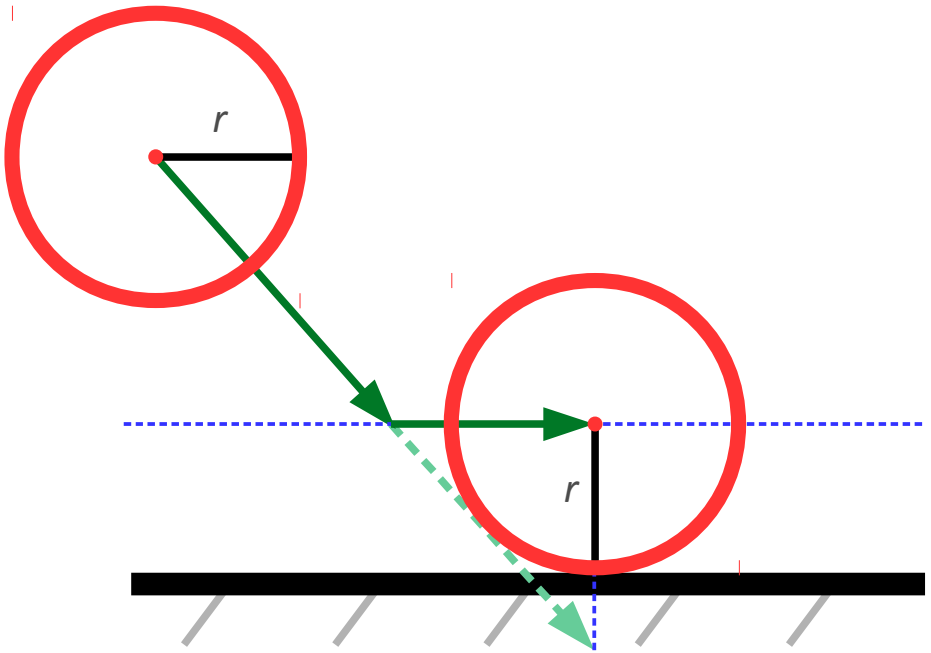
To, jak daleko gracz powinien się prześlizgnąć, określone jest przez rzut niewykorzystanej części pierwotnego wektora przemieszczenia na kierunek równoległy do ściany.

- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany



REAKCJA NA KOLIZJE

- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany



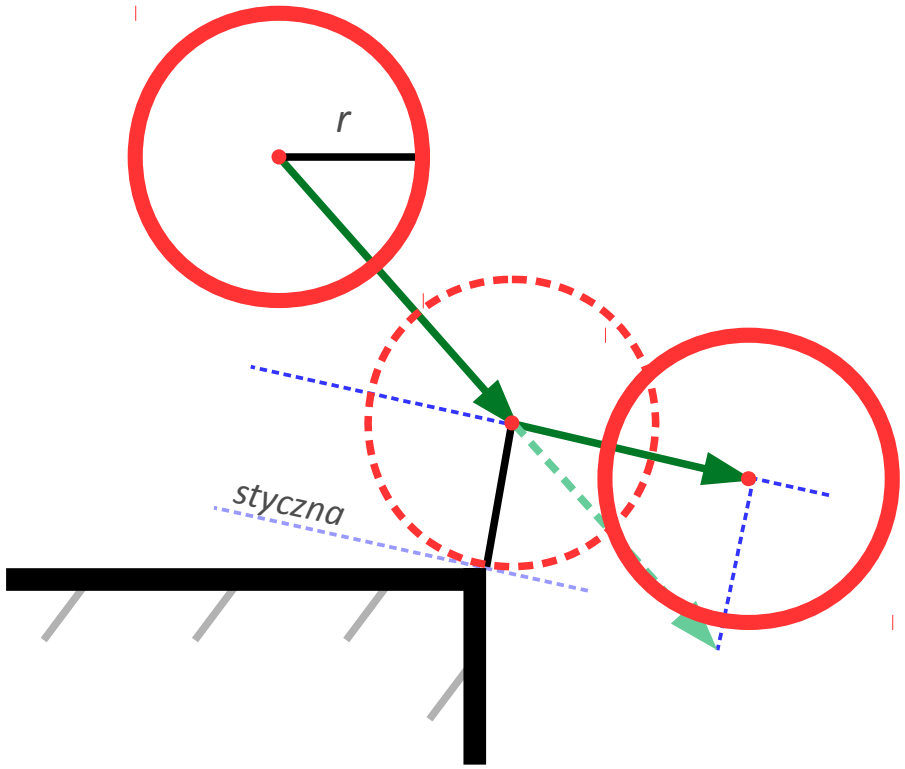
Tak zrealizowana reakcja na kolizję będzie dla gracza wygodna. Nie będzie skutkowała „przyklejaniem się” do ścian.

Jest to szczególnie ważne w wąskich korytarzach.

REAKCJA NA KOLIZJE

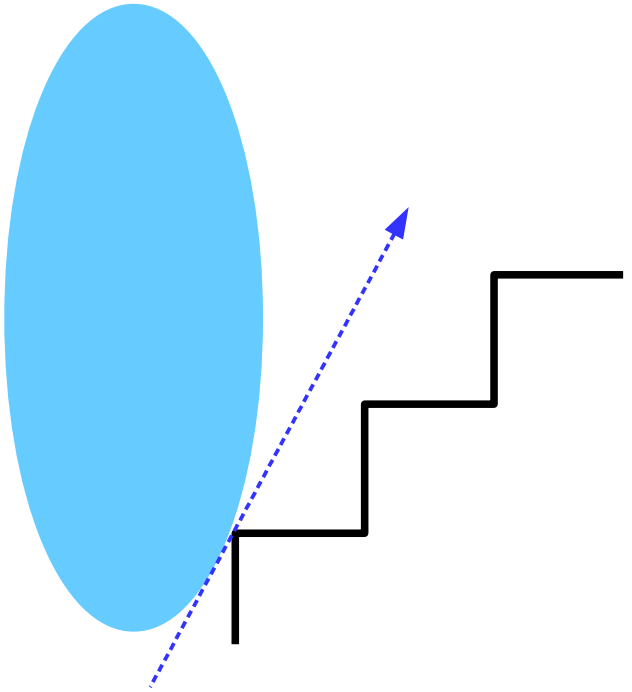
- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany

W przypadku gdy kolizja kuli następuje z krawędzią lub wierzchołkiem, dalsze przemieszczenie powinno odbyć się wzdłuż kierunku równoległego do **stycznej** do powierzchni kuli w punkcie wystąpienia kolizji.



REAKCJA NA KOLIZJE

- Kolizje **gracza ze ścianami**
  - Reakcja na zaistnienie kolizji powinna być możliwie **naturalna** z punktu widzenia obserwatora
  - Gracz powinien zawsze móc dojść **najbliżej** jak to tylko możliwe do ściany, **nie** powinien móc się w niej "**zagłębić**"
  - Gracz **nie** powinien się **blokować** po dotknięciu ściany

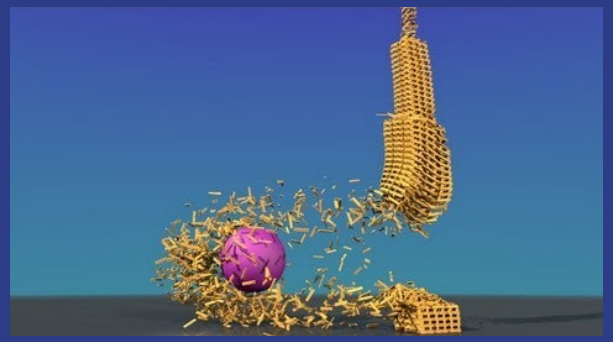


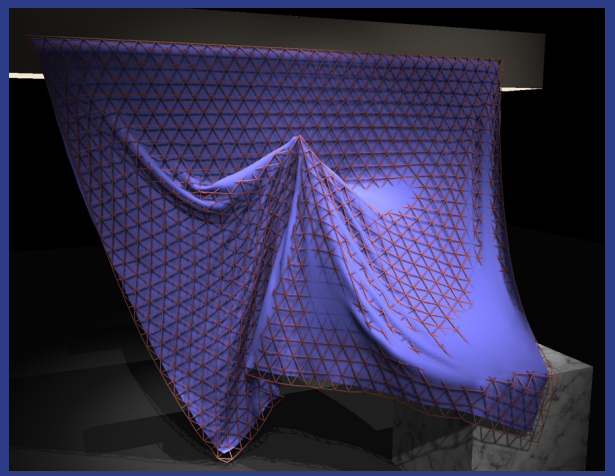
Poruszanie się wzdłuż stycznej dodatkowo umożliwia automatyczne pokonywanie niewielkich przeszkód terenowych, np. schodów.

KOLIZJE

- Porównywanie **każdego obiektu na scenie z każdym innym obiektem** w celu znalezienia kolizji byłoby **niesamowicie kosztowne**
  - Stosuje się wstępną **preselekcję** tylko tych obiektów, które mogą ze sobą kolidować
    - Np. pociski zazwyczaj w grach nie mogą kolidować z innymi pociskami
  - Odrzuca się obiekty na podstawie **organizacji przestrzennej**
    - Np. *octree*
  - Stosuje się **kilka poziomów szczegółowości** wykrywania kolizji
    - Np. najpierw *AABB*, potem jeśli jest szansa na kolizję to *AABB* z większą szczegółowością, potem jeśli dalej nie udało się odrzucić możliwości występienia kolizji to najbardziej dokładna metoda – np. *trójkąt vs. trójkąt*

SILNIK FIZYCZNY





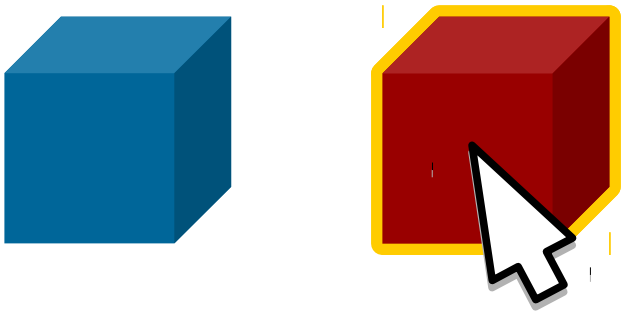
Źródła obrazów:
Triton TV
NVIDIA

- Obsługę kolizji w rozbudowanym świecie warto powierzyć sprawdzonemu **silnikowi fizycznemu**
  - Podstawowym jego zadaniem jest kontrola **pozy** brył, które opisują obiekty na scenie, w **reakcji** na zachodzące kolizje
    - Fizyka **bryły sztywnej**
  - Niektóre oferują dodatkowo np. **symulację płynów** (ang. *fluid simulation*), **tkanin** (ang. *cloths*), **brył miękkich** (ang. *soft bodies*)
  - Przykładowe silniki:
    - **NVIDIA PhysX**
    - **Havok**
    - **Bullet**
    - **Newton Game Dynamics**
    - **Open Dynamics Engine**
    - ...i wiele innych

WYBÓR OBIEKTU NA EKRANIE

Wybór obiektu również jest przykładem **detekcji kolizji**.

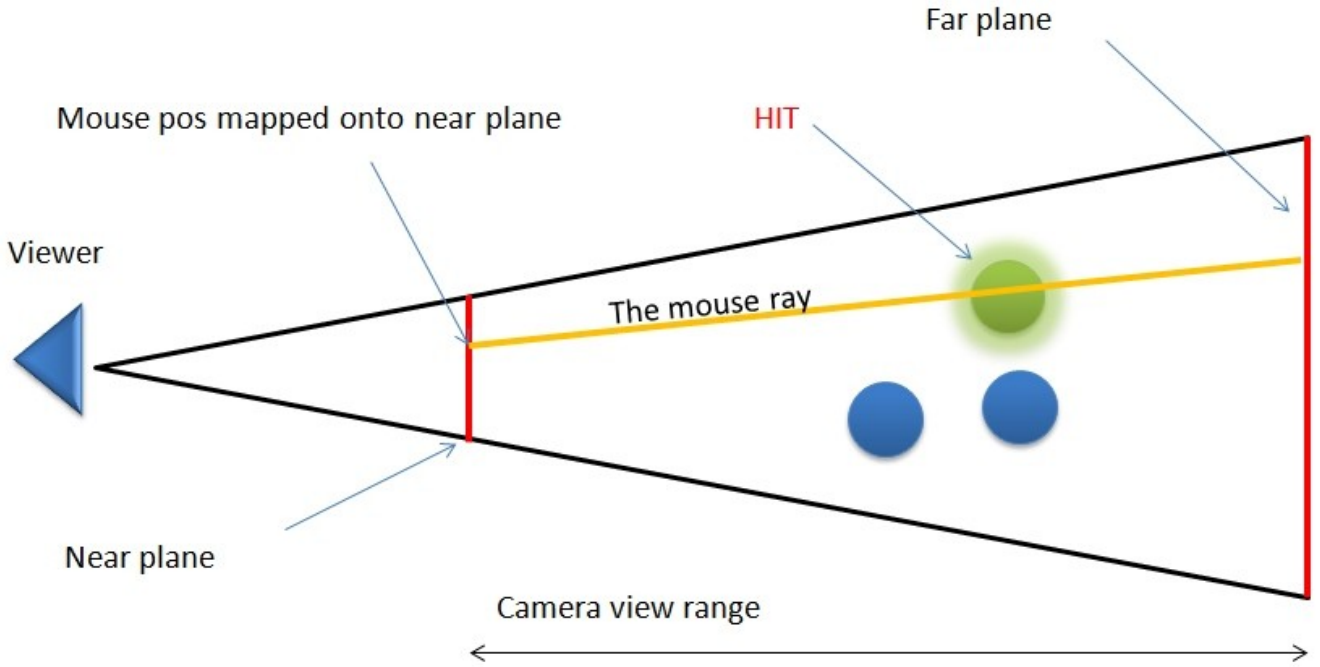
- Częstym problemem jest określenie **który** spośród wyświetlanych na ekranie obiektów został **wybrany** przez użytkownika
  - Problem nazywa się często mianem *picking*
  - Wybór może nastąpić przez kursor myszy lub np. środek ekranu w grze *FPP*
  - Są trzy popularne techniki:
    - **Śledzenie promienia** (ang. *ray casting*)
    - **Wsteczna projekcja** (ang. *unprojection*)
    - **Kodowanie kolorystyczne** (ang. *colour coding*)



WYBÓR OBIEKTU NA EKRANIE

Dokładność techniki zależy od stosowanej **metody wykrywania kolizji** *promienia* z obiektami. Może to prowadzić do problemów z precyzją wskazywania.

- **Śledzenie promienia** polega na wystrzeleniu promienia z pozycji kamery
  - Promień powinien mieć **kierunek** wyznaczony przez **położenie kursora myszy**
  - Szukamy najbliższej **kolizji z obiektem na scenie**



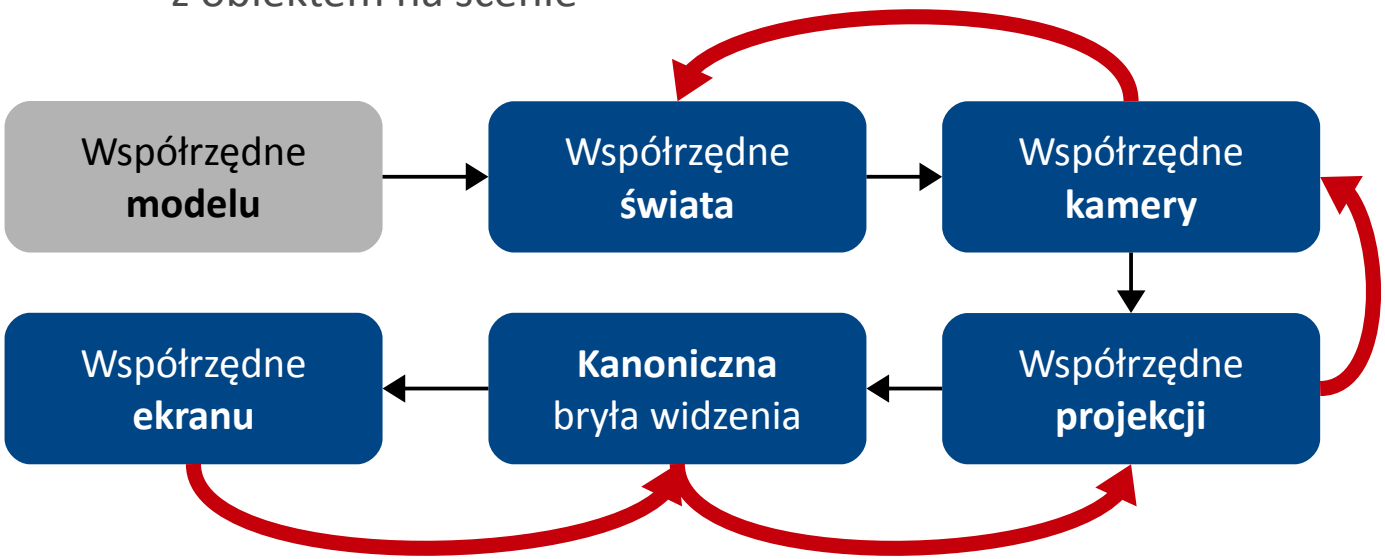
Źródło obrazu:
Bartłomiej Filipek, Code And Graphics

WYBÓR OBIEKTU NA EKRANIE

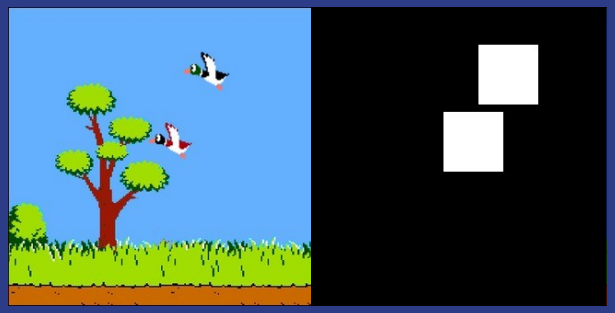
Dokładność tej techniki również zależy od stosowanej **metody wykrywania kolizji**, tym razem *punktu* z obiektami.

W implementacji opartej o *OpenGL* przydatna jest funkcja ***gluUnproject()***.

- **Projekcja wsteczna** (ang. *unprojection*) polega na uzyskaniu trójwymiarowych współrzędnych w **przestrzeni świata** na podstawie **współrzędnych ekranowych**
  - Jest to przekształcenie współrzędnych ekranowych przez **macierz odwrotną** do macierzy będącej złożeniem wszystkich zastosowanych transformacji ze współrzędnych świata do współrzędnych ekranowych
  - Konieczne jest odczytanie wartości z **bufora głębokości** w miejscu gdzie jest kursor myszy
  - Na koniec trzeba znaleźć **kolizję punktu** z obiektem na scenie



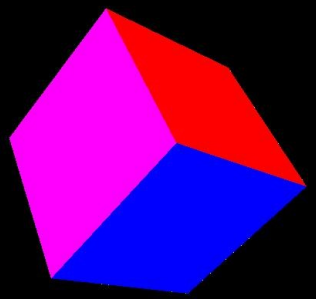
WYBÓR OBIEKTU NA EKRANIE



Zasada działania tej techniki zbliżona jest do sposobu funkcjonowania kontrolerów *Nintendo Zapper* z lat '80.

Źródło obrazów:
Nintendo
Pete's QBASIC

- **Kodowanie kolorystyczne** polega na wyrenderowaniu **dodatkowej**, nie pokazywanej użytkownikowi **klatki**
  - Dodatkowa klatka zawiera **tę samą scenę** i wykorzystuje **te same ustawienia widoku i projekcji**, co widoczna klatka
  - Dodatkowa klatka zawiera **każdy obiekt w innym, jednolitym kolorze**, bez oświetlenia i tekstur
    - **Kolor** jest **identyfikatorem** tego obiektu
  - Sprawdzany jest **kolor piksela** w miejscu, gdzie jest kolor myszy i na jego podstawie określany jest kliknięty obiekt
  - Metoda zazwyczaj ma **dokładność rzędu pojedynczych pikseli**



Zachodniopomorski
Uniwersytet
Technologiczny
w Szczecinie

http://bazyluk.net/dydaktyka

Bartosz Bazyluk

Scena i kolizje

Organizacja sceny. Techniki optymalizacji renderowania. Wykrywanie i reakcja na kolizje.

Wydział
Informatyki

Grafika Komputerowa i Wizualizacja, Informatyka S1, II Rok