Fizyka w modelowaniu i symulacjach komputerowych
Jacek Matulewski (e-mail: jacek@fizyka.umk.pl)
http://www.fizyka.umk.pl/~jacek/dydaktyka/modsym/

Symulacje komputerowe

Detekcja kolizji brył sztywnych

Wersja: 15 kwietnia 2010

Plan

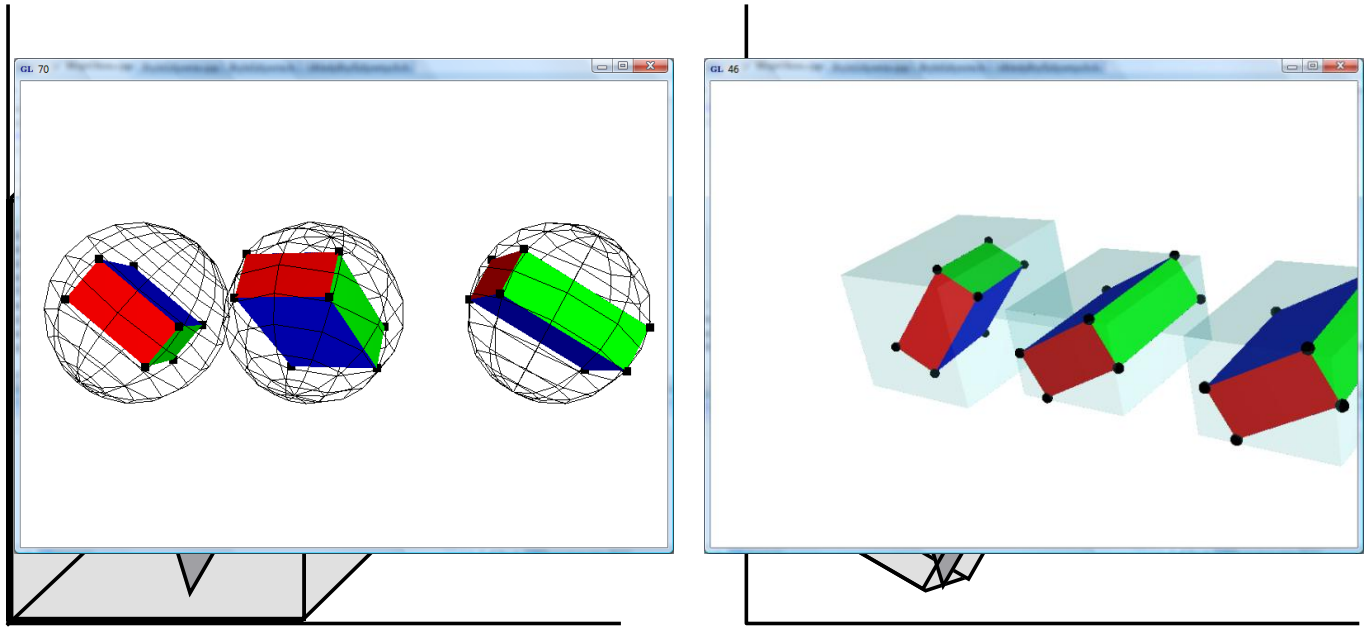
1. Wykrywanie zderzeń dowolnych brył, siatek (trójkąty)
2. Idee: otoczka wypukła, hierarchiczna dekompozycja
3. Obszary ograniczające: BS, AABB, OBB
4. Jak wykryć kolizję dwóch wypukłych brył sztywnych?
5. Wyznaczanie przekroju prostopadłościanów OBB
6. Wyznaczanie punktu styku oraz normalnej zderzenia
7. Metoda GJK
8. Reakcja na kolizję, czyli fizyka zderzenia brył sztywnych

Detekcja kolizji

- Otoczka wypukła (ang. *convex hull*) = najmniejszy wypukły fragment przestrzeni obejmujący otaczany obiekt.
- Wyobrażenie: gumowy balon, który najpierw napompowujemy, następnie wkładamy do niego model i wreszcie spuszczamy powietrze.
- Większość ogólnych metod, które pozwalają na znajdowanie odległości brył, ich przecięć itp. zakłada, że bryły są wypukłe.
- Algorytmy szukania - geometria obliczeniowa

Detekcja kolizji

Obszary ograniczające: BS, AABB, OBB

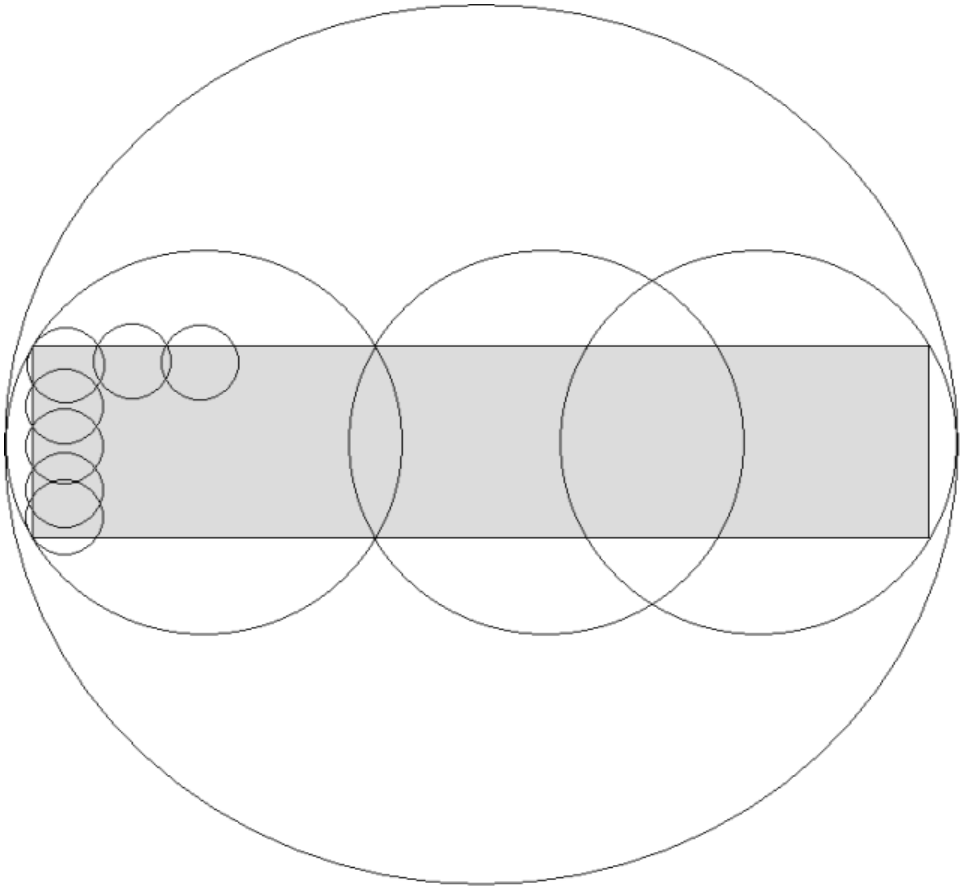


Axis-Aligned Bounding Box

Oriented Bounding Box

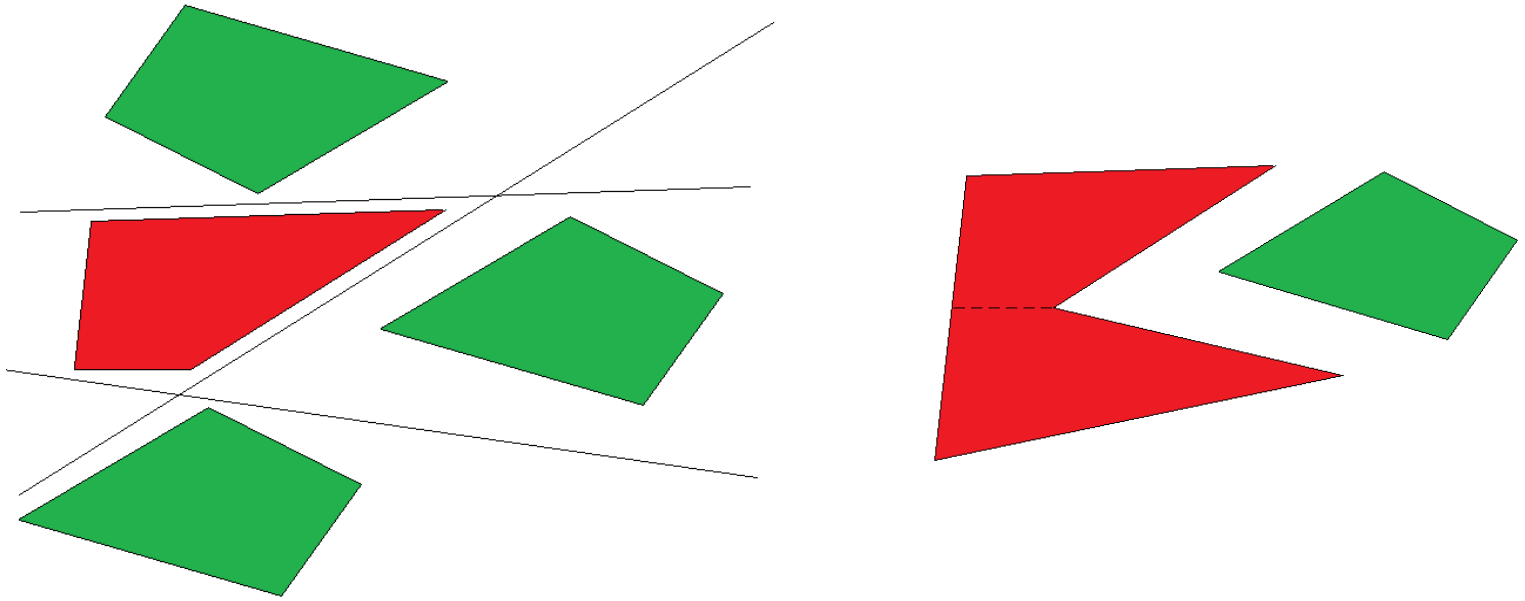
Detekcja kolizji

Hierarchiczna dekompozycja (na przykładzie BS)



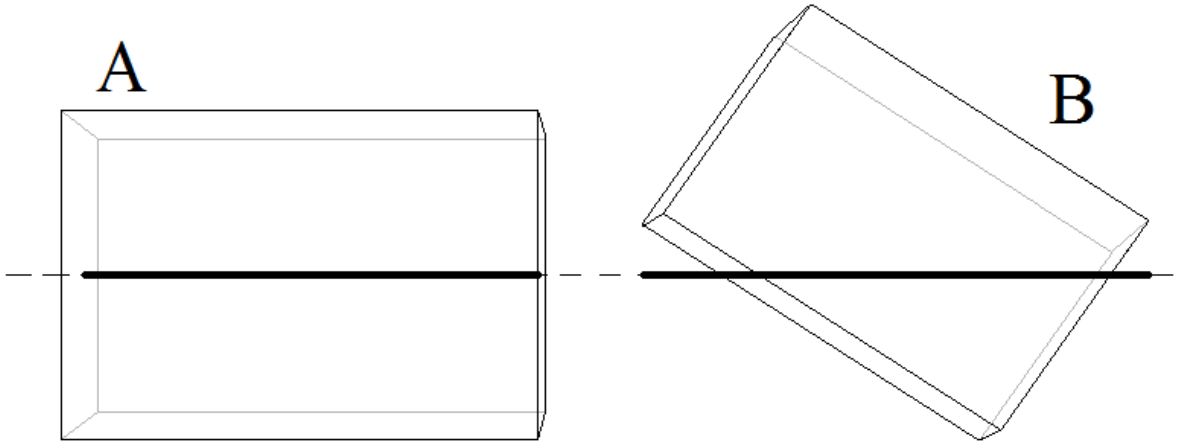
Detekcja kolizji

- Jak wykryć kolizję dwóch brył wypukłych?
- Twierdzenie SAT (*separating axis theorem*)



Detekcja kolizji

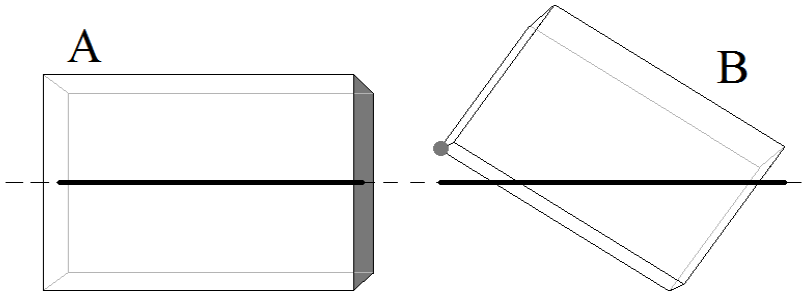
Oś rozdzielenia (oś na której widać separację)



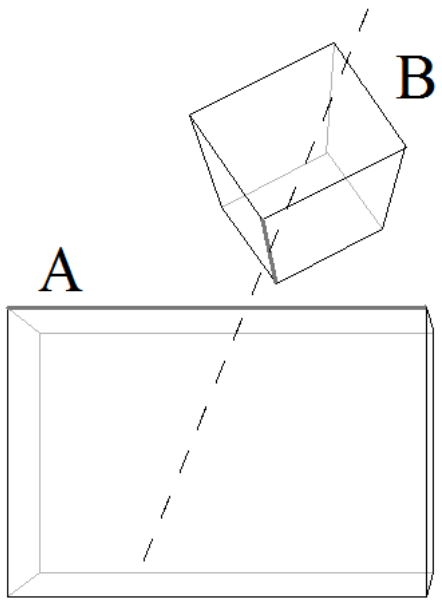
Kolizja gdy rzut odległości środków mas
na jakąś oś jest mniejszy
od sumy połówek rzutów całych brył na tę oś

Detekcja kolizji

- Dwa główne typy kolizji wielościanów (siatki):
  - wierzchołek-płaszczyzna



  - krawędź-krawędź



Detekcja kolizji

- Wzory dla dwóch prostopadłościanów:
  - rzut środka prostopadłościanu na oś $\vec{n} \cdot \vec{R}_{śm}$
  - osiem wierzchołków (w układzie odniesienia modelu):

$$\vec{r}_{\pm,\pm,\pm} = \pm\frac{a}{2}\vec{i} \pm \frac{b}{2}\vec{j} \pm \frac{c}{2}\vec{k}'$$

  - rzuty wierzchołków na testowaną oś:

$$\vec{n} \cdot \vec{r}_{\pm,\pm,\pm} = \pm\frac{a}{2}\,\vec{n} \cdot \vec{i} \pm \frac{b}{2}\,\vec{n} \cdot \vec{j} \pm \frac{c}{2}\,\vec{n} \cdot \vec{k}'$$

  - wybór maks. i min. spośród ośmiu rzutów wierzch.

$$\vec{n} \cdot \vec{R}_{śm} \pm \left(\frac{a}{2}\left|\vec{n} \cdot \vec{i}\right| + \frac{b}{2}\left|\vec{n} \cdot \vec{j}\right| + \frac{c}{2}\left|\vec{n} \cdot \vec{k}'\right|\right)$$

Detekcja kolizji

- Wzory dla dwóch prostopadłościanów:
  - ograniczenie ilości sprawdzanych osi (dla dwóch prostopadłościanów – 15 osi):

    Ax, Ay, Az, Bx, By, Bz - wierzchołek-płaszczyzna (osie prostopadłe do uderzanych ścian wielościanu)

    Ax x Bx, Ax x By, Ax x Bz, itd. - krawędź-krawędź (osie prostopadłe do obu krawędzi obu wielościanów)

Detekcja kolizji

- Twierdzenie SAT stosowane jest w przypadku obszarów ograniczających OBB
- Testy AABB mogą być rozumiane jako szczególny przypadek twierdzenia SAT (tylko trzy osie)

Detekcja kolizji

- Naiwna (czytelna) implementacja SAT dla OBB:

```
void ObliczRzutProstopadloscianuNaProsta(Prostopadloscian* p,Wektor n,double& min,
                                         double& max,bool nastepnePolozenie) const
{
   Macierz macierzObrotu;
   if(!nastepnePolozenie) macierzObrotu=p->MacierzObrotu();
   else macierzObrotu=p->NastepnaMacierzObrotu();
   Wektor osX=macierzObrotu.KolumnaX();
   Wektor osY=macierzObrotu.KolumnaY();
   Wektor osZ=macierzObrotu.KolumnaZ();

   Wektor polowaRozmiaru=p->Rozmiar()/2;
   double polowaDlugosciRzutu =
      polowaRozmiaru.X*fabs(n*osX)+polowaRozmiaru.Y*fabs(n*osY)+polowaRozmiaru.Z*fabs(n*osZ);
   double rzutSrodkaMasy;
   if(!nastepnePolozenie) rzutSrodkaMasy=n*p->PolozenieSrodkaMasy();
   else rzutSrodkaMasy=n*p->NastepnePolozenieSrodkaMasy();
   min=rzutSrodkaMasy-polowaDlugosciRzutu;
   max=rzutSrodkaMasy+polowaDlugosciRzutu;
}

bool CzyOdcinkiNaProstejPokrywajaSie(double minA,double maxA,double minB,double maxB) const
{
   return !((maxA < minB) || (minA > maxB));
}
```

Detekcja kolizji

- Naiwna (czytelna) implementacja SAT dla OBB:

```
bool CzyRzutyProstopadloscianowNaOsNakladajaSie(Wektor n,Prostopadloscian*
pA,Prostopadloscian* pB,bool nastepnePolozenie) const
{
   double minA,maxA,minB,maxB;
   ObliczRzutProstopadloscianuNaProsta(pA,n,minA,maxA,nastepnePolozenie);
   ObliczRzutProstopadloscianuNaProsta(pB,n,minB,maxB,nastepnePolozenie);
   return CzyOdcinkiNaProstejPokrywajaSie(minA,maxA,minB,maxB);
}
```

Detekcja kolizji

- Naiwna (czytelna) implementacja SAT dla OBB:

```
Wektor OsRzutowania(int indeks,Prostopadloscian* pA,Prostopadloscian* pB,
                    bool nastepnePolozenie) const
{
   Macierz macierzObrotuA,macierzObrotuB;
   if(!nastepnePolozenie)
   {
      macierzObrotuA=pA->MacierzObrotu();
      macierzObrotuB=pB->MacierzObrotu();
   }
   else
   {
      macierzObrotuA=pA->NastepnaMacierzObrotu();
      macierzObrotuB=pB->NastepnaMacierzObrotu();
   }

   switch(indeks)
   {
      ...
   }
}
```

Detekcja kolizji

- Naiwna (czytelna) implementacja SAT dla OBB:

```
Wektor OsRzutowania(int indeks,Prostopadloscian* pA,Prostopadloscian* pB,
                    bool nastepnePolozenie) const
{
   ...

   switch(indeks)
   {
      //A
      case 1: return macierzObrotuA.KolumnaX(); break;
      case 2: return macierzObrotuA.KolumnaY(); break;
      case 3: return macierzObrotuA.KolumnaZ(); break;
      //B
      case 4: return macierzObrotuB.KolumnaX(); break;
      case 5: return macierzObrotuB.KolumnaY(); break;
      case 6: return macierzObrotuB.KolumnaZ(); break;
      //iloczyny wektorowe AxB
      case 7: return macierzObrotuA.KolumnaX()^macierzObrotuB.KolumnaX(); break;
      case 8: return macierzObrotuA.KolumnaX()^macierzObrotuB.KolumnaY(); break;
      case 9: return macierzObrotuA.KolumnaX()^macierzObrotuB.KolumnaZ(); break;
      case 10: return macierzObrotuA.KolumnaY()^macierzObrotuB.KolumnaX(); break;
      ...
      case 15: return macierzObrotuA.KolumnaZ()^macierzObrotuB.KolumnaZ(); break;
      default: return Wektor::Zero(); break;
    }
}
```

Detekcja kolizji

- Naiwna (czytelna) implementacja SAT dla OBB:

```
bool TestNakrywaniaDwochProstopadloscianow(Prostopadloscian* pA,Prostopadloscian* pB,bool
nastepnePolozenie) const
{
   for(int i=1;i<=15;++i)
      if(!CzyRzutyProstopadloscianowNaOsNakladajaSie(
         OsRzutowania(i,pA,pB, nastepnePolozenie).Unormowany(),pA,pB), nastepnePolozenie)
            return false;
   return true;
}
```

Detekcja kolizji

- Optymalizacje
  - przeprowadzenie testów w układzie lokalnym A

$$R'_B = R_B R_A^{-1} \qquad \vec{R}'_{śm,B} = \vec{R}_{śm,B} - \vec{R}_{śm,A}$$

  - rozpisanie 15 konkretnych przypadków, co pozwala na uniknięcie wielu niepotrzebnych obliczeń

Detekcja kolizji

- Optymalizacje
  - Zaczynamy od pytania o to, czy długość połowy rzutów obu prostopadłościanów jest mniejsza od odległości środków tych prostopadłościanów?

$$(\vec{n}\cdot\vec{r}_A)_{max}+(\vec{n}\cdot\vec{r}_B)_{max}<\left|\vec{n}\cdot\left(\vec{R}_{śm,B}-\vec{R}_{śm,A}\right)\right|$$

połowy długości rzutów prostopadłościanów A i B na oś

$$(\vec{n}\cdot\vec{r}_A)_{max}=\frac{a_A}{2}\left|\vec{n}\cdot\vec{i}'\right|+\frac{b_A}{2}\left|\vec{n}\cdot\vec{j}'\right|+\frac{c_A}{2}\left|\vec{n}\cdot\vec{k}'\right|$$

$$(\vec{n}\cdot\vec{r}_B)_{max}=\frac{a_B}{2}\left|\vec{n}\cdot\vec{i}''\right|+\frac{b_B}{2}\left|\vec{n}\cdot\vec{j}''\right|+\frac{c_B}{2}\left|\vec{n}\cdot\vec{k}''\right|=$$

$$=\frac{a_B}{2}\left|\vec{n}\cdot\boldsymbol{R}'_B\vec{i}'\right|+\frac{b_B}{2}\left|\vec{n}\cdot\boldsymbol{R}'_B\vec{j}'\right|+\frac{c_B}{2}\left|\vec{n}\cdot\boldsymbol{R}'_B\vec{k}'\right|$$

Detekcja kolizji

- Optymalizacje
  - Jeżeli te wielkości wyrazimy we współrzędnych układu lokalnego związanego z prostopadł. A:

$$(\vec{n} \cdot \vec{r}_A)_{max} = \frac{a_A}{2} |\vec{n}' \cdot \hat{x}| + \frac{b_A}{2} |\vec{n}' \cdot \hat{y}| + \frac{c_A}{2} |\vec{n}' \cdot \hat{z}|$$

$$(\vec{n} \cdot \vec{r}_B)_{max} = \frac{a_B}{2} |\vec{n}' \cdot \boldsymbol{R}'_B \hat{x}| + \frac{b_B}{2} |\vec{n}' \cdot \boldsymbol{R}'_B \hat{y}| + \frac{c_B}{2} |\vec{n}' \cdot \boldsymbol{R}'_B \hat{z}|$$

$\vec{n}'$ to wektor wyznaczający oś rzutowania w układzie współrzędnych związanym z bryłą A

Detekcja kolizji

- Optymalizacje
  - Korzyści z tego podejścia zobaczymy, gdy za oś $\vec{n}'$ wstawimy konkretny wektor np. $\vec{n}' = \hat{x} = (1,0,0)$

$$\vec{n} \cdot \left(\vec{R}_{śm,B} - \vec{R}_{śm,A}\right) = \hat{x} \cdot \vec{R}'_{śm,B} = \left(\vec{R}'_{śm,B}\right)_x$$

$$(\vec{n} \cdot \vec{r}_A)_{max} = \frac{a_A}{2}$$

$$(\vec{n} \cdot \vec{r}_B)_{max} = \frac{a_B}{2} |\hat{x} \cdot \boldsymbol{R}'_B \hat{x}| + \frac{b_B}{2} |\hat{x} \cdot \boldsymbol{R}'_B \hat{y}| + \frac{c_B}{2} |\hat{x} \cdot \boldsymbol{R}'_B \hat{z}|$$

Elementy macierzy transformacji (obrotu) układu A do układu B,
czyli elementy macierzy obrotu bryły B w lokalnym układzie odniesienia bryły A.

$$\hat{x} \cdot \boldsymbol{R}'_B \hat{y} = \hat{x} \cdot \hat{y}' = \hat{x} \cdot \left(R_{xy}, R_{yy}, R_{zy}\right) = R_{xy} \qquad (\boldsymbol{R}'_B)_{ij} = R_{ij}$$

Detekcja kolizji

- Optymalizacje
  - Ostatecznie test separacji względem osi Ax przyjmie następującą postać:

$$\frac{a_A}{2} + \frac{a_B}{2}|R_{xx}| + \frac{b_B}{2}|R_{xy}| + \frac{c_B}{2}|R_{xz}| < \left|\left(\vec{R}'_{śm,B}\right)_x\right|$$

  - Dla osi Bx:

$$\frac{a_A}{2}|R_{xx}| + \frac{b_A}{2}|R_{yx}| + \frac{c_A}{2}|R_{zx}| + \frac{a_B}{2} < \left|R_{xx}\left(\vec{R}'_{śm,B}\right)_x + R_{yx}\left(\vec{R}'_{śm,B}\right)_y + R_{zx}\left(\vec{R}'_{śm,B}\right)_z\right|$$

Detekcja kolizji

- Optymalizacje
  - Dla osi Ax x Bx równej

$$\vec{n}' = \hat{x} \times \boldsymbol{R}'_B \hat{x} = \hat{x} \times \left(R_{xx}, R_{yx}, R_{zx}\right) = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ 1 & 0 & 0 \\ R_{xx} & R_{yx} & R_{zx} \end{vmatrix} = \hat{z}R_{yx} - \hat{y}R_{zx} = \left(0, -R_{zx}, R_{yx}\right)$$

otrzymamy test postaci:

$$\frac{b_A}{2}\left|R_{zx}\right| + \frac{c_A}{2}\left|R_{yx}\right| + \frac{b_B}{2}\left|R_{xz}\right| + \frac{c_B}{2}\left|R_{xy}\right| < \left|R_{yx}\left(\vec{R}'_{sm,B}\right)_z - R_{zx}\left(\vec{R}'_{sm,B}\right)_y\right|$$

Wyprowadzenie (nie do końca banalne) w domu!

  - implementacja (zazwyczaj poprzedzone testem BS)
  - kolejność testów: prawdopodobieństwo spełnienia

Detekcja kolizji

- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

  W przypadku wielościanów można użyć metody V-clip (Briana Mitrich). Rozpoczyna się od podziału przestrzeni wokół wielościanu na obszary. Każdy z obszarów w tym podziale składa się z punktów, które są bliżej jednego wierzchołka, krawędzi lub ściany niż innych. Dwuwymiarowy analog tego problemu nazywany jest teselacją Woronoja i prowadzi do słynnego diagramu Woronoja. W detekcji kolizji fakt, że do jednego obszaru należą punkty bliższe jednemu elementowi wielościanu niż innym, ma istotne znaczenie i jest podstawą wydajności tego algorytmu, pozwalając na zastąpienie operacji na elementach wielościanu operacjami na związanych z nimi obszarach.

Detekcja kolizji

- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

  Najbardziej poprawną odpowiedź daje iteracyjna metoda GJK (Gilberta, Johnsona i Keerthi). Dla brył wypukłych. dla których umiemy zdefiniować tzw. funkcję odwzorowania wspierającego (ang. *support mapping function*), zwracającą punkt siatki modelu, który jest najbardziej wysunięty we wskazanym w jej argumencie kierunku. Algorytm rozpoczyna się, przynajmniej formalnie, od wyznaczenia tzw. różnicy Minkowskiego siatek dwóch modeli, których kolizja jest brana pod uwagę. Różnica Minkowskiego to zbiór wszystkich wektorowych różnic położeń punktów z obu siatek. Modele nakładają się na siebie, jeżeli w ich różnicy Minkowskiego znajduje się punkt bliski początkowi układu współrzędnych.

Detekcja kolizji

- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

  Po wyznaczeniu różnicy Minkowskiego tworzymy bryłę opisaną na maksymalnie czterech dowolnie wybranych jej punktach. Następnie korzystając ze wspomnianej przed chwilą funkcji, szukamy punktu *P* należącego do nowego obszaru (niekoniecznie jego wierzchołka), który jest najbliżej początku układu współrzędnych. Jeżeli ten punkt znajduje się w początku układu współrzędnych - znaleźliśmy punkt styku. W przeciwnym razie redukujemy obszar do odcinka lub trójkąta, na którym znajduje się znaleziony przed chwilą punkt *P*.
  Jak widać w obliczeniach nie ma potrzeby wyznaczania całej różnicy Minkowskiego. Potrzebujemy tylko czterech jej punktów i tylko te punkty obliczamy korzystając z funkcji odwzorowania wspierającego (wydajność!).

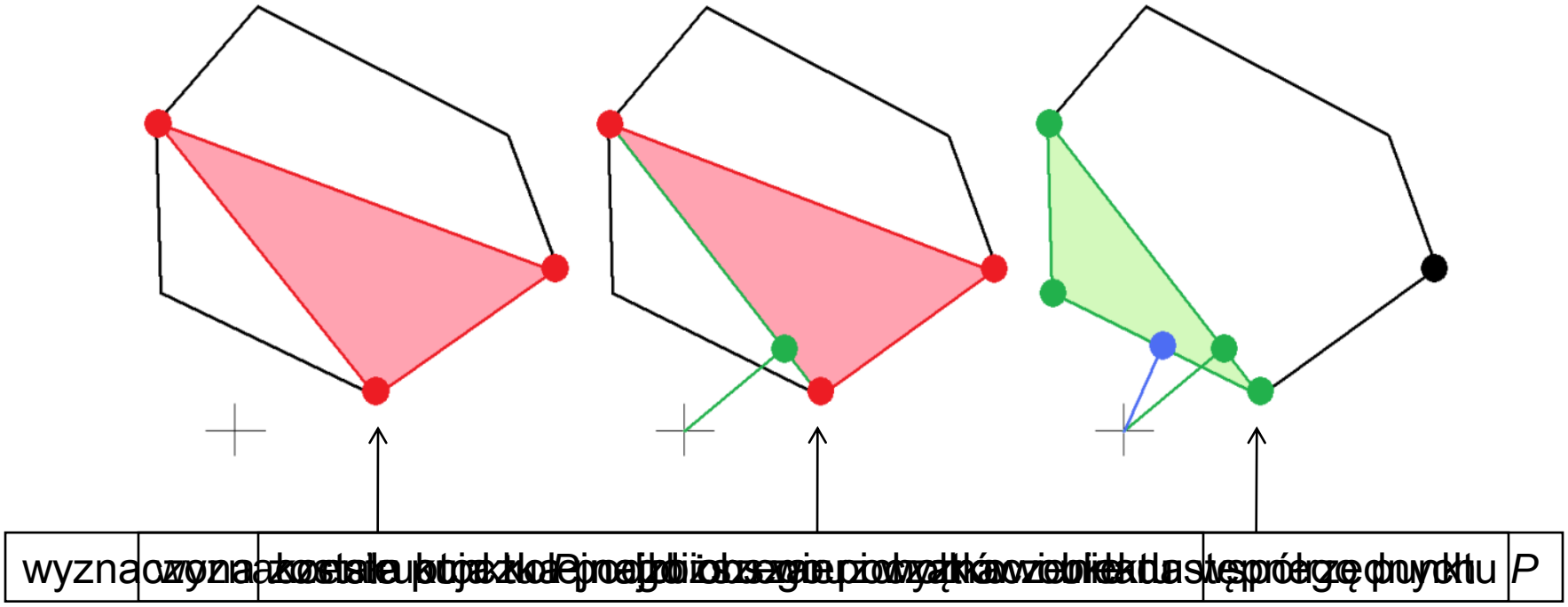
Detekcja kolizji

- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

  W następnym kroku szukamy wierzchołka modelu, który jest najbardziej odległy od punktu *P*. Jeżeli jego odległość od początku układu współrzędnych jest taka sama, jak punktu *P*, to znaczy, że nowy wierzchołek odpowiada różnicy dwóch najbliższych siebie punktów z obu siatek. To jest punkt styku.
  Jeżeli tak nie jest, włączamy nowy wierzchołek do podobszaru i powstaje nowy obszar (trójkąt lub czworościan), w którym szukamy punktu najbliższego początkowi układu współrzędnych.
  To rozpoczyna drugą iterację algorytmu.

Detekcja kolizji

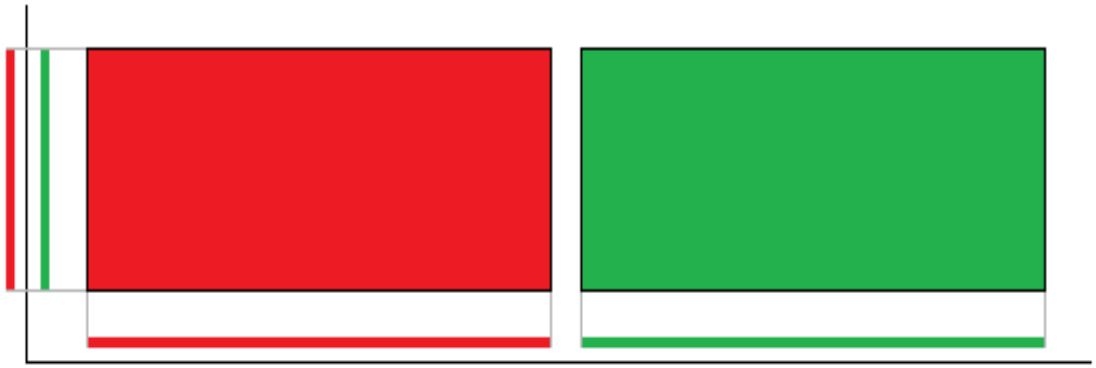
- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?



Detekcja kolizji

- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

„Wydobywanie” punktu styku i normalnej z twierdzenia SAT:
Normalną zderzenia można ustalić śledząc zmiany, jakie następują wzdłuż każdej z piętnastu osi separacji. Należy ustalić, w którym kierunku nakrywanie pojawiło się jako ostatnie. Ta oś będzie wskazywała normalną.



Detekcja kolizji

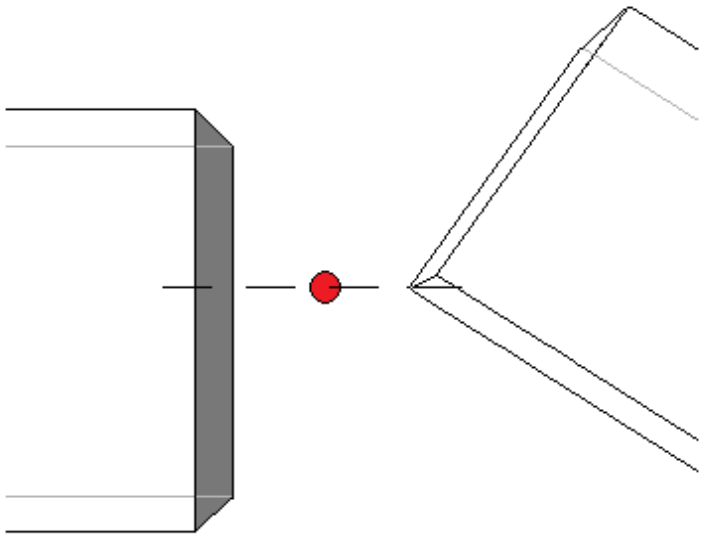
- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

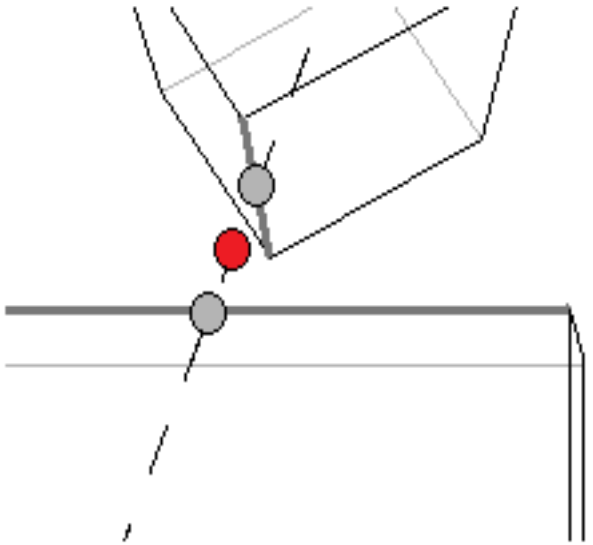
  W praktyce sytuacja jest bardziej skomplikowana ze względu na kwantyzację czasu w symulacjach. Zazwyczaj tuż przed zderzeniem to nie jedna, ale dwie osie pokazują separację. Typową sytuacją jest, że jedna z osi odpowiada zderzeniu typu wierzchołek-ściana, a druga – krawędź-krawędź. W tej sytuacji trzeba rozstrzygnąć, na której osi przy ciągłym upływie czasu separacja zniknęłaby później. Możemy to zrobić, obliczając odległości rzutów prostopadłościanów na poszczególne osie tuż przed i tuż po zderzeniu. Czas, jaki upłynął od początku kroku do chwili zetknięcia można obliczyć zakładając, że w przybliżeniu zmiana ta zachodzi z jednakową prędkością. Jeżeli na jednej z osi doszło do zderzenia, to odległość rzutów tuż po zderzeniu jest ujemna. Jej wartość bezwzględna jest wówczas nazywana głębokością penetracji.

Detekcja kolizji

- Jak znaleźć punkt styku i normalną zderzenia (linię akcji), czyli układ odniesienia zderzenia?

Punkt styku:





Tu pojawia się problem odległości dwóch prostych