## II : KINEMATYKA

W tej części zajmiemy się fundamentalnymi aspektami kinematyki. W szczególności wyjaśnimy pojęcia przemieszczenia liniowego i kątowego, prędkości i przyspieszenia. Po omówieniu ruchu cząsteczek, wyjaśnimy określone aspekty ruchu ciała sztywnego.

### Wprowadzenie

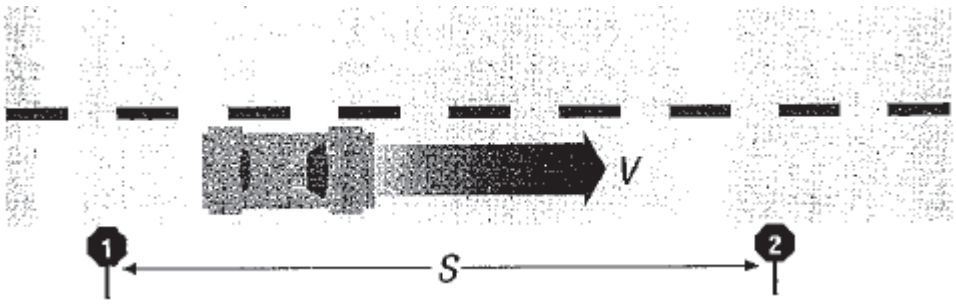
Kinematyka jest badaniem ruchu ciał bez względu na siły działające na ciało. Dlatego też w kinematyce uwaga będzie skupiona na pozycji, prędkości i przyspieszeniu ciała; jak te właściwości są powiązane i jak zmieniają się w czasie. Tu przyjrzymy się dwóm typom ciał : cząstkom i ciałom sztywnym. Ciało sztywne jest systemem cząstek , które pozostają w ustalonych odległościach od siebie, bez względnego przesunięcia lub rotacji między nimi. Innymi słowy, ciało sztywne nie zmienia swojego kształtu podczas ruchu lub jakiekolwiek zmiany w kształcie ciała są tak małe lub nieistotne , że można je bezpiecznie pominąć. Biorąc pod uwagę ciało sztywne, jego wymiary i orientacja są ważne, i musisz wziąć pod uwagę zarówno liniowy ruch ciała jak i ruch kątowy. Z drugiej strony, cząstka jest ciałem które ma masę, ale jej wymiary są nieistotne lub nieważne w badanym problemie. Na przykład, kiedy rozpatrujemy ścieżkę pocisku lub rakiety na dużej odległości, możesz bezpiecznie zignorować wymiary ciała analizując jego trajektorię. Kiedy rozważasz cząstkę, jej ruch liniowy jest ważny, ale ruch kątowy samej cząstki nie jest. Wygląda na to ,że patrząc na problem, oddalasz się od niego, patrząc na duży obraz, że tak powiem, w przeciwieństwie do powiększania ciała, jak robisz to , patrząc na rotację ciała sztywnych. Niezależnie od tego czy patrzysz na problemy z udziałem cząstek czy ciała sztywnego, istnieją pewne ważne cechy kinematyczne wspólne dla obu. Chodzi oczywiści o pozycję obiektu, prędkość i przyspieszenie.

### Prędkość I Przyspieszenie

Ogólnie rzecz biorąc, orędkość jest wielkością wektora , która ma wielkość i kierunek. Modułem prędkości jest szybkość. Szybkość jest terminem znanym : to szybkość jaką pokazuje Twój prędkościomierz, kiedy jedziesz samochodem po autostradzie. Formalnie, szybkość jest współczynnikiem podróży , lub współczynnikiem odległości przebytej do czasy pokonania tej odległości. W terminach matematycznych możemy zapisać:

$v = \Delta s / \Delta t$

gdzie v jest szybkością, moduł prędkości v, a Δs jest odległością przebytą przez odcinek czasu Δt. Zauważ ,że ta relacja ujawnia ,że jednostki szybkości są złożone z podstawowych wymiarów długości podzielonych przez czas, L/T. Niektóre typowe jednostki prędkości to metry na sekundę, m/s. Oto prosty przykład : Samochód jedzie prostą drogą, mija znacznik 1 w czasie $t_1$ i znacznik 2 w czasie $t_2$, gdzie $t_1$ jest równe 0 a $t_2$ jest równe 1,136 s. Odległość między tymi dwoma znacznikami , s, to 30 metrów. Obliczymy szybkość samochodu



Wiemy ,że s równa się 30 metrów, Δs równa się 30 metrów, a Δt równa się $t_2 – t_1$, lub 1,136. Sybkość samochodu na tej odegłości to

v = Δs /Δt = 30 / 1,136 = 26,4 m/s

co oznacza w przybliżeniu 95 km/h. Jest to prosty, jednowymiarowy przykład, ale przedstawia ważny punkt, który polega na ty, ,że obliczona właśnie prędkość jest średnią szybkością samochodu na tej odległości. W tej chwili nie wiesz nic o przyspieszeniu samochodu ani o tym, czy lub nie porusza się ze stałą 95 km/h. Równie dobrze mogłoby się zdarzyć, że samochód przyspieszył (lub zwalniał) na odległości 30 metrów.

Aby dokładniej przeanalizować ruch samochodu w tym przykładzie, musisz zrozumieć pojęcie prędkości chwilowej. Prędkość chwilowa jest specyficzną prędkością w danej chwili czasu, a nie w dużym przedziale czasu, jak w przykładzie z samochodem. Oznacza to ,że musisz spojrzeć na bardzo małe Δt. W terminologii matematycznej , musisz rozważyć granicę przy Δt dążącym do zera, tzn. gdy Δt staje się nieskończenie małe. Jest to zapisane w następujący sposób:

$v = \lim\Delta_{t->0}(\Delta s / \Delta t)$

W terminologii różniczkowej, prędkość jest pochodną przemieszczenia (zmiana położenia) względem czasu

v = ds / dt

Możesz zmienić ten stosunek i całkować w przedziale od $s_1$ do $s_2$ i $t_1$ do $t_2$ jak pokazano tu:

$$v\,dt = ds$$

$$\int_{s_1}^{s_2} ds = \int_{t_1}^{t_2} v\,dt$$

$$s_2 - s_1 = \Delta s = \int_{t_1}^{t_2} v\,dt$$

Ta relacja pokazuje ,że przemieszczenie jest całką prędkości w czasie. Daje to nam sposób poruszania się między przemieszczeniem a prędkością. W kinematyce rozróżnia się między przemieszczeniem a przebytą odległością. W jednym wymiarze, przesunięcie jest takie samo co pokonana odległość; jednak kiedy rozważamy wektory w przestrzeni, przemieszczenie jest po prostu wektorem od pozycji początkowej do końcowej, bez względu na drogę którą przebył; przemieszczenie jest różnicą między współrzędnymi położenia początkowego a współrzędnym położenia końcowego. W związku z tym trzeba być ostrożnym przy obliczaniu średniej prędkości z danym przemieszczeniem, jeśli droga od pozycji początkowej do końcowej nie jest linią prostą. Kiedy Δt jest bardzo małe (dąży do zera), przemieszczenie i odległość przebyta są takie same. Kolejną ważną cechą kinematyki jest przyspieszenie, które też powinno ci być znane. Odnosząc się do doświadczenia z jazdą, wiesz ,że przyspieszenie jest współczynnikiem przy którym możesz zwiększyć prędkość. Twój przyjaciel ,który szczyci się ,że jego nowy XYZ 2001 może przyspieszać od 0 do 100 w 4,2 sekundy. W szczególności odnosi się on do średniego przyspieszenia. Formalnie , średnie przyspieszenie jest współczynnikiem zmiany prędkości, lub Δv przez Δt

a = Δv / Δt

Przyjmując granicę jako Δt dążące do zer , dostajemy przyspieszenie chwilowe:

$$a = \lim_{\Delta t \to 0} \Delta v/\Delta t$$
$$a = dv/dt$$

Zatem przyspieszenie jest stopniem szybkości zmiany w prędkości lub pochodnej prędkości względem czasu. Zmieniając i integrując pola

$$dv = a\,dt$$
$$\int_{v_1}^{v_2} dv = \int_{t_1}^{t_2} a\,dt$$
$$v_2 - v_1 = \Delta v = \int_{t_1}^{t_2} a\,dt$$

Ta zależność zapewnia środki do pracy między prędkością a przyspieszeniem. Zatem związek między przemieszczeniem , prędkością a przyspieszeniem

$a = dv/dt = d^2s/dt$

i

$v\,dv = a\,ds$.

Jest to kinematyczne różniczkowe równanie ruchu.

**Stałe Przyspieszenie**

Jedną z najprostszych klas problemów w kinematyce obejmuje stałe przyspieszenie. Dobrym przykładem tego rodzaju problemu jest przyspieszenie ze względu na grawitację g, na obiekty poruszające się stosunkowo blisko powierzchni Ziemi, gdzie przyspieszenie grawitacyjne jest stałe i wynosi 9,8 $m/s^2$. Posiadanie stałego przyspieszenia sprawia ,że całkowanie w czasie jest stosunkowo łatwe, ponieważ można wyciągnąć stałą przyspieszenia z funkcji podcałkowej, pozostawiając tylko dt. Całkowanie zależności między prędkością a przyspieszeniem opisane wcześniej ,gdy przyspieszenie jest stałe, daje następujące równanie dla prędkości chwilowej:

$$\int_{v_1}^{v_2} dv = \int_{t_1}^{t_2} a\,dt$$
$$\int_{v_1}^{v_2} dv = a\int_{t_1}^{t_2} dt$$
$$v_2 - v_1 = a\int_{t_1}^{t_2} dt$$
$$v_2 - v_1 = a(t_2 - t_1)$$
$$v_2 = at_2 - at_1 + v_1$$

Kiedy $t_1$ jest równe zero, możesz przepisać to równanie w poniższej formie

$$v_2 = at_2 + v_1$$
$$v_2 = v_1 + at_2$$

To proste równanie pozwala obliczyć chwilową prędkość w danym momencie, w danym momencie, znając czas który minął, prędkość początkową i stałe przyspieszenie. Możesz również wyprowadzić równanie prędkości  jako funkcję przesunięcia zamiast czasu, biorąc pod uwagę kinematyczne równanie różniczkowe ruchu

v dv = a ds.

Całkowanie obu stron tego równania daje alternatywna funkcję dla prędkości chwilowej

$$\int_{v_1}^{v_2} v\,dv = a\int_{s_1}^{s_2} ds$$
$$(v_2^2 - v_1^2)/2 = a(s_2 - s_1)$$
$$v_2^2 = 2a(s_2 - s_1) + v_1^2$$

Możesz również wyprowadzić podobny wzór dla przemieszczenia jaki funkcję prędkości, przyspieszenia i czasy przez całkowanie równania różniczkowego

v dt = ds.

ze sformułowanym wcześnie wzorem dla chwilowej prędkości

$v_2 = v_1 + at$

zastępując dla v. W ten sposób uzyskuje się wzór

$$s_2 = s_1 + v_1 t + (at^2)/2$$

Podsumowując , powyższe trzy równania kinematyczne są następujące:

$$v_2 = v_1 + at_2$$
$$v_2^2 = 2a(s_2 - s_1) + v_1^2$$
$$s_2 = s_1 + v_1 t + (at^2)/2$$

Pamiętaj, że te równania są ważne tylko wtedy gdy przyspieszenie jest stałe. Zauważ ,że przyspieszenie może być zerowe lub nawet ujemne w przypadku gdy ciało zwalnia. Możesz zmienić te równania przez algebraiczne rozwiązania dla różnych zmiennych, i możesz również wyprowadzać inne przydatne równania, używając tego samego podejścia, które właśnie poznałeś. Poniżej przedstawione jest kilka przydatnych równań kinetycznych, dotyczących problemów ze stałym przyspieszeniem

| Znajdź | Dane | Użyj Tego |
|---|---|---|
| $a$ | $\Delta t, v_1, v_2$ | $a = (v_2 - v_1)/\Delta t$ |
| $a$ | $\Delta t, v_1, \Delta s$ | $a = (2\Delta s - 2v_1\Delta t)/(\Delta t)^2$ |
| $a$ | $v_1, v_2, \Delta s$ | $a = (v_2^2 - v_1^2)/(2\Delta s)$ |
| $\Delta s$ | $a, v_1, v_2$ | $\Delta s = (v_2^2 - v_1^2)/(2a)$ |
| $\Delta s$ | $\Delta t, v_1, v_2$ | $\Delta s = (\Delta t / 2)(v_1 + v_2)$ |
| $\Delta t$ | $a, v_1, v_2$ | $\Delta t = (v_2 - v_1)/a$ |
| $\Delta t$ | $a, v_1, \Delta s$ | $\Delta t = \left(\sqrt{v_1^2 + 2a\Delta s} - v_1\right)/a$ |
| $\Delta t$ | $v_1, v_2, \Delta s$ | $\Delta t = (2\Delta s)/(v_1 + v_2)$ |
| $v_1$ | $\Delta t, a, v_2$ | $v_1 = v_2 - a\Delta t$ |
| $v_1$ | $\Delta t, a, \Delta s$ | $v_1 = \Delta s / \Delta t - (a\Delta t)/2$ |
| $v_1$ | $a, v_2, \Delta s$ | $v_1 = \sqrt{v_2^2 - 2a\Delta s}$ |

W przypadku, w którym przyspieszenie nie jest stałe ale jest funkcją czasu, prędkości lub pozycji, możesz zastąpić funkcję przyspieszenia na równanie różniczkowe pokazane wcześniej dla wyprowadzenia nowego równania dla prędkości chwilowej i przemieszczenia. Kolejna sekcja rozpatruje taki problem.

**Przyspieszenie Niestałe**

Powszechną sytuacją która pojawia się w problemach świata realnego, jest sprawa kiedy siła oporu działa na ciało w ruchu. Zazwyczaj siła oporu jest proporcjonalna do prędkości do kwadratu. Przywołując równanie z drugiego prawa ruchu Newtona , F = ma, można wnioskować , że przyspieszenie wywołane prze te siły oporu jest również proporcjonalne do prędkości do kwadratu. Później zobaczysz kilka technik, które pozwolą ci obliczyć tego rodzaju siły oporu, ale na chwilę obecną niech funkcjonalna forma przyspieszenia wywołane przez opór to

$a = - kv^2$ ,

gdzie k jest stałą a ujemny znak wskazuje ,że to przyspieszenie działa w kierunku przeciwnym do prędkości ciała. Teraz zastępując ten wzór dla przyspieszenia do powyższego równania a potem przestawiając

$$a = dv/dt$$
$$-kv^2 = dv/dt$$
$$-k\,dt = dv/v^2$$

Jeśli scałkujemy prawą stronę tego równania od $v_1$ do $v_2$, a lewą stronę od 0 do t, a potem rozwiązując dla $v_2$, otrzymamy wzór dla prędkości chwilowej jako funkcji prędkości początkowej i czas jak pokazano tutaj:

$$-k\int_0^t dt = \int_{v_1}^{v_2} (1/v^2)\,dv$$
$$-k\,t = 1/v_1 - 1/v_2$$
$$v_2 = v_1/(1 + v_1 kt)$$

Jeśli zastąpimy to równanie dla v w relacji v = ds./dt i ponownie je scałkujesz , otrzymasz nowe równanie dla przemieszczenia jako funkcji prędkości początkowej i czasu. Procedura jest pokazana poniżej

$$v\,dt = ds, \quad \text{where} \quad v = v_1/(1 + v_1 kt)$$
$$\int_0^t v\,dt = \int_{s_1}^{s_2} ds$$
$$\int_0^t [v_1/(1 + v_1 kt)]\,dt = \int_{s_1}^{s_2} ds$$
$$\ln(1 + v_1 kt)/k = s_2 - s_1$$

Jeśli $s_1$ jest równe zero , wtedy

$s = \ln(1 + v_1 kt)/k$

Zwróć uwagę ,że w tym równania ln jest operatorem logarytmu naturalnego.

Ten przykład demonstruje względną złożoność problemów niestałego przyspieszenia w porównaniu do problemów stałego przyspieszenia. Jest to dość prosty przykład, w którym można wyprowadzić równania w postaci zamkniętej dla prędkości i przemieszczenia. W praktyce jednak może istnieć kilka różnych rodzajów sił działających na dane ciało w ruchu, co może sprawić ,że ekspresja indukowanego przyspieszenia jest dość skomplikowana. Ta złożoność spowodowałaby, że rozwiązanie w kształcie zamkniętym, jak powyżej, byłoby niemożliwe, chyba ,że nałożysz pewne upraszczające ograniczenie. na problem, zmuszając Cię do polegania na innych rozwiązaniach takich jak całkowanie numeryczne.

**Kinematyka Cząstek W 2D**

Rozważając ruch w jednym wymiarze, to znaczy, kiedy ruch ogranicza się do linii prostej, można łatwo zastosować wzory otrzymane wcześniej, w celu określenie prędkości chwilowej, przyspieszenia i przemieszczenia. Jednak, w dwóch wymiarach, z ruchem możliwym w dowolnym kierunku na danej płaszczyźnie, musisz uwzględnić kinematyczną właściwość wiązania prędkości, przyspieszenia i przemieszczenia jak wektory. Używając prostokątnych współrzędnych w standardowym systemie współrzędnych kartezjańskich, musisz uwzględnić komponenty x i y przemieszczenie, prędkości i przyspieszenia. Zasadniczo można potraktować komponenty x i y oddzielnie a potem nałożyć te komponenty dla zdefiniowania odpowiednich wielkości wektorowych. Aby pomóc śledzić te komponenty x i y, niech i ij będą wektorami jednostkowymi odpowiednio w kierunku x i y. Teraz możemy zapisać kinematyczną właściwość wektorów pod względem ich komponentów w następujący sposób:

$v = vx_i + vy_j$

$a = ax_i + av_j$

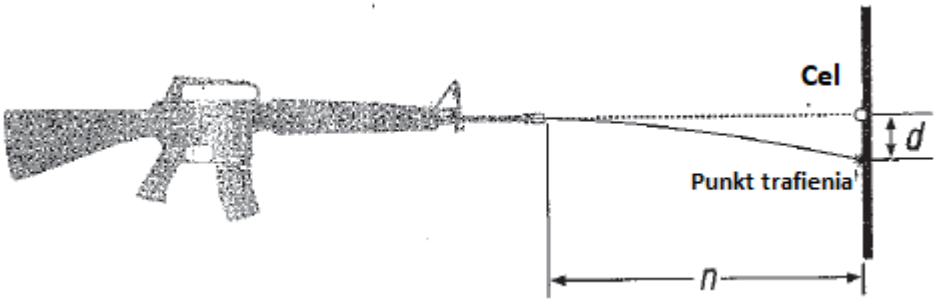
Jeśli x jest przemieszczeniem w kierunku x a y jest przemieszczeniem w kierunku y, wtedy wektor przemieszczenia to

s = xi + yj

Stąd wynika

$$\mathbf{v} = d\mathbf{s}/dt = dx/dt\,\mathbf{i} + dy/dt\,\mathbf{j}$$
$$\mathbf{a} = d\mathbf{v}/dt = d^2\mathbf{s}/dt = d^2x/dt\,\mathbf{i} + d^2y/dt\,\mathbf{j}$$

Rozważmy prosty przykład ,w którym piszesz grę łowiecką i musisz znaleźć pionowy spadek wystrzelone kuli od miejsca docelowego do miejsca w którym trafia cel. Zakładamy ,że nie ma wiatru i nie ma oporu ponieważ kula leci w powietrzu. Takie założenia redukują problem do stałego przyspieszenie, które w tym przypadku wynika z grawitacji. To właśnie przyspieszenie grawitacyjne jest odpowiedzialne za spadek kuli podczas przemieszczania się z karabinu do celu




Niech początek układy współrzędnych 2D znajduje się na końcu karabinu, z osią x skierowaną w stronę celu i osią y wskazującą w górę. Dodatnie przesunięcie wzdłuż osi x są skierowane w stronę celu a dodatnie przesunięcie wzdłuż osi y jest skierowane w górę. Oznacza to ,że przyspieszenie grawitacyjne będzie działało w ujemnym kierunku y. Potraktowanie komponentów x i y oddzielnie pozwala ci na rozbicie problemu na małe , łatwe do obsługi elementy. Patrząc najpierw na komponent x, wiesz ,że pocisk opuści karabin z początkową prędkością wylotową $v_m$ w kierunku x, a ponieważ pomijamy opór, ta prędkość będzie stała. Zatem,

$$a_x = 0$$
$$v_x = v_m$$
$$x = v_x t = v_m t$$

Teraz spójrzmy na komponent y, wiemy ,że początkowa szybkość w kierunku y, jeśli pocisk opuści karabin, jest zerowa, ale przyspieszenie y to –g (ze względu na grawitację). Zatem

$$a_y = -g = dv_y/dt$$
$$v_y = a_y t = -gt$$
$$y = (1/2)a_y t^2 = -(1/2)gt^2$$

Teraz wektory przemieszczenia, prędkości i przyspieszenie mogą być zapisane następująco:

$$\mathbf{s} = (v_m t)\mathbf{i} - (1/2gt^2)\mathbf{j}$$
$$\mathbf{v} = (v_m)\mathbf{i} - (gt)\mathbf{j}$$
$$\mathbf{a} = -(g)\mathbf{j}$$

Te równania podają przesunięcie chwilowe, prędkość i przyspieszenie dla dowolnej chwili czasu pomiędzy momentem w którym pocisk opuści karabin, a czasem, kiedy trafia w cel. Moduły tych wektorów dają całkowite przemieszczenie, prędkość i przemieszczenie w danym czasie. Na przykład:

$$s = \sqrt{(v_m t)^2 + (1/2gt^2)^2}$$
$$v = \sqrt{(v_m)^2 + (gt)^2}$$
$$a = \sqrt{g^2} = g$$

Aby obliczyć pionowy spadek pocisku w momencie gdy pocisk trafia w cel, musisz najpierw obliczyć czas potrzebny do osiągnięcia celu, a następnie możesz użyć tego czasu aby wyliczyć przemieszczenie komponentu y, które jest pionowym spadkiem. Oto wzory

$$t_{hit} = x_{hit}/v_m = n/v_m$$
$$d = y_{hit} = -(1/2)g(t_{hit})^2$$

gdzie n jest odległością od karabinu do celu a d jest pionowym spadkiem kuli w miejscu docelowym. Jeśli odległość od celu, n, równa jest 500 metrów (m) a prędkość wylotowa $v_m$ , równa jest 800 m/s, wtedy $t_{hit}$ i d , równają się

$t_{hit}$ = 0,635 s

d = 1,9

Wyniki te mówią nam ,że aby trafić do zamierzonego celu w danym zakresie, musisz mierzyć około 2 metry nad nim

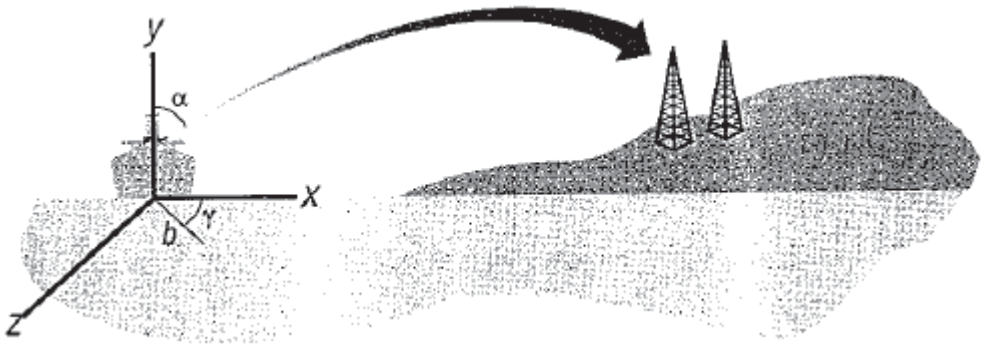
**Kinematyka Cząstek 3D**

Rozszerzenie właściwości kinematycznych wektorów do trzech wymiarów nie jest bardzo trudne. Chodzi po prostu o dodanie jeszcze jednego komponentu do reprezentacji wektorowej przedstawionej wcześniej, dotyczącej kinematyki 2D. Wprowadzając k jako wektor jednostkowy w kierunku z, można teraz zapisać

$$\mathbf{s} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$
$$\mathbf{v} = d\mathbf{s}/dt = dx/dt\,\mathbf{i} + dy/dt\,\mathbf{j} + dz/dt\,\mathbf{k}$$
$$\mathbf{a} = d^2\mathbf{s}/dt = d^2x/dt\,\mathbf{i} + d^2y/dt\,\mathbf{j} + d^2z/dt\,\mathbf{k}$$

Zamiast potraktować dwa składniki osobno, a następnie nakładać je na siebie, teraz traktujemy trzy komponenty oddzielnie i nakładamy je na siebie. Najlepiej zilustruje to przykład. Załóżmy ,że zamiast gry myśliwskiej, teraz napiszmy grę, która polega na wystrzeliwaniu z armaty, powiedzmy z okrętu wojennego na cel oddalony o pewną odległość, na przykład inny statek lub cel na lądzie, np. budynek. Aby zwiększyć złożoność takiej aktywności dla Twojego użytkownika, musisz dać mu kontrolę nad kilkoma czynnikami, które wpływają na trajektorie pocisku, a mianowicie kąt strzału armaty, zarówno

poziomy jak i pionowy, jak i prędkość wylotową pocisku, która jest kontrolowana przez ilość prochu pod kulą gdy jest ładowana do armaty




Skonfigurujemy równania kinematyczne dla tego problemu, traktując każdy komponent wektorowy oddzielnie na początku, a następnie łącząc te komponenty.

**Komponent x**

Komponenty x są podobne do tych pokazanych w przykładzie z karabinem, w którym nie moa siły oporu działającej na pocisk; zatem komponent x przyspieszenia to zero, co oznacza ,że komponent x prędkości jest stały i jest równy komponentowi x prędkości wylotowej, gdy kula opuszcza działo. Zauważ ,że skoro lufa działa nie może być pozioma , będziesz musiał obliczyć komponent x prędkości wylotowej która jest funkcją kierunku w którym skierowane jest działo. Wektor prędkości wylotowej to

$$\mathbf{v}_m = v_{mx}\mathbf{i} + v_{my}\mathbf{j} + v_{mz}\mathbf{k}$$

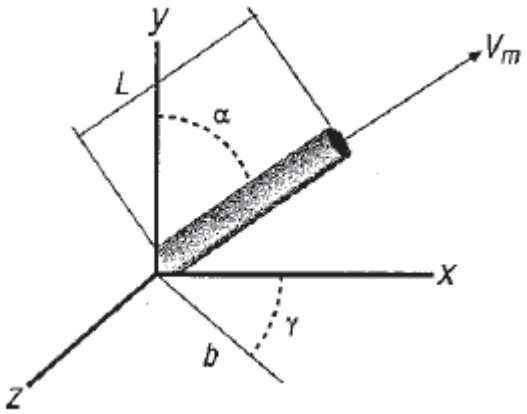
a otrzymasz tylko kierunek $v_m$ określony przez kierunku w którym użytkownik wskazuje działo a jego wielkość jest określona prze ilość prochu jaką użytkownik pakuje do armaty. Aby obliczyć składniki prędkości wylotowej, musisz opracować pewne równania dla tych komponentów pod względem kierunku kątów aramty i moduł prędkości wylotowej. Możesz użyć wektora cosinusów kierunku dla określenie składowej prędkości

$$\cos\theta_x = v_{mx}/v_m$$
$$\cos\theta_y = v_{my}/v_m$$
$$\cos\theta_z = v_{mz}/v_m$$

Ponieważ wektor kierunku początkowej prędkości wylotowej jest taki sam jak kierunek w którym wycelowane jest działo, możemy potraktować działo jako wektor o wielkości L , długości i wskazując w kierunku zdefiniowanym przez kąty podane w tym problemie. Użycie długości armaty, L , i jej komponentów zamiast prędkości wylotowej w równaniach dla danych cosinusów kierunkowych

$$\cos\theta_x = L_x/L$$
$$\cos\theta_y = L_y/L$$
$$\cos\theta_z = L_z/L$$

W danym przykładzie mamy podane kąty α i γ, które definiują orientację działa




Używając tych kątów, wynika ,że rzut b , długości działa L, na płaszczyznę xy to

$$b = L\cos(90° - \alpha)$$

a komponent długości działa L, na każdej osi współrzędnych to =

$$L_x = b\cos\gamma$$
$$L_y = L\cos\alpha$$
$$L_z = b\sin\gamma$$

Teraz , gdy masz informacje wymagane dla obliczenia cosinusów kierunkowych, możesz napisać równania dla początkowych składników prędkości wylotowej

$$v_{mx} = v_m\cos\theta_x$$
$$v_{my} = v_m\cos\theta_y$$
$$v_{mz} = v_m\cos\theta_z$$

Na koniec, możemy zapisać komponent x przemieszczenia , prędkości i przyspieszenia jak następuje:

$$a_x = 0$$
$$v_x = v_{mx} = v_m\cos\theta_x$$
$$x = v_x t = (v_m\cos\theta_x)\,t$$

**Komponent y**

Komponent y są podobne do poprzedniego przykładu z karabinem, z wyjątkiem tego , że początkowa prędkość w kierunku y

$v_{my} = v_m\cos\theta_y$

Zatem

$$a_y = -g$$
$$v_y = v_{my} + at = (v_m\cos\theta_y) - gt$$

Przed zapisaniem równania dla przemieszczenia komponentu y , musisz rozważyć wysokość podstawy armaty, plus wysokość końca lufy działa, aby obliczyć początkowy komponent y przemieszczenia kiedy kula opuszcza działo. Niech $y_b$ będzie wysokością podstawy działa i niech L będzie długością lufy armaty; wtedy początkowy komponent y przemieszczenie , $y_0$ to

$$y_o = y_b + L\cos\alpha$$

Teraz możesz zapisać równanie dla y jako

$$y = y_o + v_{my}t + (1/2)at^2$$
$$y = (y_b + L\cos\alpha) + (v_m\cos\theta_y)t - (1/2)gt^2$$

**Komponent z**

Komponenty z w dużej mierze analogiczne do komponentów x i można je zapisać w następujący sposób:

$$a_z = 0$$
$$v_z = v_{mz} = v_m\cos\theta_z$$
$$z = v_z\,t = (v_m\cos\theta_z)t$$

**Wektory**

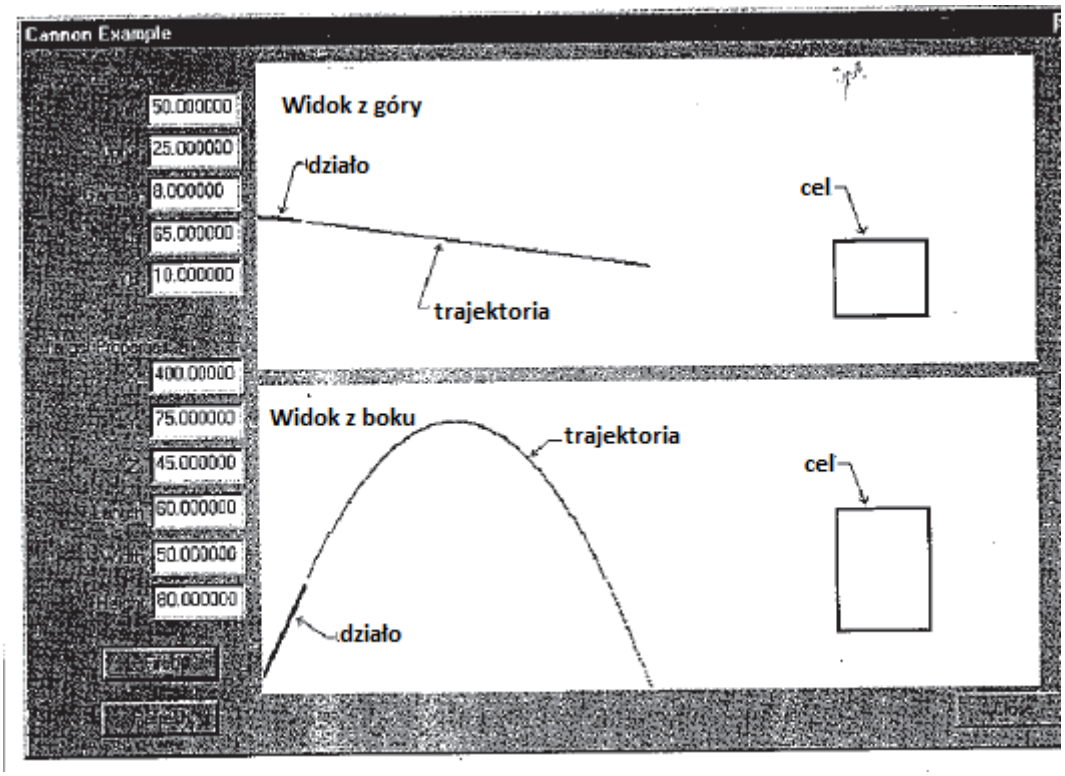
Po opracowaniu wszystkich komponentów można je teraz połączyć, tworząc wektor dla każdej cechy kinematycznej. Wykonanie dla tego przykładu daje wektory przemieszczenia prędkości i przyspieszenia

$$\mathbf{s} = [(v_m\cos\theta_x)t]\mathbf{i} + [(y_b + L\cos\alpha) + (v_m\cos\theta_y)t - (1/2)gt^2]\mathbf{j} + [(v_m\cos\theta_z)t]\mathbf{k}$$
$$\mathbf{v} = [v_m\cos\theta_x]\mathbf{i} + [(v_m\cos\theta_y) - gt]\mathbf{j} + [v_m\cos\theta_z]\mathbf{k}$$
$$\mathbf{a} = -g\mathbf{j}$$

Należy zauważyć ,że wektor przemieszczenia zasadniczo podaje pozycję środka masy kuli w dowolnej chwili w czasie; w ten sposób możesz użyć tego wektora do wykreślenia trajektorii pocisku z działa do celu

**Trafienie W Cel**

Teraz , kiedy mamy równania w pełni opisujące trajektorię kuli, musimy rozważyć lokalizację celu ,aby ustalić kiedy nastąpi bezpośrednie trafienie. Aby pokazać jak się to robi, pokażemy przykładowy program, który implementuje te równania kinematyczne wraz z prostą metodą wykrywania kolizji obwiedni w celu sprawdzania czy lub nie kula trafiła w cel. Zasadniczo ,za każdym razem gdy obliczamy pozycję pocisku po opuszczeniu działa, sprawdzamy czy ta pozycja mieści się w granicach wymiaru obiektu docelowego reprezentowanego przez sześcian. Przykładowy program jest tak skonfigurowany ,że możesz zmienić wszystkie zmienne w symulacji i zobaczyć efekty swoich zmian. Ten program jest prosta aplikacją dialogową, napisaną w standardowym języku C, używając funkcji Windows API. Nazwa pliku wykonywalnego to cannon.exe. Istnieje tylko jeden plik źródłowy cannon.c i jeden plik nagłówkowy cannon.h , dla tego przykładu. Poniższy rysunek pokazuje główny ekran dla przykładowego programu cannon, w którym zmienne sterujące pokazane są po lewej stronie




Górna ilustracja jest widokiem z lotu ptaka patrząc na działo i cel; dolna ilustracja jest widokiem z boku. Możesz zmienić dowolne zmienne pokazane w oknie głównym i nacisnąć przycisk ognia aby zobaczyć wynikową ścieżkę lotu pocisku. Okno komunikatu pojawi się po trafieniu w cel lub gdy pocisk uderzy o ziemię. Program jest skonfigurowany tak ,że możesz wielokrotnie zmieniać zmienne i naciskać ogień, aby zobaczyć wynik, bez usuwania poprzedniego procesu. Pozwoli Ci to ocenić jak trzeba dostosować każdą zmienną , aby trafić w cel. Naciśnij przycisk odświeżania aby przerysować widoki , gdy są zbyt zagracone. Rysunek poniższy pokazuje kilka próbnych strzałów jakie się odbyły nim cel został trafiony cel

Kod tego przykładu jest naprawdę prosty. Oprócz kosztów związanych z ustawieniem okna, kontrolek i ilustracji, cała akcja odbywa się po naciśnięciu przycisku ognia

```
case IDC_FIRE:
        // update the variables with
        // the values shown in the edit controls
        GetDlgItemText(hDlg, IDC_VM, str, 15);
        Vm = atof(str);

        GetDlgItemText(hDlg, IDC_ALPHA, str, 15);
        Alpha = atof(str);

        GetDlgItemText(hDlg, IDC_GAMMA, str, 15);
        Gamma = atof(str);

        GetDlgItemText(hDlg, IDC_L, str, 15);
        L = atof(str);

        GetDlgItemText(hDlg, IDC_YB, str, 15);
        Yb = atof(str);

        GetDlgItemText(hDlg, IDC_X, str, 15);
        X = atof(str);

        GetDlgItemText(hDlg, IDC_Y, str, 15);
        Y = atof(str);

        GetDlgItemText(hDlg, IDC_Z, str, 15);
        Z = atof(str);

        GetDlgItemText(hDlg, IDC_LENGTH, str, 15);
        Length = atof(str);

        GetDlgItemText(hDlg, IDC_WIDTH, str, 15);
        Width = atof(str);

        GetDlgItemText(hDlg, IDC_HEIGHT, str, 15);
        Height = atof(str);

        // initialize the time and status variables
        status = 0;
        time = 0;

        // start stepping through time for the sim.
        // until the target is hit, the shell hits
        // the ground, or the sim. times out.
        while(status == 0)
        {
                // do the next time step
                status = DoSimulation();

                // update the views
                hdc = GetDC(hTopView);
                GetClientRect(hTopView, &r);
                DrawTopView(hdc, &r);
                ReleaseDC(hTopView, hdc);
```

```
        hdc = GetDC(hSideView);
        GetClientRect(hSideView, &r);
        DrawSideView(hdc, &r);
        ReleaseDC(hSideView, hdc);
}

// Report results
if (status == 1)
        MessageBox(NULL, "Direct Hit", "Score!", MB_OK);

if (status == 2)
        MessageBox(NULL, "Missed Target", "No Score.", MB_OK);

if (status == 3)
        MessageBox(NULL, "Timed Out", "Error", MB_OK);
break;
```

Pierwsze kilka linii przyjmuje po prostu nowe wartości dla zmiennych pokazanych w oknie głównym. Następnie program wprowadza pętlę while, przechodząc przez przyrosty czasu i ponownie obliczając położenie pocisku za pomocą wzoru dla wektora przesunięcia s, pokazanego wcześniej. Pozycja pocisku w bieżącym czasie jest wyliczany w funkcji DoSimulation. Natychmiast po wywołaniu DoSimulation, program aktualizuje ilustracje w oknie główny pokazujące trajektorię powłoki. DoSimulation zwraca 0 utrzymując pętlę while, jeśli nie było jeszcze kolizji lub jeśli czas nie osiągnął ustawionej wartości limitu czasu. Po zakończeniu pętli while, zwracając wartość niezerową, do DoSimulation, zwracana jest wartość z tego wywołania funkcji, aby zobaczyć czy nastąpiło trafienie między kulą a ziemię i kulą a celem. Aby program nie utknął w tej pętli, DoSimulation zwróci wartość 3, wskazując ,że trwa to zbyt długo. Teraz spójrzmy na to co się dzieje w funkcji DoSimulation (uwzględniono tu zmienne globalne używane w DoSimulation)

```
//-------------------------------------------------------------------------//
// Define a custom type to represent
// the three components of a 3D vector, where
// i represents the x-component, j represents
// the y-component, and k represents the z-
// component
//-------------------------------------------------------------------------//
typedef struct TVectorTag
{
     double i;
     double j;
     double k;
} TVector;

//-------------------------------------------------------------------------//
// Now define the variables required for this simulation
//-------------------------------------------------------------------------//
double          Vm;    // Magnitude of muzzle velocity, m/s
double          Alpha; // Angle from y-axis (upward) to the cannon.
                       // When this angle is zero, the cannon is pointing
                       // straight up; when it is 90 degrees, the cannon
                       // is horizontal
```

```
double          Gamma;  // Angle from x-axis, in the xz-plane to the cannon.
                        // When this angle is zero the cannon is pointing in
                        // the positive x-direction; positive values of this angle
                        // are toward the positive z-axis
double          L;      // This is the length of the cannon, m
double          Yb;     // This is the base elevation of the cannon, m

double          X;      // The x-position of the center of the target, m
double          Y;      // The y-position of the center of the target, m
double          Z;      // The z-position of the center of the target, m
double          Length; // The length of the target measured along the x-axis, m
double          Width;  // The width of the target measured along the z-axis, m
double          Height; // The height of the target measure along the y-axis, m

TVector          s;     // The shell position (displacement) vector

double          time;   // The time from the instant the shell leaves
                        // the cannon, seconds
double          tInc;   // The time increment to use when stepping through
                        // the simulation, seconds

double          g;      // acceleration due to gravity, m/s^2

//---------------------------------------------------------------------------//
// This function steps the simulation ahead in time. This is where the kinematic
// properties are calculated. The function will return 1 when the target is hit
// and 2 when the shell hits the ground (xz-plane) before hitting the target;
// otherwise, the function returns 0.
//---------------------------------------------------------------------------//
int DoSimulation(void)
//---------------------------------------------------------------------------//
{
    double cosX;
    double cosY;
    double cosZ;
    double xe, ze;
    double b, Lx, Ly, Lz;
    double tx1, tx2, ty1, ty2, tz1, tz2;

    // step to the next time in the simulation
    time+=tInc;

    // First calculate the direction cosines for the cannon orientation.
    // In a real game you would not want to put this calculation in this
    // function, since it is a waste of CPU time to calculate these values
    // at each time step as they never change during the sim. I put them
    // here in this case only so that you can see all the calculation steps in a
    // single function.
    b = L * cos((90-Alpha) *3.14/180); // projection of barrel onto xz-plane
    Lx = b * cos(Gamma * 3.14/180);    // x-component of barrel length
    Ly = L * cos(Alpha * 3.14/180);    // y-component of barrel length
    Lz = b * sin(Gamma * 3.14/180);    // z-component of barrel length

    cosX = Lx/L;
    cosY = Ly/L;
    cosZ = Lz/L;

    // These are the x- and z-coordinates of the very end of the cannon barrel
    // we'll use these as the initial x and z displacements
    xe = L * cos((90-Alpha) *3.14/180) * cos(Gamma * 3.14/180);
    ze = L * cos((90-Alpha) *3.14/180) * sin(Gamma * 3.14/180);
```

```
    // Now we can calculate the position vector at this time
    s.i = Vm * cosX * time + xe;
    s.j = (Yb + L * cos(Alpha*3.14/180)) + (Vm * cosY * time) -
          (0.5 * g * time * time);
    s.k = Vm * cosZ * time + ze;

    // Check for collision with target
    // Get extents (bounding coordinates) of the target
    tx1 = X - Length/2;
    tx2 = X + Length/2;
    ty1 = Y - Height/2;
    ty2 = Y + Height/2;
    tz1 = Z - Width/2;
    tz2 = Z + Width/2;

    // Now check to see whether the shell has passed through the target
    // I'm using a rudimentary collision detection scheme here in which
    // I simply check to see whether the shell's coordinates are within the
    // bounding box of the target. This works for demo purposes, but
    // a practical problem is that you might miss a collision if for a given
    // time step the shell's change in position is large enough to allow
    // it to "skip" over the target.
    // A better approach is to look at the previous time step's position data
    // and to check the line from the previous position to the current position
    // to see whether that line intersects the target bounding box.
    if( (s.i >= tx1 && s.i <= tx2) &&
          (s.j >= ty1 && s.j <= ty2) &&
          (s.k >= tz1 && s.k <= tz2) )
          return 1;

    // Check for collision with ground (xz-plane)
    if(s.j <= 0)
          return 2;

    // Cut off the simulation if it's taking too long
    // This is so the program does not get stuck in the while loop
    if(time>3600)
          return 3;

    return 0;
}
```

Ta funkcja zasadniczo składa się z czterech rzeczy:

- inkrementuje zmienną czasową prze określony przyrost czasu
- oblicza początkową prędkość wylotową komponentów w kierunkach x, y i z
- oblicza nową pozycję kuli
- sprawdza  kolizję z celem, używając schematu prostokąta lub ziemi

Oto kod obliczający pozycję kuli

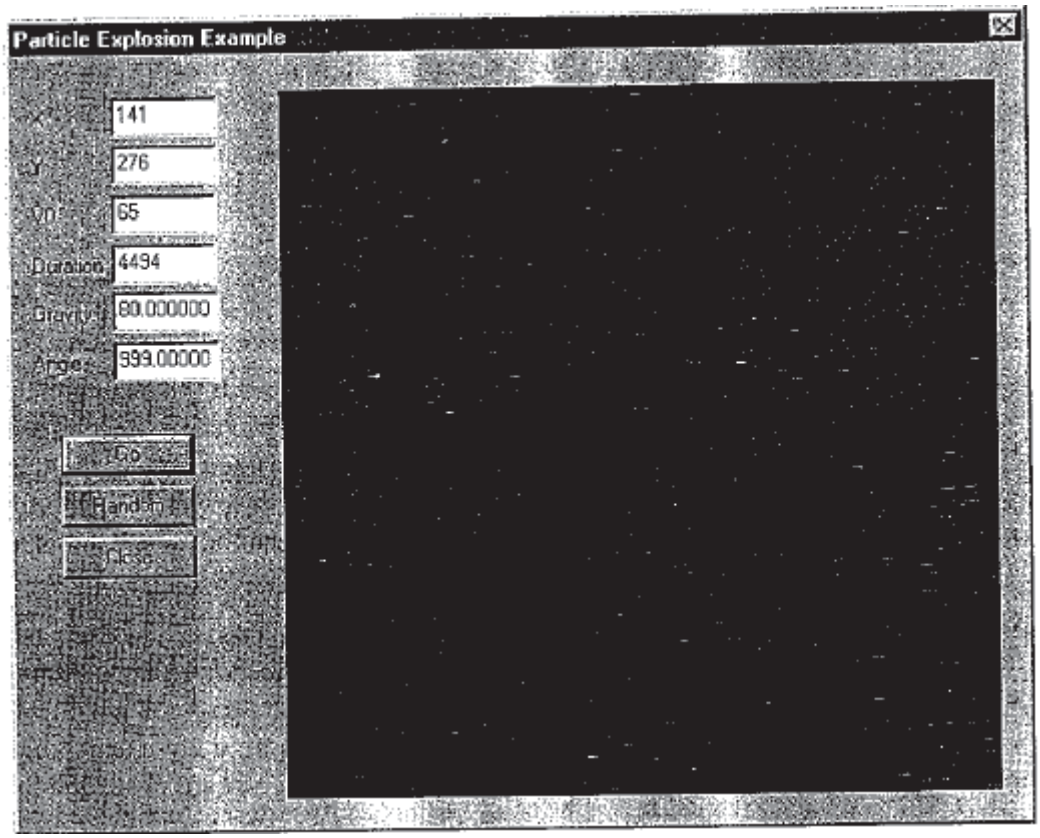
```
// Now we can calculate the position vector at this time
s.i = Vm * cosX * time + xe;
s.j = (Yb + L * cos(Alpha*3.14/180)) + (Vm * cosY * time) -
      (0.5 * g * time * time);
s.k = Vm * cosZ * time + ze;
```

Ten kod oblicza trzy komponenty wektora przemieszczenia  s, używając wzorów , które widziałeś wcześniej. Jeśli chciałbyś obliczyć wektory prędkości i przyspieszenia, aby zobaczyć ich wartości, powinieneś zrobić to w tej sekcji programu. Możesz ustawić kilka nowych zmiennych globalnych aby przedstawić wektory prędkości i przyspieszenia, podobnie jak w przypadku wektora przyspieszenia, zastosuj wzory na prędkość i przyspieszenie jakie poznałeś wcześniej. To wszystko. Oczywiście ,w tym programie , trajektoria pocisku ma kształt paraboliczny, który jest typowym ruchem pocisku.

**Wybuchowa Kinematyka Cząstek**

W tym momencie możesz się zastanawiać, w jaki sposób kinematyka cząstek może Ci pomóc w stworzeniu realistycznej gry, chyba ,że piszesz grę , która wymaga strzelania z pistoletu  lub działa. Jeśli tak, pozwól przedstawić sobie kilka pomysłów, a następnie zobaczysz przykład.  Powiedz ,że piszesz grę symulacyjną footbolu amerykańskiego. Możesz użyć kinematyki cząstek dla modelowania trajektorii piłki  po rzuceniu lub kopnięciu. Możesz również potraktować skrzydłowych jako cząstki podczas obliczania czy będą oni w stanie złapać rzuconą piłkę. W tym scenariuszu będziesz miał dwie cząstki, skrzydłowego i piłkę, poruszający się niezależnie, i będziesz musiał obliczyć kiedy wystąpi zderzenie miedzy tymi dwoma cząstkami, wskazując na chwyt. Możesz znaleźć podobne aplikacje również dla innych gier sportowych. A co z trójwymiarową strzelanką? Jak można użyć kinematyki cząstek w tym gatunku, oprócz pocisków, armat , granatów i tym podobnych? Cóż, możesz użyć kinematyki cząstek aby wymodelować swojego graca gdy on lub ona wyskakują w powietrze, albo biegnąc, albo stając nieruchomo. Na przykład, twój gracz osiąga środek pola tylko po to aby znaleźć brakującą sekcję ,a gracz musi cofnąć się natychmiast kilka kroków aby  złapać lecącą piłkę zanim wyskoczy w powietrze. Ten scenariusz długiego skoku jest idealny do zastosowania kinematyki cząstek. Wszystko co naprawdę musisz zrobić , to zdefiniować początkową prędkość gracza, zarówno szybkość jak i kąt startowy, a potem zastosować wzór wektora dla przemieszczenia do obliczenia czy lub nie wykonuje skok. Możesz również użyć formuły  przesunięcia, aby obliczyć trajektorię  gracza aby odpowiednio przesunąć punkt widzenia gracza , dając iluzję wyskoczenia w powietrze. W rzeczywistości możesz już używać tych zasad do modelowania tego działania w swoich grach. Jeśli gracz traci szybkość skoku, możesz użyć wzorów na prędkość dla obliczenia  prędkości uderzenia gracza kiedy spadnie na ziemię. Na podstawie tej prędkości uderzenia możesz określić  odpowiednią ilość obrażeń do odliczenia od wyniku zdrowia gracza, lub jeśli wartość jest powyżej pewnego progu, możesz pożegnać się z twoim niedoszłym poszukiwaczem przygód!

Inne zastosowanie prostej kinematyki cząstek dotyczy pewnych efektów specjalnych, takich jak wybuchające cząstki. Ten rodzaj efektu jest dość prosty do implementacji i dodaje efektu realizmu do efektów wybuchu. Cząstki nie latają po prostu w przypadkowych, prostoliniowych trajektoriach. Zamiast tego wznoszą się i opadają pod wpływem ich początkowej prędkości, kąta  i przyspieszenia pod wpływem grawitacji, co sprawia ,że cząstki mają masę. kod dla tego przykładu wzięliśmy z omówionego wcześniej przykładu armaty więc wiele powinno wyglądać znajomo. Poniższy rysunek pokazuje główne okno programu




Efekt eksplozji ma miejsce w dużym, prostokątnym obszarze po prawej stronie  Nie pokazano eksplozji na tym ekranie, ponieważ wszystko co być zobaczył to pęczek kropek. W polach edycyjnych po lewej stronie określamy pozycję x i y dla tego efektu, wraz z początkową prędkością cząstek, która jest miarą siły wybuchu, czas trwania w milisekundach, współczynnik grawitacji i wreszcie kąt. Parametr kąta może być dowolną liczbą między 0 a 360 stopni lub 999. Kiedy podasz kąt  w zakresie od 0 do 360 stopni, wszystkie te cząstki w eksplozji będą uruchamiane generalnie w tym kierunku. Jeśli określisz wartość 999, wszystkie cząstki będą rozrzucane w losowych kierunkach. Parametr czasu trwania jest zasadniczo czasem życia  tego efektu.  Cząstki zanikają, gdy zbliżą się do tego życia. Pierwszą rzeczą , którą musisz zrobić dla tego przykładu, jest skonfigurowanie niektórych struktur i zmiennych globalnych do reprezentowania efektu cząsteczek i poszczególnych cząstek tworzących efekt wraz z początkowymi parametrami opisującymi zachowanie efektu. A oto kod:

```
//-----------------------------------------------------------------------------//
// Define a custom type to represent each particle in the effect.
//-----------------------------------------------------------------------------//
typedef     struct _TParticle
{
     float          x;          // x-coordinate of the particle
     float          y;          // y-coordinate of the particle
     float          vi;         // initial velocity
     float          angle;      // initial trajectory (direction)
     int               life;    // duration in milliseconds
     int               r;       // red component of particle's color
     int               g;       // green component of particle's color
     int               b;       // blue component of particle's color
     int               time;    // keeps track of the effect's time
     float          gravity;    // gravity factor
     BOOL          Active;      // indicates whether this particle
                                // is active or dead
} TParticle;

#define     _MAXPARTICLES 50

typedef struct _TParticleExplosion
{
     TParticle            p[_MAXPARTICLES];    // list of particles
                                               // making up this effect
     int              Vo; // initial velocity, or strength, of the effect
     int              x;  // initial x location
     int              y;  // initial y location
     BOOL             Active;       // indicates whether this effect is
                                    //active or dead
} TParticleExplosion;

//-----------------------------------------------------------------------------//
// Now define the variables required for this simulation
//-----------------------------------------------------------------------------//
TParticleExplosion     Explosion;

int                    xc;          // x-coordinate of the effect
int                    yc;          // y-coordinate of the effect
int                    Vo;          // initial velocity
int                    Duration;    // life in milliseconds
float                  Gravity;     // gravity factor (acceleration)
float                  Angle;       // indicates particles' direction
```

W kodzie tym możesz zobaczyć, że efekt wybuchu cząstek składa się z kolekcji cząstek. Zachowanie każdej cząstki określa się za pomocą kinematyki i początkowych parametrów ustawionych dla każdej cząsteczki. Kiedy naciśniesz przycisk GO, parametry początkowe , które określiłeś, zostaną użyte do zainicjowania eksplozji cząstek (jeśli naciśniesz przycisk Random, program losowo wybierze te parametry). Odbywa się to w funkcji o nazwie CreateParticleExplosion:

```
//////////////////////////////////////////////////////////////////////
/*      This function creates a new particle explosion effect.

      x,y:      starting point of the effect
      Vinit:    a measure of how fast the particles will be sent flying
                (it's actually the initial velocity of the particles)
      life:     life of the particles in milliseconds; particles will
                fade and die out as they approach their specified life
      gravity:  the acceleration due to gravity which controls the
                rate at which particles will fall as they fly
      angle:    initial trajectory angle of the particles,
                specify 999 to create a particle explosion
                that emits particles in all directions; otherwise,
                0 right, 90 up, 180 left, etc.
*/
void CreateParticleExplosion(int x, int y, int Vinit, int life,
                             float gravity, float angle)
{
      int    i;
      int    m;
      float f;

      Explosion.Active = TRUE;
      Explosion.x = x;
      Explosion.y = y;
      Explosion.V0 = Vinit;

      for(i=0; i<_MAXPARTICLES; i++)
      {

           Explosion.p[i].x = 0;
           Explosion.p[i].y = 0;
           Explosion.p[i].vi = tb_Rnd(Vinit/2, Vinit);

           if(angle < 999)
           {
                if(tb_Rnd(0,1) == 0)
                     m = -1;
                else
                     m = 1;
                Explosion.p[i].angle = -angle + m * tb_Rnd(0,10);
           } else
                Explosion.p[i].angle = tb_Rnd(0,360);

           f = (float) tb_Rnd(80, 100) / 100.0f;
           Explosion.p[i].life = tb_Round(life * f);
           Explosion.p[i].r = 255;//tb_Rnd(225, 255);
           Explosion.p[i].g = 255;//tb_Rnd(85, 115);
           Explosion.p[i].b = 255;//tb_Rnd(15, 45);


          Explosion.p[i].time = 0;
          Explosion.p[i].Active = TRUE;
          Explosion.p[i].gravity = gravity;
      }
}
```

Tu widać ,że cząstki są ustawione tak ,aby rozpoczynały się w tej samej pozycji, co określone przez współrzędne x i y, które podajesz; jednak możesz zauważyć ,że prędkość początkowa każdej cząstki jest faktycznie losowo wybierana z zakresu $V_{init}/2$ do $V_{init}$. Robimy to aby nadać zachowaniu cząstek taką samą różnorodność. Robimy to też dla parametru life każdej cząstki, tak ,że nie wszystkie cząstki zanikają i umierają dokładnie w tym samym czasie. Po utworzeniu eksplozji cząstek, program wchodzi w pętlę aby ją propagować i narysować efekt. Pętla ta jest pętlą while:

```
while(status)
{
     DrawRectangle(hBufferDC, &r, 1, RGB(0,0,0));
     status = DrawParticleExplosion(hBufferDC);
     hdc = GetDC(hSideView);
     if(!BitBlt(hdc, 0, 0, r.right, r.bottom, hBufferDC, 0, 0, SRCCOPY))
     {
          MessageBox(NULL, "BitBlt failed", "Error", MB_OK);
          status = FALSE;
     }
     ReleaseDC(hSideView, hdc);
}
```

Pętla while trwa tak długo , jak długo status pozostaje prawdziwy, co wskazuje ,ze efekt cząsteczkowy jest wciąż żywy. Po tym jak wszystkie cząsteczki w efekcie osiągną ustawiony czas życia, efekt będzie martwy a status zostanie ustawiony na fałsz. Wszystkie obliczenia zachować cząstek zachodzą w funkcji DrawParticleExpolosion; reszta kodu w pętli while służy do usuwania bufora ekranowego, a następnie skopiowanie go do głównego okna. DrawParticleExplosion aktualizuje stan każdej cząstki w efekcie wywołując inną funkcję, UpdateParticleState, a potem rysuje efekt do bufora pozakeranowego przekazywanego w niej jako parametr. Oto jak wyglądają dwie funkcje:

```
//-----------------------------------------------------------------------------//
// Draws the particle system and updates the state of each particle.
// Returns false when all of the particles have died out.
//-----------------------------------------------------------------------------//
BOOL     DrawParticleExplosion(HDC hdc)
{
      int    i;
      BOOL    finished = TRUE;
      float              r;
      COLORREF    clr;

      if(Explosion.Active)
        for(i=0; i<_MAXPARTICLES; i++)
        {
           if(Explosion.p[i].Active)
```

```
            {
                finished = FALSE;
            r = ((float)(Explosion.p[i].life-
                  Explosion.p[i].time)/(float)(Explosion.p[i].life));
            clr = RGB(tb_Round(r*Explosion.p[i].r),
                  tb_Round(r*Explosion.p[i].g),
                  tb_Round(r*Explosion.p[i].b));
            DrawCircle(    hdc,
                           Explosion.x+tb_Round(Explosion.p[i].x),
                           Explosion.y+tb_Round(Explosion.p[i].y),
                           2,
                           clr);
            Explosion.p[i].Active = UpdateParticleState(&(Explosion.p[i]),
                                                        10);
            }
        }

      if(finished)
          Explosion.Active = FALSE;

      return !finished;
}

//-------------------------------------------------------------------------//
/* This is generic function to update the state of a given particle.
    p:          pointer to a particle structure
    dtime:      time increment in milliseconds to
                advance the state of the particle

    If the total elapsed time for this particle has exceeded the particle's
    set life, then this function returns FALSE, indicating that the particle
    should expire.
*/
BOOL    UpdateParticleState(TParticle* p, int dtime)
{
    BOOL retval;
    float     t;

    p->time+=dtime;
    t = (float)p->time/1000.0f;
    p->x = p->vi * cos(p->angle*PI/180.0f) * t;
    p->y = p->vi * sin(p->angle*PI/180.0f) * t + (p->gravity*t*t)/2.0f;

    if (p->time >= p->life)
        retval = FALSE;
    else
        retval = TRUE;

    return retval;
}
```

UpdateParticleState wykorzystuje kinematyczne wzory, które już widziałeś ,aby zaktualizować położenie cząstek jako funkcję prędkości początkowej ,czasu i przyspieszenia ze względu na grawitację. Po wywołaniu UpdateParticleState, DrawParticleState skaluje kolor każdej cząstki, zmniejszając ją do czerni, w oparciu o czas życia każdej cząstki i upływający czas. Efekt zanikania polega na tym ,że cząstki powoli umierają z upływem czasu, a nie po prostu znikają z ekranu. Efekt przypomina zachowanie fajerwerków rozbłyskujących na nocnym niebie.

**Kinematyka Ciała Sztywnego**

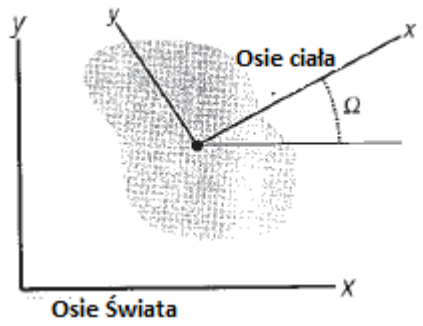
Wzory dla przemieszczenia, prędkości i przyspieszenia omówione wcześniej, dotyczą tak ciał stałych jak i cząstek. Różnica jest taka ,że biorąc pd uwagę ciała sztywne, punkt ciała sztywnego, który śledzisz pod kątem ruchu liniowego, jest środkiem masy ciała (ciężar). Kiedy ciało sztywne przekłada się bez obrotu, wszystkie cząstki tworzące ciało sztywne poruszają się po ścieżkach równoległych, ponieważ ciało nie zmienia swojego kształtu. Ponadto, gdy ciało sztywne obraca się, generalnie obraca się wokół osi, które przechodzą przez jego środek masy, chyba ,że ciało jest zawieszone w punkcie, wokół którego jest zmuszony się obracać. Fakty te sprawiają, że środek masy jest wygodnym punktem do śledzenia jego ruchu liniowego To dobra wiadomość dla Ciebie, ponieważ możesz wykorzystać całą wiedzę dotychczasową dotyczącą kinematyki cząstek do badania kinematyki ciała sztywnego. Procedura postępowania z ciałami sztywnymi obejmuje dwa różne aspekty :

- śledzenie translacji środka masy ciała i
- śledzenie obrotu ciała

Pierwszy aspekt jest dość stary - po prostu potraktuj ciało jako cząstkę; jednak drugi aspekt wymaga rozważenia kilku innych pojęć, a mianowicie współrzędnych lokalnych, przemieszczenia kątowego, prędkości kątowej i przyspieszenia kątowego. Będziemy omawiali kinematykę płaskich ciał sztywnych Ruch płaski oznacza po prostu , że ruch ciała jest ograniczony do płaskiej płaszczyzny w przestrzeni, w której jedyna oś obrotu wokół której obraca się ciało, jest prostopadła do płaszczyzny. Ruch płaski jest zasadniczo dwuwymiarowy. Pozwala nam to skupić się na nowych koncepcjach kinematycznych kątowego przemieszczenie, prędkości i przyspieszenia bez zagłębiania się w matematyce wymaganej do opisania arbitralnej rotacji w trzech wymiarach. Możesz być zaskoczony jak wiele problemów nadaje się do rozwiązań kinematycznych. Na przykład, w pewnych popularnych „strzelankach, twoja postać może pchać obiekty, takie jak pudła i beczki po podłożu. Chociaż gra jest trójwymiarowa, te szczególne obiekty są ograniczone do przesuwania po podłożu, płaszczyźnie, zatem mogą być traktowane jako problem 2D. Nawet jeśli gracz popycha te obiekty pod pewnym kątem, a nie prosto, będzie można symulować przesuwanie i obracanie tych obiektów za pomocą technik kinematyki (i kinetyki) 2D.

**Lokalne Osie Współrzędnych**

Wcześniej zdefiniowaliśmy kartezjański układ współrzędnych, którego używamy do ustalonego globalnego odniesienia lub współrzędnych świata. Ten światowy układ współrzędnych jest wszystkim co jest wymagane, gdy zajmujemy się cząstkami; jednak w przypadku ciał sztywnych, użyjemy zbioru lokalnych współrzędnych przymocowanych do ciała. Szczególnie ten lokalny układ współrzędnych zostanie ustawiony na środek ciężkości ciała. Użyjemy tego systemu współrzędnych do śledzenia orientacji ciała podczas jego obracania. Dla ruchu płaskiego potrzebujemy tylko jednej wielkości skalarnej do opisania orientacji ciała, co zilustrowano



W tym przypadku, orientacja, Q, jest definiowana jako różnica kątowa między dwoma zbiorami osi współrzędnych: stałymi osiami światowymi a lokalnymi osiami ciała. Jest to tzw. kąt Eulera. W ogólnym ruchu 3D, występują łącznie trzy kąt Eulera, które zwykle nazywa się odchyleniem, pochyleniem i przechodzeniem, w żargonie aerodynamicznym i hydrodynamicznym. Chociaż te reprezentacje kątowe są łatwe do wizualizacji pod względem ich fizycznego znaczenia, nie są tak ładne z liczbowego punktu widzenia.

**Prędkość Kątowa I Przyspieszenie**

W ruchu w płaszczyźnie 2D, gdy ciało się obraza, Ω będzie się zmieniać, a współczynnik przy którym Ω się zmienia, jest prędkością kątową ω. Podobnie, współczynnik z jakim zmienia się ω jest przyspieszeniem kątowym α. Te właściwości kątowe są analogiczne do liniowych właściwości przemieszczania, prędkości i przyspieszenia. Jednostki dla kątowego przemieszczenia , prędkości i przyspieszenia to radiany (rad), radiany na sekundę (rad/s) i radiany na sekundę do kwadratu (rad/$s^2$), odpowiednio. Matematycznie, możesz zapisać te związki między kątowym przemieszczeniem, prędkością kątową i kątowym przyspieszeniem.

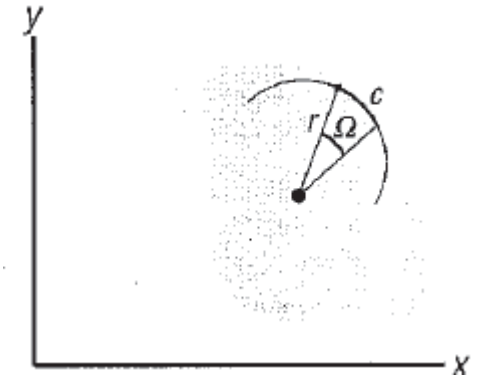
$$\omega = d\Omega/dt$$
$$\alpha = d\omega/dt = d^2\Omega/dt^2$$
$$\omega = \int \alpha\, dt$$
$$\Omega = \int \omega\, dt$$
$$\omega\, d\omega = \alpha\, d\Omega$$

W rzeczywistości możesz zastąpić właściwości kątowe Ω, ω i α dla linowych właściwości s,v i a w równaniach uzyskanych wcześniej dla kinematyki cząstek aby uzyskać podobne równania kinematyczne dla obrotu. Przy stałym przyspieszeniu kątowy, otrzymasz następujące równania:

$$\omega_2 = \omega_1 + \alpha t$$
$$\omega_2^2 = \omega_1^2 + 2\alpha(\Omega_2 - \Omega_1)$$
$$\Omega_2 = \Omega_1 + \omega_1 t + (1/2)\alpha t^2$$

Kiedy ciało sztywne obraca się wokół danej osi, każdy punkt na ciele sztywnym przesuwa się okrągłą ścieżką wokół osi obrotu. Możesz pomyśleć o obrocie ciała jako powodującego dodatkowy ruch liniowy każdej cząstki tworzącej ciało. Ten ruch liniowy jest dodatkiem co ruchu liniowego środka ciężkości ciała. Aby uzyskać całkowity ruch liniowy dowolnego elementu lub punktu na ciele sztywnym, musisz być w stanie powiązać ruch kątowy ciała z ruchem liniowym cząstki lub punktu jeśli przebywa swoją okrągłą ścieżkę wokół osi obrotu. Zasadniczo w dynamice, wiedza ,że dwa obiekty zderzyły się ze sobą, czasami nie wystarcza, i chcesz wiedzieć ,jak ciężko , by tak rzec, te dwa obiekty się zderzyły. Kiedy mamy do czynienia ze współpracującymi ciałami sztywnymi, które mogą w pewnym momencie stykać się ze sobą lub innymi stałymi obiektami, musisz określić nie tylko położenie tych punktów styku, ale również względną prędkość lub przyspieszenie między punktami styku. Informacja ta pozwoli ci obliczyć siły interakcji między zderzającymi się ciałami. Długość łuku ścieżki omiatanej przez cząstkę na ciele sztywnym jest funkcją odległości od osi obrotu do cząstki i przemieszczenia kątowego, Ω. Użyjemy c dla określenia długości łuku i r dla określenia odległości od osi obrotu do cząstki. Wzór odnoszący się do długości łuku do przemieszczenia kątowego

$c = r\Omega$



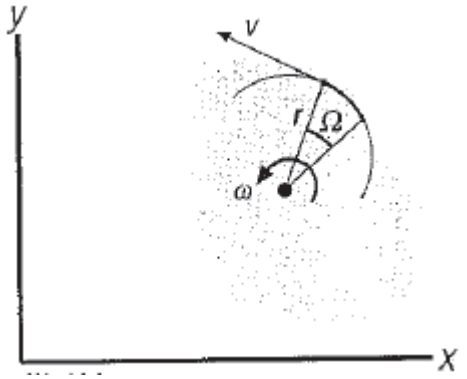

gdzie Ω musi być w radianch, nie stopniach. Jeśli scakujemy ten wzór w odniesieniu do czasu

dc/dt = rdΩ/dt

otrzymujesz równanie przeliczające prędkość liniową cząstki , która porusza się wzdłuż okrągłej ścieżki do prędkości kątowej ciała stałego. To równanie jest napisane w następujący sposób:

v = rω

Ta prędkość jako wektor jest styczna do okrągłej ścieżki omiatanej przez element. Jeśli możesz sobie wyobrazić tę cząstkę jako kule na końcu pręta którego drugi koniec jest przymocowany do obracającej się osi, następnie gdy kula zostanie zwolniona z pręta, kula odleci w kierunky stycznym do kolistej ścieżki, którą pokonywała gdy była przymocowana do pręta. Jest to ten sam kierunek ,co prędkość liniowa styczna podana w powyższym równaniu.



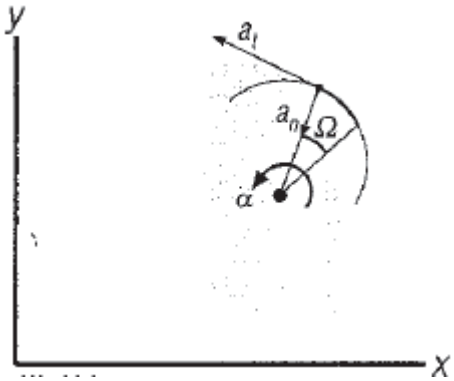

Całkując równanie, v = rω:

dv/dt = r dω/dt

uzyskujemy wzór dla stycznego przyspieszenia liniowego w funkcji kątowego przyspieszenia:

$a_t = r\alpha$

Zauważ ,ze istnieje inne składnik przyspieszenia dla cząstki, który wynika z obrotu sztywnego ciała. Ten komponent jest normalny lub prostopadły ro kolistej ścieżki cząstki i jest nazywany przyspieszeniem dośrodkowym, które jest zawsze skierowane w kierunku osi obrotu




Pamiętaj ,że prędkość jest wektorem , a ponieważ przyspieszenie jest wskaźnikiem zmian w wektorze prędkości, istnieją dwa sposoby w jaki przyspieszenie można osiągnąć. Jednym ze sposobów jest zmiana wielkości wektora prędkości , tzn. zmiana szybkości; drugim sposobem jest zmiana kierunku wektora prędkości . Zmiana prędkości powoduje powstanie przyspieszenia stycznego, podczas gdy zmiana kierunku powoduje wzrost składnika przyspieszenia dośrodkowego. Powstały wektor przyspieszenia jest sumą wektorową przyspieszeń stycznego i dośrodkowego. Przyspieszenie dośrodkowe jest tym, co czujesz, kiedy jedziesz swoim samochodem wąskim zakrętem, nawet jeśli twoja prędkość jest stała.

Wzór na wielkość przyspieszenia dośrodkowego $a_n$ ,to

$a_n = v^2 / r$

gdzie v jest prędkością styczną. Podstawienie równania dla prędkości stycznej do tego równania dla przyspieszenia dośrodkowego daje następującą formę alternatywną:

$a_n = r\omega^2$

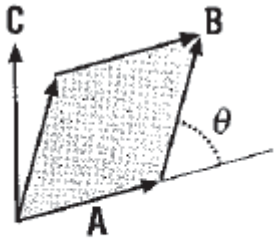
W dwóch wymiarach, można łatwo uciec używając tych równań skalarnych; jednak, w trzech wymiarach musisz użyć form wektorowych tych równań. Prędkość kątowa jako wektor jest równoległa do osi obrotu. Sens lub kierunek obrotu określa zasada prawej ręki. Jeśli weźmiesz prawą rękę i zwiniesz palce w łuk wokół osi obrotu, gdzie palce wskazują kierunek w którym obraca się ciało, kciuk powinien wskazywać wektora prędkości kątowej. Jeśli weźmiemy wektor iloczynu wektorowej, wektor prędkości kątowej i wektor do osi obrotu rozważanej cząstki, otrzymasz liniowy, styczny wektor prędkości. Zapisujemy to

$v = \omega \times r$

Zauważ ,że daje to zarówno wielkość i kierunek liniowy, prędkości stycznej. Pamiętaj również ,aby zachować porządek wektorów przy iloczynie wektorowym, to znaczy , ω razy r, a nie na odwrót, co dałoby zły kierunek dla v

***Iloczyn Wektoroy***

Biorąc pod uwagę dowolne dwa wektory A i B, iloczyn wektorowy A x B jest definiowany przez trzeci wektor z wielkością równą $AB \sin\theta$, gdzie θ jest kątem między dwoma wektorami A i B



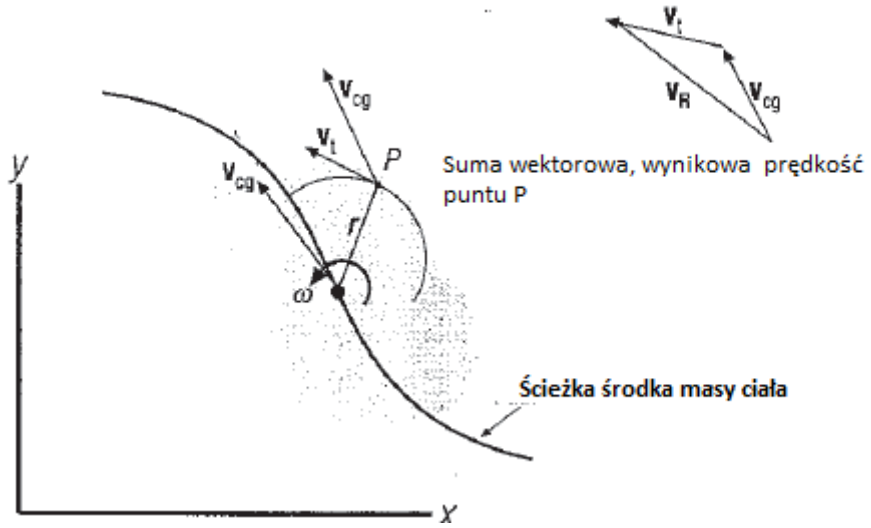

$$\mathbf{C} = \mathbf{A} \times \mathbf{B}$$
$$\mathbf{C} = AB \sin\theta$$

Kierunek C jest określony przez regułę prawej ręki. Reguła prawej ręki jest prostą sztuczką, która pomaga śledzić kierunki wektorowe. Załóżmy ,że A i B leżą w płaszczyźnie, i niech oś obrotu biegnie prostopadle do tej płaszczyzn przez punkt umieszczony na końcu A. Weź prawą rękę i udawaj ,że zwijasz palce wokół osi obrotu od wektora A w kierunku B. Teraz wyciągnij kciuk, jak gdybyś unosił go w górę, trzymając palce zwinięte wokół osi. Kierunek który wskazuje Twój kciuk, wskazuje kierunek wektora C. Na powyższym rysunku, równoległobok jest formowany przez A i B (zacieniony obszar). Obszar równoległoboku jest modułem z C, co jest AB sinθ.

Istnieją dwa równania potrzebne do określenia wektorów dla przyspieszenia stycznego i dośrodkowego:

$$\mathbf{a}_n = \omega \times (\omega \times \mathbf{r})$$
$$\mathbf{a}_t = \alpha \times \mathbf{r}$$

Innym sposobem patrzenia na wielkości v, $a_n$ i $a_t$ , jest to ,że są one prędkościami i przyspieszeniem cząstki pod względem rozważań o ciele sztywnym w stosunku do punktu wokół którego obraca się ciało sztywne, na przykład , środek masy ciała. Jest to bardzo wygodne, ponieważ będziesz chciał śledzić ruch ciała sztywnego jako cząstki kiedy patrzymy na duży obraz bez martwienia się o to ,co każda cząstka ciała stałego robi w danym czasie. Dlatego ruch liniowy ciała sztywnego i jego ruch kątowy oddzielnie. Kiedy musisz przyjrzeć się konkretnym cząstkom ciała sztywnego, możesz to zrobić wykonując ruch ciała sztywnego jako cząstki, a następnie dodając do niego ruch względny rozpatrywanego punktu. Poniższy rysunek pokazuje ciało sztywne poruszające się z prędkością $v_{cg}$ ,gdzie $v_{cg}$ jest szybkością środka masy (lub środkiem ciężkości) ciała sztywnego. Pamiętaj ,że środek masy jest punktem do śledzenia kiedy traktujesz ciało sztywne jako cząstkę. To ciało sztywne również obraca się z prędkością kątową ω wokół osi przechodzącej przez środek masy ciała. Wektor r jest wektorem od środka masy ciała sztywnego do określonego punktu, P umieszczonego na ciele sztywnym




W tym przypadku wypadkowa prędkość punktu P może być znaleziona, przyjmujemy wektor sumy prędkości środka masy ciała i prędkości stycznej punktu P w zależności od prędkości kątowej ciała ω. Oto jak wyglądają równania wektorowe

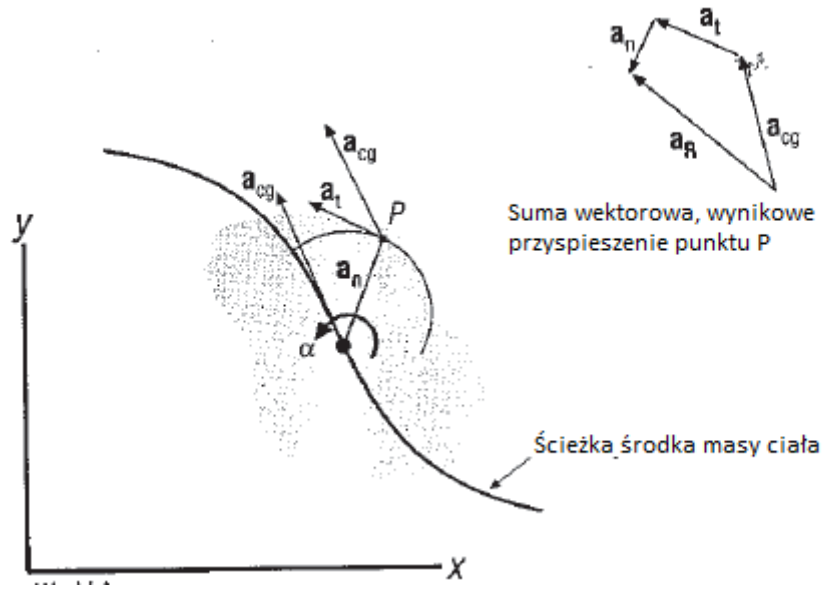
$$\mathbf{v}_R = \mathbf{v}_{cg} + \mathbf{v}_t$$

lub

$$\mathbf{v}_R = \mathbf{v}_{cg} + (\omega \times \mathbf{r})$$

Możesz zrobić to samo z przyspieszeniem, aby określić przyspieszenie wypadkowe punktu P. Weźmy sumę wektorową przyspieszenia środka masy ciała sztywnego, przyspieszenia stycznego ze względu na przyspieszenie kątowe ciała i przyspieszenie dośrodkowe ze względu na zmiany w kierunku prędkości kątowej. Równanie :

$$\mathbf{a}_R = \mathbf{a}_{cg} + \mathbf{a}_n + \mathbf{a}_t$$




Możesz przepisać równania dla wynikowego przyspieszenia w poniższej formie

$$\mathbf{a}_R = \mathbf{a}_{cg} + (\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r})) + (\boldsymbol{\alpha} \times \mathbf{r})$$

Jak widać, użycie tych zasad względnej prędkości i przyspieszenia pozwala ci obliczyć wynikowe właściwości kinematyczne dowolnego punktu na ciele sztywnym w danym momencie, poprzez poznanie tego co środek ciężkości ciała robi, w czasie gdy ciało jest obracane.