Box2D

Collision Module
Dynamics Module

Collision Module

Collision Module zawiera kod i struktury danych do wykrywania kolizji.

Organizacja modułu:

- b2BroadPhase.cpp
- b2BroadPhase.h
- b2CollideCircle.cpp
- b2CollideEdfe.cpp
- b2CollidePolygon.cpp
- b2Collision.cpp
- b2Collision.h
- b2Distance.cpp
- b2Distance.h
- b2DynamicTree.cpp
- b2DynamicTree.h
- b2TimeOfImpact.cpp
- b2TimeOfImpact.h

Shapes

- b2ChainShape.cpp
- b2ChainShape.h
- b2CircleShape.cpp
- b2CircleShape.h
- b2EdgeShape.cpp
- b2Edgeshape.h
- b2PolygonShape.cpp
- b2PolygonShape.h
- b2Shape.h

Klasa b2Shape jest klasą bazową dla czterech klas konstruujących geometryczny opis obiektów.

b2CircleShape

Klasa konstruująca kształt kół (nie okręgów!). Do opisu kształtu koła wystarczy podanie promienia, domyślnie centrum masy znajduje się w środku obiektu.

Kod tworzący koło o średnicy 2 jednostek:

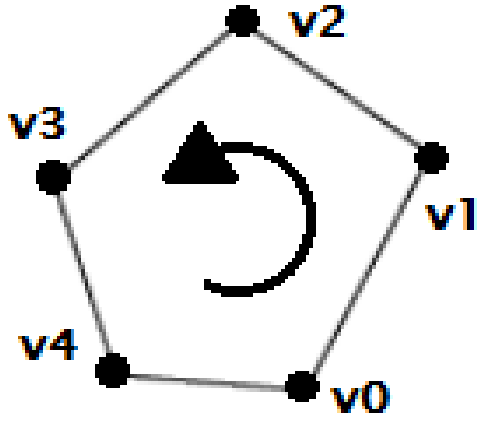
```
b2CircleShape circle;
circle.m_radius = 2.0f;
circle.m_p.Set(2.0f, 3.0f);
```

b2PolygonShape

Klasa konstruująca wielokąt zamknięty, którego wnętrze jest częścią obiektu. Do opisu kształtu podaje się listę punktów, które będą tworzyć dany wielokąt.

Przy tworzeniu wielokąta musi być spełnionych kilka warunków:

- Wierzchołki muszą być podawane w kolejności odwrotnej do ruchu wskazówek zegara (CCW)
- Liczba wierzchołków musi być w przedziale od 3 do 8
- Odległość między wierzchołkami musi być większa niż kwadrat z wartości `b2_linearSlop(0.005f)`
- Opisywany wielokąt musi być wypukły



Przykładowy kod tworzący wielokąt (tutaj trójkąt):

```
b2Vec2 vertices[3]; // tablica wierzchołków
vertices[0].Set(0.0f, 0.0f);
vertices[1].Set(1.0f, 0.0f);
vertices[2].Set(0.0f, 1.0f);
int32 count =3;

b2PolygonShape polygon; // tworzymy wielokąt
polygon.Set(vertices, count);
```

Istnieje również pomocnicza metoda tworząca prostokątny kształt – zamiast `Set()` można użyć metody:

```
polygon.SetAsBox(float half_width, float half_height);
```

lub drugiej wersji pozwalającej na nieco większą kontrolę nad pozycją i orientacją tworzonego prostokąta:

```
polygon.SetAsBox(float hald_width, float half_height,
                 const b2Vec2& center, float angle);
```

b2EdgeShape

Klasa konstruująca odcinek. Odcinki mogą wchodzić w kolizję z okręgami oraz wielokątami, ale nie mogą z innymi odcinkami.

Przykładowy kod:

```
b2Vec2 v1(0.0f, 0.0f);
b2Vec2 v2(1.0f, 0.0f);

b2EdgeShape edge;
edge.Set(v1,v2);
```

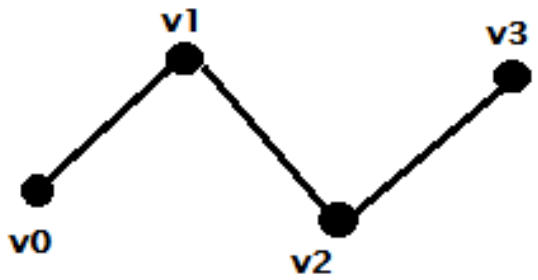
b2ChainShape

Klasa konstruująca łamaną z listy punktów. Wymagane jest, aby odległość między punktami była większa niż kwadrat z wartości `b2_linearSlop`.

Przykładowy kod tworzący łamaną otwartą:

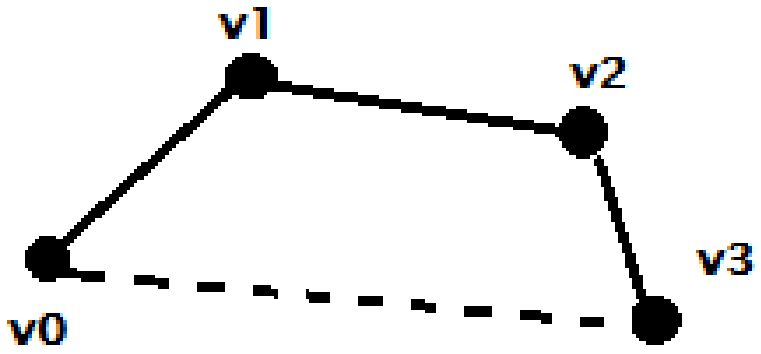
```
b2Vec2 vs[4];
vs[0].Set(1.7f, 0.0f);
vs[1].Set(1.0f, 0.25f);
vs[2].Set(0.0f, 0.0f);
vs[3].Set(-1.7f, 0.4f);

b2ChainShape chain;
chain.CreateChain(vs, 4);
```



Tworzenie łamanej zamkniętej:

```
b2ChainShape chain;
chain.CreateLoop(vs, 4);
```

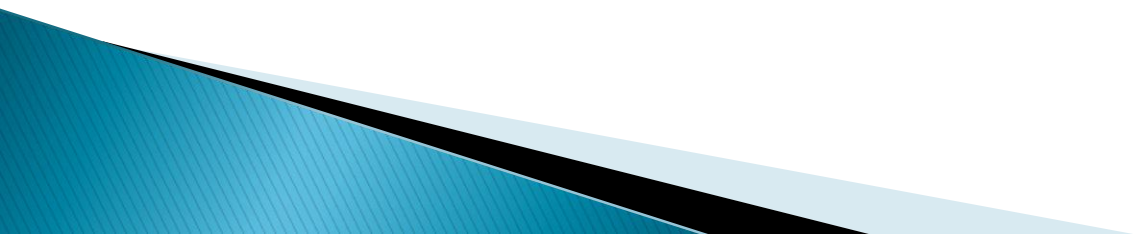


Shapes – Podsumowanie

Klasy `b2CircleShape` oraz `b2PolygonShape` tworzą obiekty, które mają określoną powierzchnię i mogą mieć przypisaną masę (warunek konieczny, aby ciało mogło mieć cechy ciała dynamicznego lub kinematycznego), a ich wnętrze jest również częścią obiektu.

Klasy `b2EdgeShape` oraz `b2ChainShape` tworzą obiekty nieposiadające powierzchni w sensie matematycznym, dlatego mogą być tylko ciałami statycznymi.

Kolizje w Box2D



Odpytywanie ciał

Jedną z metod wykrywania kolizji w Box2D jest odpytywanie interesujących nas ciał, czy uczestniczyły w kolizji. Każde ciało ma listę, w której trzyma obiekty będące w stanie kolizji z odpytywanym ciałem. Listy w ciałach są aktualizowane w każdym kroku symulacji.

```
for(b2ContactEdge *ce = body->GetContactList();
    ce != NULL; ce = ce->next)
    {
      if(ce->contact->IsTouching())
      {
         ce->other; // ciało, z którym koliduje odpytywany obiekt
      }
    }
```

Ciało znajduję się na liście kolizji już w momencie, kiedy ich obrysy prostokątne AABB (Axis Aligned Bounding Box) nakładają się na siebie.

AABB wykorzystuje się ze względu na szybkość i prostotę wykonywania testów na nakładanie się prostokątów na siebie.

Pobieranie listy kolizji

Ta metoda sprawdzania kolizji pobiera listę wszystkich kolizji występujących w danym kroku w symulowanym świecie.

Lista zamiast `b2ContactEdge` (metoda odpytywania ciał) zwraca obiekt `b2Contact`.

```
for(b2Contact *c = m_world->GetContactList();
    ce != NULL; ce = ce->next)
    {
      if(c->IsTouching())
      {
         //para kolidujących ciał
        b2Body *bodyA = c->GetFixtureA()->GetBody();
        b2Body *bodyB = c->GetFixtureB()->GetBody();
      }
    }
```

Nasłuchiwanie kolizji

Inną metodą jest zastosowanie obiektu nasłuchiwania kolizji – `b2ContactListener`.

Box2D pozwala na zarejestrowanie takiego obiektu metodą:

```
b2World::SetContactListener
(b2ContactListener* contactListener);
```

W przypadku kolizji wywołuje, w zależności od sytuacji, jedną z czterech metod.

Metody klasy b2ContactListener:

- `BeginContact (b2Contact*)`

  Jest wywoływana, gdy następuję kontakt między dwoma ciałami

- `EndContact(b2Contact*)`

  Jest wywoływana, gdy ciała tracą kontakt

- `PreSolve(b2Contact*, b2Manifold*)`

  Jest wywoływana przed obliczeniami związanymi z reakcją ciał na kontakt (prędkością, odbiciem, tarciem, itp.)

- `PostSolve(b2Contact*, b2ContactImpulse*)`

  Jest wywoływana po obliczeniach związanych z kontaktem ciał.

Raycasting

Raycasting polega na odpytywaniu obiektu świata, na jakie obiekty w świecie trafia zdefiniowany promień.

```
Void b2World::RayCast(b2RayCastCallback* callback,
                      const b2Vec2& point1,
                      const b2Vec2& point2);
```

Nachodzenie obiektów

Do sprawdzenia, czy dwa obiekty AABB nachodzą na siebie, można wykorzystać funkcję:

```
Bool b2testOverlap(const b2AABB& aabb_0,
                   const b2AABB& aabb_1);
```

Aby sprawdzić, czy dwa obiekty geometryczne b2Shape nakładają się na siebie, można wykorzystać powyższą metodę w przeciążonej wersji.

```
Bool b2testOverlap(const b2Shape* shapeA, int32 indexA,
                   const b2Shape* shapeB, int32 indexB,
                   const b2Transform& xfA,
                   const b2Transform& xfB);
```

Dynamics Module

Dynamics Module definiuje zasady świata fizycznego.

Organizacja modułu:

- b2Body.cpp
- b2Body.h
- b2ContactManager.cpp
- b2ContactManager.h
- b2Fixture.cpp
- b2Fixture.h
- b2Island.cpp
- b2Island.h
- b2TimeStep.h
- b2World.cpp
- b2World.h
- b2WorldCallbacks.cpp
- b2WorldCallback.h