Fizyka i silniki fizyczne
nVidia PhysX

Część 1: Wprowadzenie
(PhysX 3.4)

nVidia PhysX – wprowadzenie

Początkowo nazwa **PhysX** oznaczała kartę rozszerzeń firmy **Ageia** służącą do sprzętowego wspomagania obliczeń fizycznych na potrzeby gier komputerowych.

W roku 2008 firmę **Ageia** przejęła **nVidia**. Funkcje sprzętowego wspomagania obliczeń fizycznych przejął procesor GPU kart graficznych serii **GeForce** (na bazie technologii CUDA).

Nazwa **PhysX** oznacza obecnie pakiet programistyczny (SDK) wspierający obliczenia fizyczne w przestrzeni 3D.

PhysX API

Istnieją dwie główne, niekompatybilne ze sobą wersje interfejsu API biblioteki PhysX:

- **Wersja 2** – ostatnia dostępna wersja to 2.8.7,
- **Wersja 3** – udostępniona w 2011 roku (wersja 3.0).

Wersja 2.8.x biblioteki nie jest już oficjalnie wspierana, i nie powinna być używana w nowych projektach.

PhysX 2.x -> 3.x

Najważniejsze zmiany w API w wersji 3 w porównaniu do 2.8.x:

- Nowa konwencja nazw klas, struktur i funkcji (Px... zamiast Nx...)
- Nowa hierarchia klas aktorów
- Elastyczne filtrowanie kolizji (na bazie „shaderów”)
- Bezpośrednia reprezentacja materiałów (klasa PxMaterial)
- Szkielet obiektu nie jest już potrzebny do ciągłej detekcji kolizji (CCD)
- Określenie pozycji wykorzystuje teraz wektor translacji i kwaternion rotacji w nowej strukturze PxTransform
- Zmieniony sposób opisu kształtu aktorów
- Nowa hierarchia zawiasów, brak napędu w zawiasie typu D6

Ograniczenia symulacji w PhysX API 3

W stosunku do poprzedniej generacji technologii PhysX (2.8.x), symulacja bardziej złożonych zjawisk została uproszczona, zoptymalizowana i częściowo usunięta z samej biblioteki.

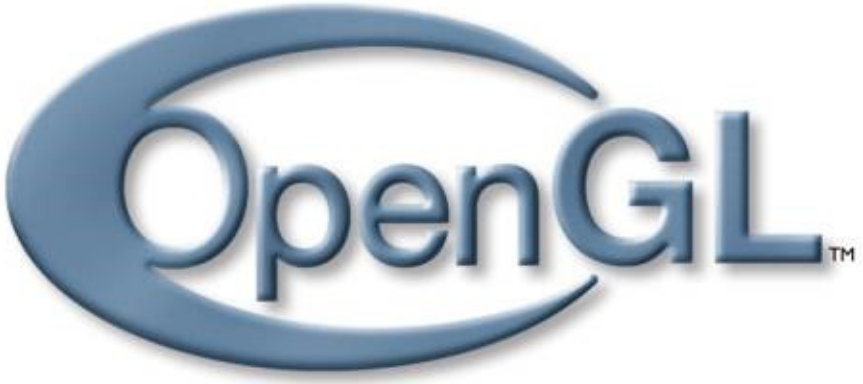
PhysX w wersji 3 **nie obsługuje** m.in.:

- symulacji ciał miękkich (ang. *softbody*),
- rozrywania tkanin (ang. *tearing*),
- dwu-kierunkowej interakcji tkanin ze światem,
- innych...

Część wyciętych funkcji przeniesionych została do dodatkowych narzędzi dostarczanych równolegle z PhysX SDK (np. NVIDIA Blast, NVIDIA FleX).

Podstawy programowania z użyciem PhysX 3

OpenGL i PhysX

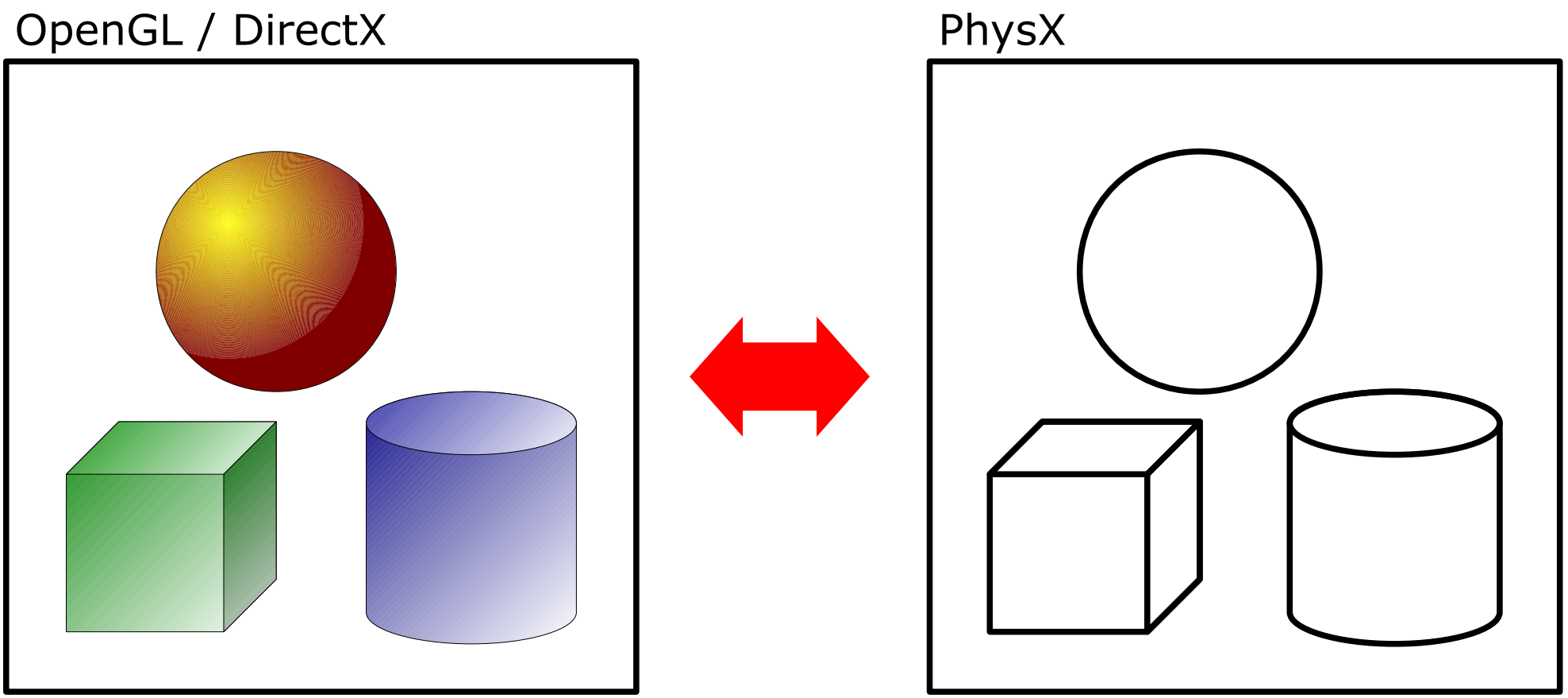


Niezbędne będą:

- **nVidia PhysX SDK** (do ściągnięcia ze strony nVidii po uprzednim zarejestrowaniu się);
- **Microsoft Visual C++** (wystarczy darmowa wersja Express Edition).

Obiekty i aktory

Obiekty OpenGL które mają podlegać symulacji należy skojarzyć z odpowiednimi **aktorami PhysX**!



Do kształtu obiektu OpenGL należy dopasować jeden z dostępnych kształtów zdefiniowanych dla aktorów PhysX, lub ich grupę.

Symulacja i animacja

PhysX symuluje zachowanie aktorów **na podstawie czasu** uaktualnianego przez aplikację.

Pozycje aktorów PhysX dla danej chwili czasu mogą zostać odczytane i zamienione na **macierz transformacji** OpenGL, która posłuży do narysowania obiektu w odpowiedniej pozycji.

Konwencja nazw PhysX w wersji 3

Wszystkie funkcje, własne typy danych i klasy PhysX mają przedrostek **„Px”**, np. **PxActor**, **PxVec3**, **PxScene**...

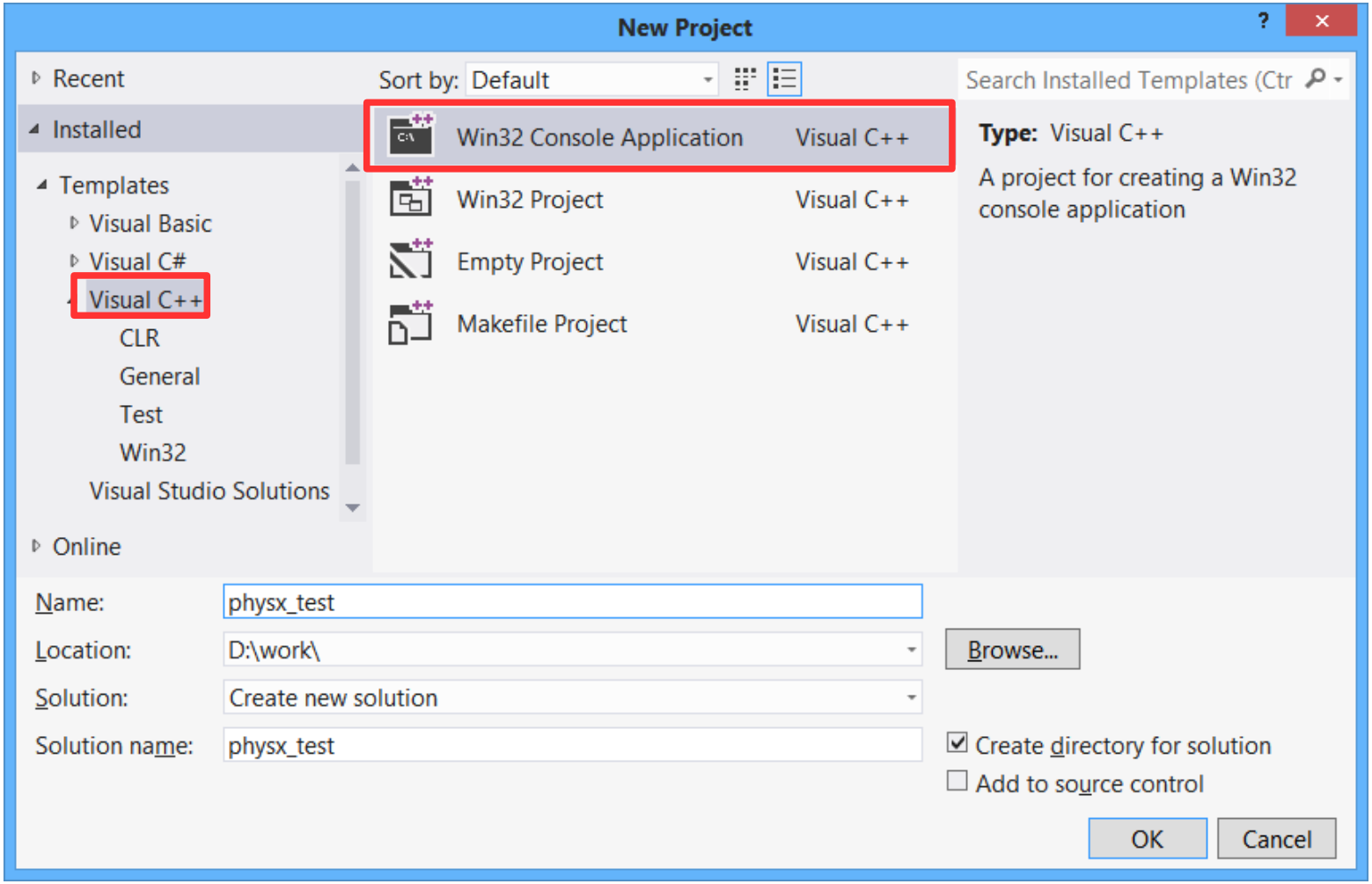
Pomocnicze klasy i typy proste (wybrane)

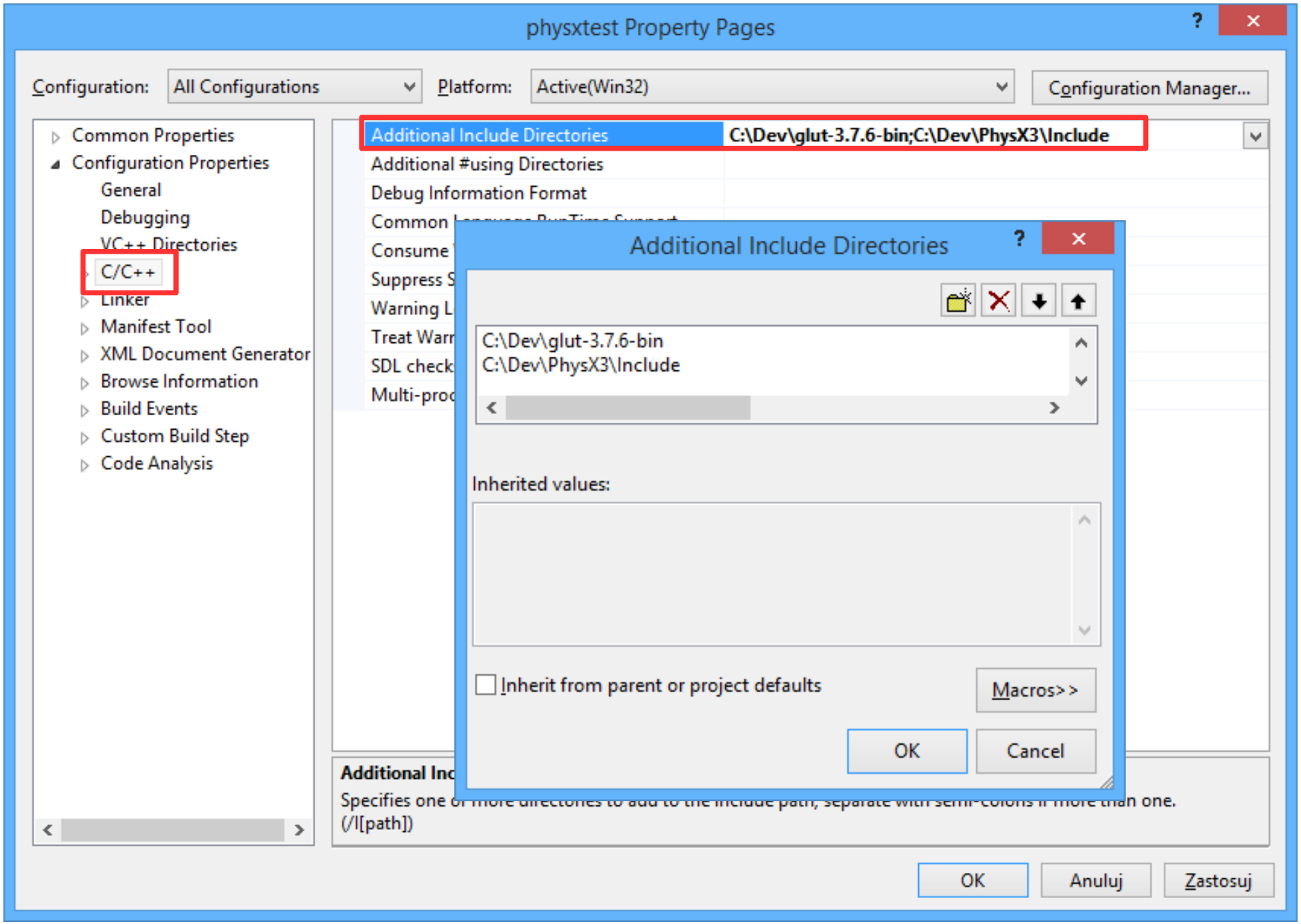
Nazwa klasy	Opis
PxVec3	Wektor 3-elementowy.
PxMat33, *PxMat44*	Macierz 3x3 i 4x4.
PxQuat	Kwaternion reprezentujący obroty.

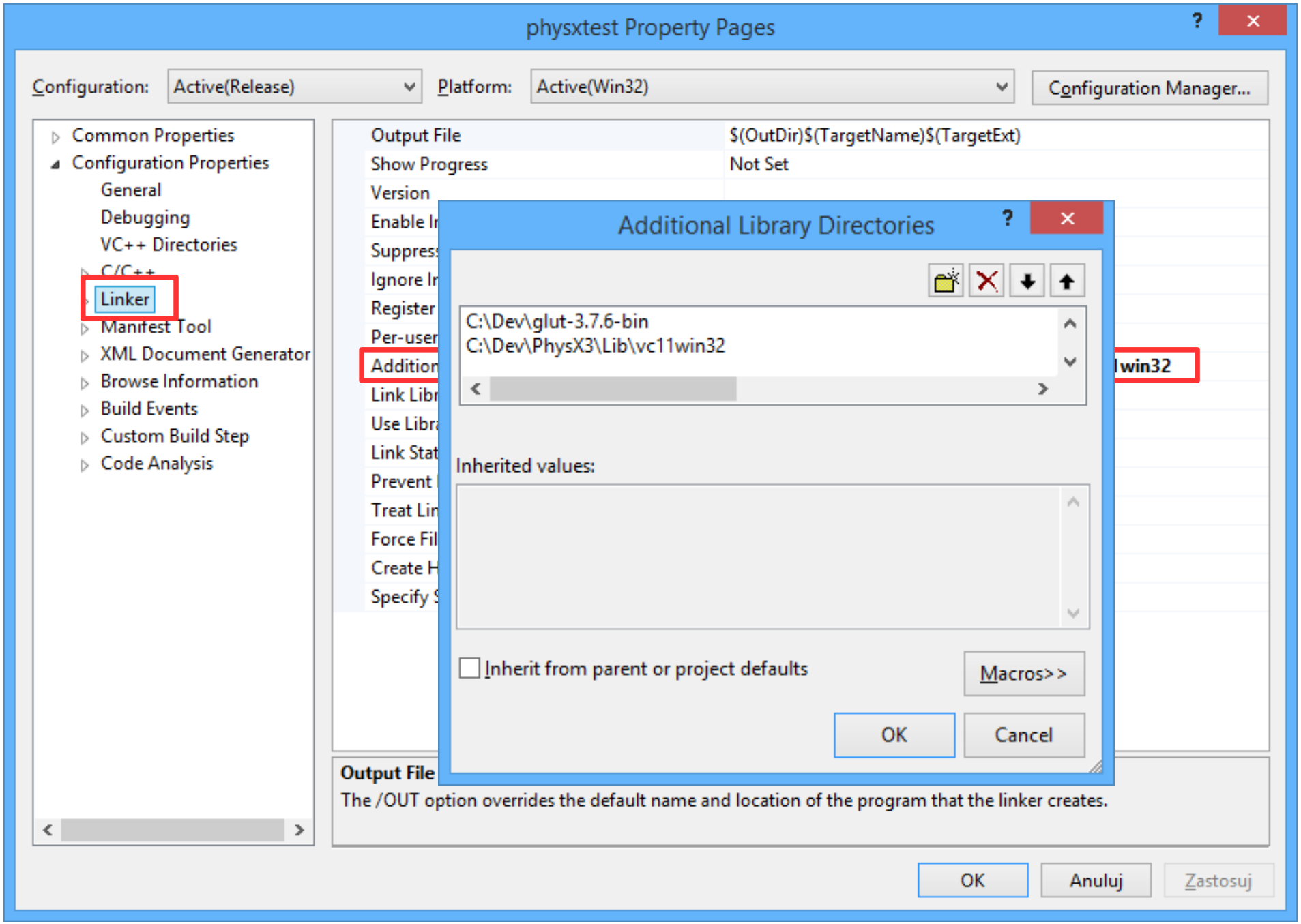
Nazwa typu	Typ w języku C
PxReal	float
PxF32	float
PxF64	double
PxU32	unsigned int
PxI32	signed int

Nazwa klasy	Opis
PxPhysics	Klasa reprezentuje silnik PhysX, każdy program musi utworzyć jeden egzemplarz obiektu tej klasy.
PxScene	Klasa reprezentuje scenę. Scena symuluje zachowanie skojarzonych z nią obiektów. Może jednocześnie istnieć wiele scen.
PxSceneDesc	Opis sceny (scene description), używany przy definiowaniu i tworzeniu sceny.
PxActor	Klasa bazowa dla wszystkich aktorów – to jeden z najważniejszych elementów PhysX, reprezentuje obiekt podlegający symulacji.
PxRigidStatic	Klasa aktora reprezentującego nieruchomą (statyczną) bryłę sztywną o nieskończonej masie.
PxRigidDynamic	Klasa aktora reprezentującego ruchomą (dynamiczną) bryłę sztywną.
PxGeometry	Opis kształtu opisuje kształt aktora i obiektu. Jest to klasa bazowa dla klas definiujących pewne podstawowe kształty, jak np. PxBoxGeometry, PxPlaneGeometry.
PxTransform	Klasa reprezentująca transformacje aktora (przesunięcie i rotację), na podstawie której dokonuje się transformacji obiektów na scenie 3D.
PxMaterial	Materiał – reprezentuje właściwości powierzchni aktora, takie jak sprężystość, tarcie statyczne i dynamiczne, itp.

Konfiguracja projektu Visual C++


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Visual C++
CLR
General
Test
Win32
Visual Studio Solutions
Online
Sort by: Default
Win32 Console Application Visual C++
Win32 Project Visual C++
Empty Project Visual C++
Makefile Project Visual C++
Search Installed Templates (Ctr
Type: Visual C++
A project for creating a Win32 console application
Name: physx_test
Location: D:\work\
Browse...
Solution: Create new solution
Solution name: physx_test
Create directory for solution
Add to source control
OK
Cancel


physxtest Property Pages
Configuration: All Configurations
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
Additional Include Directories
C:\Dev\glut-3.7.6-bin;C:\Dev\PhysX3\Include
Additional #using Directories
Debug Information Format
Additional Include Directories
C:\Dev\glut-3.7.6-bin
C:\Dev\PhysX3\Include
Inherited values:
Inherit from parent or project defaults
Macros>>
OK
Cancel
(/I[path])
OK
Anuluj
Zastosuj


physxtest Property Pages
Configuration: Active(Release)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
Output File
$(OutDir)$(TargetName)$(TargetExt)
Show Progress
Not Set
Version
Enable I
Suppres
Ignore I
Register
Per-user
Addition
Link Libr
Use Libra
Link Stat
Prevent
Treat Lin
Force Fil
Create H
Specify
win32
Output File
The /OUT option overrides the default name and location of the program that the linker creates.
OK
Anuluj
Zastosuj
Additional Library Directories
C:\Dev\glut-3.7.6-bin
C:\Dev\PhysX3\Lib\vc11win32
Inherited values:
Inherit from parent or project defaults
Macros>>
OK
Cancel

Szablon programu korzystającego z PhysX

```
// Główny nagłówek biblioteki PhysX:
#include <PxPhysicsAPI.h>

// Biblioteki (tylko Visual C++):
#pragma comment(lib, "PhysX3_x86.lib")
#pragma comment(lib, "PhysX3Common_x86.lib")
#pragma comment(lib, "PhysX3Extensions.lib")
#pragma comment(lib, "PhysXProfileSDKCHECKED.lib")

// Przestrzeń nazw PhysX:
using namespace physx;
```

Zmienne globalne

```
// Globalne zmienne:
PxFoundation *pfund;        // Fundament
PxPhysics* pphys;           // PhysX SDK
PxScene* gsc;               // Opis sceny
PxVec3 gg(0, -10, 0);       // Wektor grawitacji (m/s2)
```

```
// Domyślna obsługa błędów:
PxDefaultErrorCallback gDefaultErrorCallback;
// Domyślna obsługa alokacji pamięci:
PxDefaultAllocator gDefaultAllocatorCallback;
```

Inicjalizacja PhysX SDK

```
// Obiekt klasy PxFoundation:
pfund = PxCreateFoundation(PX_PHYSICS_VERSION,
              gDefaultAllocatorCallback, gDefaultErrorCallback);
if(!pfund) {
    puts("BŁĄD!");
    exit(0);
}

// Tworzenie SDK, domyślne ustawienia materiałów:
pphys = PxCreatePhysics(PX_PHYSICS_VERSION, *pfund,
                                              PxTolerancesScale());
if(!pphys) {
    puts("BŁĄD!");
    exit(0);
}
```

Utworzenie sceny

```
// Utworzenie "sceny" z domyślnymi parametrami skali:
PxSceneDesc sceneDesc( pphys->getTolerancesScale() );
sceneDesc.gravity = gg;

// Jeden wątek domyślnego ekspedytora (dispatcher):
PxDefaultCpuDispatcher *cpudisp = PxDefaultCpuDispatcherCreate(1);
if(!cpudisp) {
    puts("BŁĄD!");
    exit(0);
}
sceneDesc.cpuDispatcher    = cpudisp;

// Domyślny filtr symulujący działanie biblioteki z wersji 2.8.x:
sceneDesc.filterShader    = &PxDefaultSimulationFilterShader;

gsc = pphys->createScene(sceneDesc);
if(!gsc) {
    puts("BŁĄD!");
    exit(0);
}
```

Parametry materiałów

Klasą odpowiedzialną za parametry materiałów jest **PxMaterial**, który posiada trzy główne parametry – ustawiane z pomocą konstruktora lub metod:

Metoda	Opis
setDynamicFriction(x)	Tarcie dynamiczne, wartość w zakresie 0..∞
setStaticFriction(x)	Tarcie statyczne, wartość w zakresie 0..∞
setRestitution(x)	Sprężystość (w zderzeniach z innymi aktorami), wartość w zakresie 0..1

```
// static friction, dynamic friction, restitution
PxMaterial *mat = pphys->createMaterial(0.5f, 0.5f, 0.5f);
```

Inicjalizacja aktorów i symulacji

```
// Jeden krok silnika fizycznego (domyślnie 1/60):
#define SIM_TIME   1/30.0

// ...

// Utworzenie aktorów:
a_plane = make_plane(mat);
a_box = make_box(mat);

// Zainicjowanie pierwszej klatki animacji (symulacji):
gsc->simulate(SIM_TIME);
```

Zakończenie programu

```
// Zamknięcie i sprzątnięcie biblioteki PhysX:
void kill_physx() {

   if(pphys != NULL) {
      if(gsc != NULL)      gsc->release();
      pphys->release();
   }

   if(pfund != NULL) pfund->release();
}
```

Aktory

Rodzaje aktorów

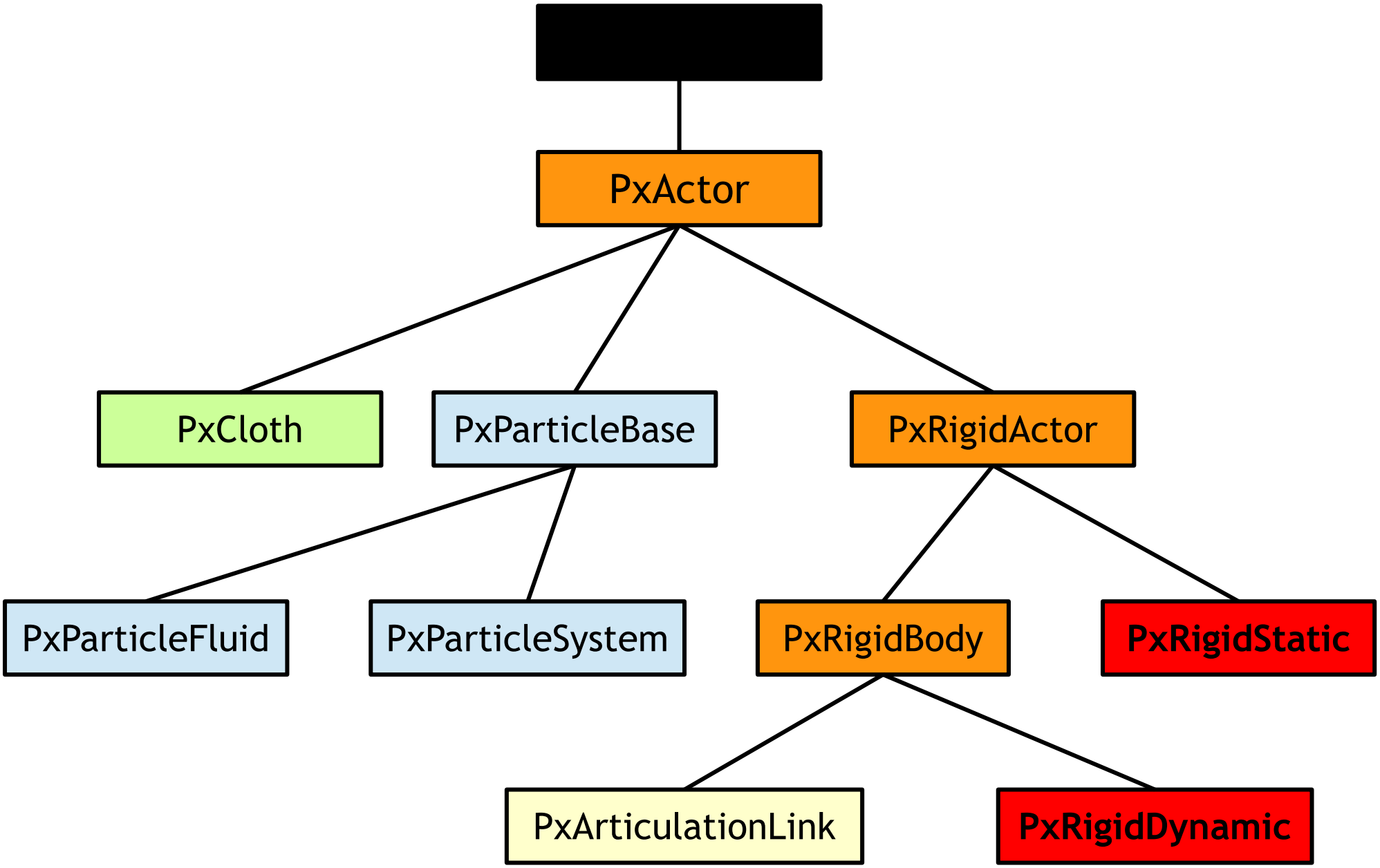
Trzy główne typy aktorów w PhysX to:

- **Statyczne** – obiekty nieruchome, ale mimo tego podlegające prawom fizyki, mają nieskończenie dużą masę.
- **Dynamiczne** – ruchome ciała sztywne. Mogą być reprezentowane przez abstrakcyjne punkty posiadające masę, bez definicji kształtu.
- **Kinematyczne** – obiekty o nieskończenie dużej masie, ale mimo tego mogą być przemieszczane i oddziaływać na inne aktory w scenie.

Podstawowym wymogiem poprawnie zdefiniowanej sceny jest to, aby kształty aktorów wzajemnie się **nie przecinały**.

Oprócz symulacji fizyki, PhysX daje możliwość **detekcji kolizji** i reakcji programu na takie zdarzenia.

Klasy reprezentujące aktory



Wybrane metody klas aktorów

```
void  setGlobalPose (const PxTransform &pose);
PxTransform  getGlobalPose();
```

Ustawienie i odczyt bieżącej pozycji i rotacji aktora we współrzędnych świata.

```
void setActorFlags (PxActorFlags  inFlags);
void setActorFlag (PxActorFlag::Enum flag, bool value);
PxActorFlags getActorFlags ();
```

Odczyt i ustawienie flag aktora które pozwalają m.in. wyłączyć wpływ grawitacji na aktora i wyłączyć danego aktora z symulacji fizyki.

Definicja kształtu aktora

Najważniejszy atrybut każdego aktora to jego kształt, który opisany jest z pomocą klasy **PxShape**.

Pojedynczy aktor może posiadać wiele kształtów tworzących bardziej skomplikowane modele. Kształty dołącza się do aktora wywołując metodę **createShape()** aktora.

Obiekt klasy **PxShape** powstaje na bazie obiektów klas geometrii (**Px***Geometry**).

Każdy z kształtów składających się na aktora może mieć swoją lokalną pozycję względem środka obiektu (local pose).

Dostępne kształty aktorów

Kształty podstawowe:

Nazwa klasy	Kształt
PxBoxGeometry	Prostopadłościan
PxCapsuleGeometry	Kapsuła – cylinder z półkulami na końcach
PxPlaneGeometry	Nieskończona płaszczyzna
PxSphereGeometry	Sfera (kula)

Kształty złożone:

Nazwa klasy	Kształt
PxConvexMeshGeometry	Wielościan wypukły opisany wierzchołkami, co najwyżej 256 ścian
PxHeightFieldGeometry	Pole wysokości, czyli nierówna powierzchnia opisana przez wysokości punktów
PxTriangleMeshGeometry	Bryła dowolnego kształtu opisana przez wierzchołki i ściany

Utworzenie aktora

Ostatni etap to utworzenie aktora z pomocą metody obiektu sceny:

```
void addActor (PxActor &actor)
```

Przykład – statyczna podłoga

```
PxRigidStatic* make_plane(PxMaterial *mat) {

    // Pozioma podłoga:
    PxTransform pose = PxTransform(PxVec3(0, 0, 0),
                                   PxQuat(PxHalfPi, PxVec3(0, 0, 1)));

    // Utworzenie aktora statycznego:
    PxRigidStatic *plane = pphys->createRigidStatic(pose);

    // Utworzenie kształtu i przypisanie do aktora:
    PxShape* shape = plane->createShape(PxPlaneGeometry(), *mat);

    // Utworzenie aktora i dołączenie do sceny:
    gsc->addActor(*plane);
    return(plane);
}
```

Przykład – statyczne pudełko

```
PxRigidStatic* make_static_box(PxMaterial *mat, PxVec3 &pos,
                                                PxVec3 &dim) {
    // Położenie pudełka:
    PxTransform transform(pos);

    // Utworzenie STATYCZNEGO aktora:
    PxRigidStatic *boxie = pphys->createRigidStatic(transform);

    // Kształt pudełka o podanych rozmiarach:
    PxShape* shape = boxie->createShape(PxBoxGeometry(dim), *mat);

    gsc->addActor(*boxie);
    return(boxie);
}
```

Aktory dynamiczne

Tworzenie aktorów dynamicznych

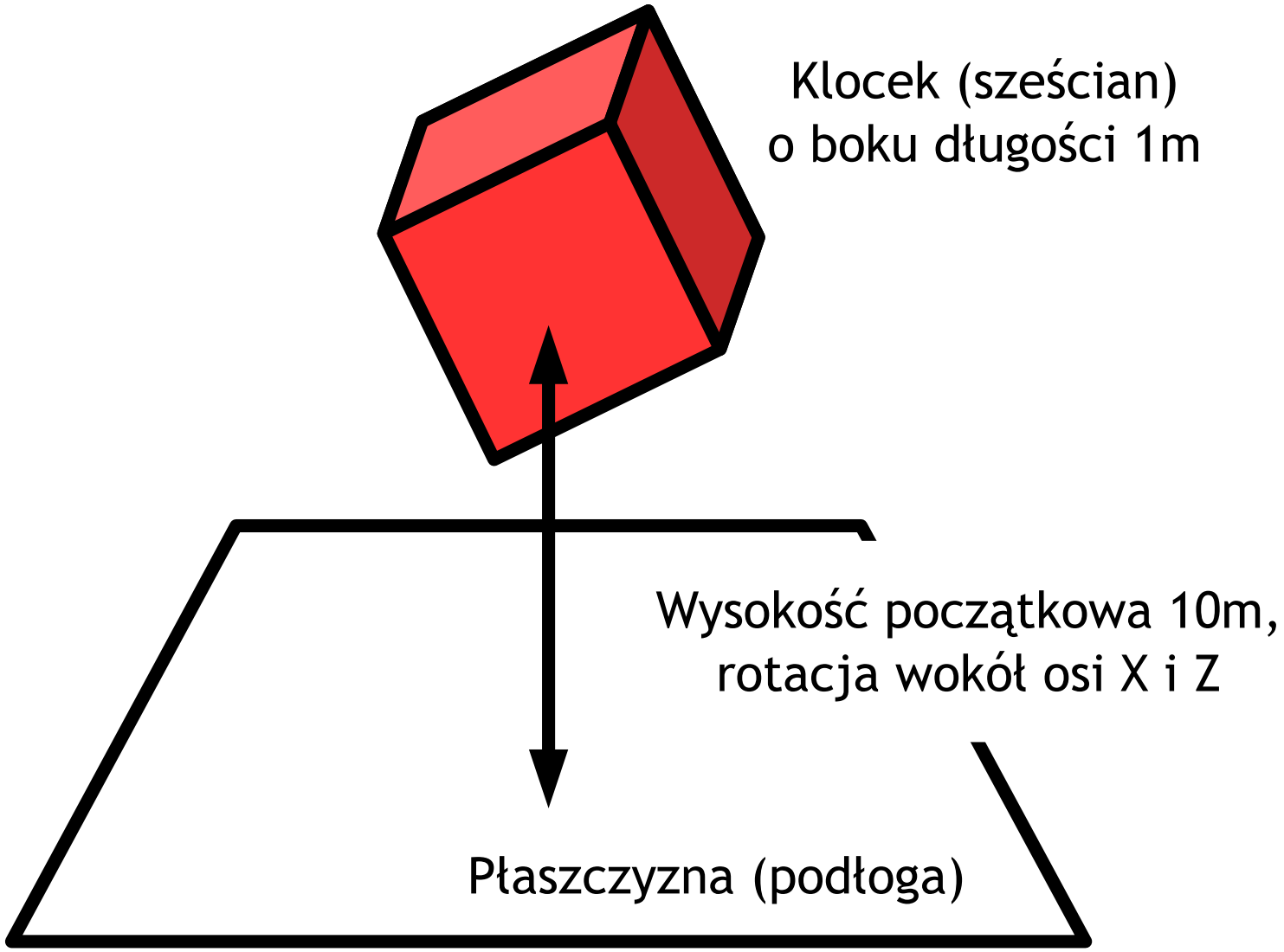
```
PxRigidDynamic* PxCreateDynamic(PxPhysics &sdk,
    const PxTransform &transform, const PxGeometry &geometry,
    PxMaterial &material, PxReal density)
```

- **sdk** – referencja do obiektu SDK;
- **transform** – transformacja (rotacja i przesunięcie) początkowa aktora;
- **geometry** – kształt bryły;
- **material** – materiał którym „pokryty” jest aktor;
- **density** – gęstość materiału z którego zrobiony jest aktor (jednostka masy na jednostkę objętości).

Aktor dynamiczny – przykład (1)

```
PxRigidDynamic* make_box(PxMaterial *mat) {
    PxReal density = 1.0f;

    // Rotacja w przestrzeni (żeby klocek nie spadał płasko):
    PxQuat kwa(3.1415/3, PxVec3(1, 0, 0));
    // 10 jednostek nad powierzchnią:
    PxTransform transform(PxVec3(0.0f, 10.0f, 0.0f), kwa);

    // Rozmiary i kształt klocka:
    PxVec3 dimensions(0.5, 0.5, 0.5);
    PxBoxGeometry geometry(dimensions);

    // Utworzenie dynamicznego aktora:
    PxRigidDynamic *actor = PxCreateDynamic(*pphys, transform, geometry,
                                                        *mat, density);
    actor->setAngularDamping(0.75);

    gsc->addActor(*actor);
    return(actor);
}
```

Przykładowy program (klocek3)



Aktor dynamiczny – przykład (2)

```
PxRigidDynamic* make_sfera(PxMaterial *mat, float radius) {
    // Położenie sfery:
    PxTransform transform(PxVec3(0, 12, 0));

    // Kształt sfery o podanym promieniu:
    PxSphereGeometry geometry(radius);

    // Utworzenie dynamicznego aktora:
    PxRigidDynamic *actor = PxCreateDynamic(*pphys,
                                transform, geometry, *mat, 20.0f);

    gsc->addActor(*actor);
    return(actor);
}
```

Metody klasy PxRigidBody

`void setCMassLocalPose (const PxTransform &pose)`

Ustawia położenie środka ciężkości względem środka aktora.

`void setMass (PxReal mass)`

Bezpośrednie ustawienie masy aktora.

`void setLinearVelocity (const PxVec3 &linVel)`

Ustawienie (wymuszenie) prędkości liniowej.

`void setAngularVelocity (const PxVec3 &angVel)`

Ustawienie (wymuszenie) prędkości kątowej.

`addForce (…), addTorque (…)`

Przyłożenie siły i momentu obrotowego do aktora.

Animacja

Animacja

Kolejne kroki symulacji fizycznej sterowane są metodą **simulate()** klasy **PxScene**. Metoda jako argument przyjmuje **czas**, który upłynął od poprzedniego wywołania metody **simulate()**.

Czas przekazany do metody simulate() może być wyznaczony na (co najmniej) dwa sposoby:

- **Wartość zmierzona** – czas pomiędzy kolejnymi klatkami animacji zmierzony z pomocą funkcji których dostarcza system operacyjny (np. *gettimeofday*, *getTickCount*).
- **Wartość stała** – czas wynika z założonej prędkości animacji (ramek na sekundę). Jeżeli animacja będzie się „zacinać”, to razem z nią będzie zwalniać symulacja fizyczna.

Rozpoczęcie symulacji

Pierwsze wywołanie metody **simulate()** rozpoczyna symulację fizyczną. Należy to zrobić przed uruchomieniem głównej pętli programu (przed wywołaniem funkcji **glutMainLoop()** w przypadku biblioteki GLUT).

```
// Jeden krok silnika fizycznego:
#define SIM_TIME    1/30.0

// ...

PxScene* gsc;

void init_physx() {
    // ...

    // Zainicjowanie pierwszej klatki animacji (symulacji):
    gsc->simulate(SIM_TIME);
}
```

Rezultaty symulacji

Na początku funkcji odpowiadającej za rysowanie sceny, należy pobrać aktualny stan symulacji fizycznej, z pomocą metody **fetchResults()** obiektu sceny:

```
gsc->fetchResults(true);
```

Kolejny krok symulacji

UWAGA! Wywołania metod **simulate()** i **fetchResults()** zawsze powinny występować naprzemiennie.

Po odczytaniu rezultatów symulacji należy zainicjować jej kolejny krok (np. na końcu funkcji rysującej):

```
gsc->simulate(SIM_TIME);
```

Kolejny krok symulacji

Aktualny stan danego aktora (obrót i położenie), odczytać można z pomocą metody **getGlobalPose()** klasy aktora:

```
PxRigidDynamic* a_box;

// ...

// Odczytanie aktualnej pozycji klocka (PhysX):
PxTransform pose = a_box->getGlobalPose();
```

Przeniesienie transformacji do OpenGL

Procedura na podstawie podanego obiektu klasy **PxTransform** konstruuje macierz przekształcenia OpenGL i ustawia ją jako aktywną. Po jej wywołaniu należy narysować odpowiedni obiekt z pomocą funkcji OpenGL.

```
// Wygenerowanie maierzy przekształceń dla OpenGL:
void SetupGLMatrix(const PxTransform &pose) {
    float glmat[16];// Macierz 4x4 dla OpenGL

    // Pobranie macierzy przekształceń i wektora translacji:
    PxMat33 m = PxMat33(pose.q);

    getColumnMajor(m, pose.p, glmat);

    // Dołączenie aktualnego przekształcenia:
    glMultMatrixf(&(glmat[0]));
}
```

Zbudowanie macierzy OpenGL

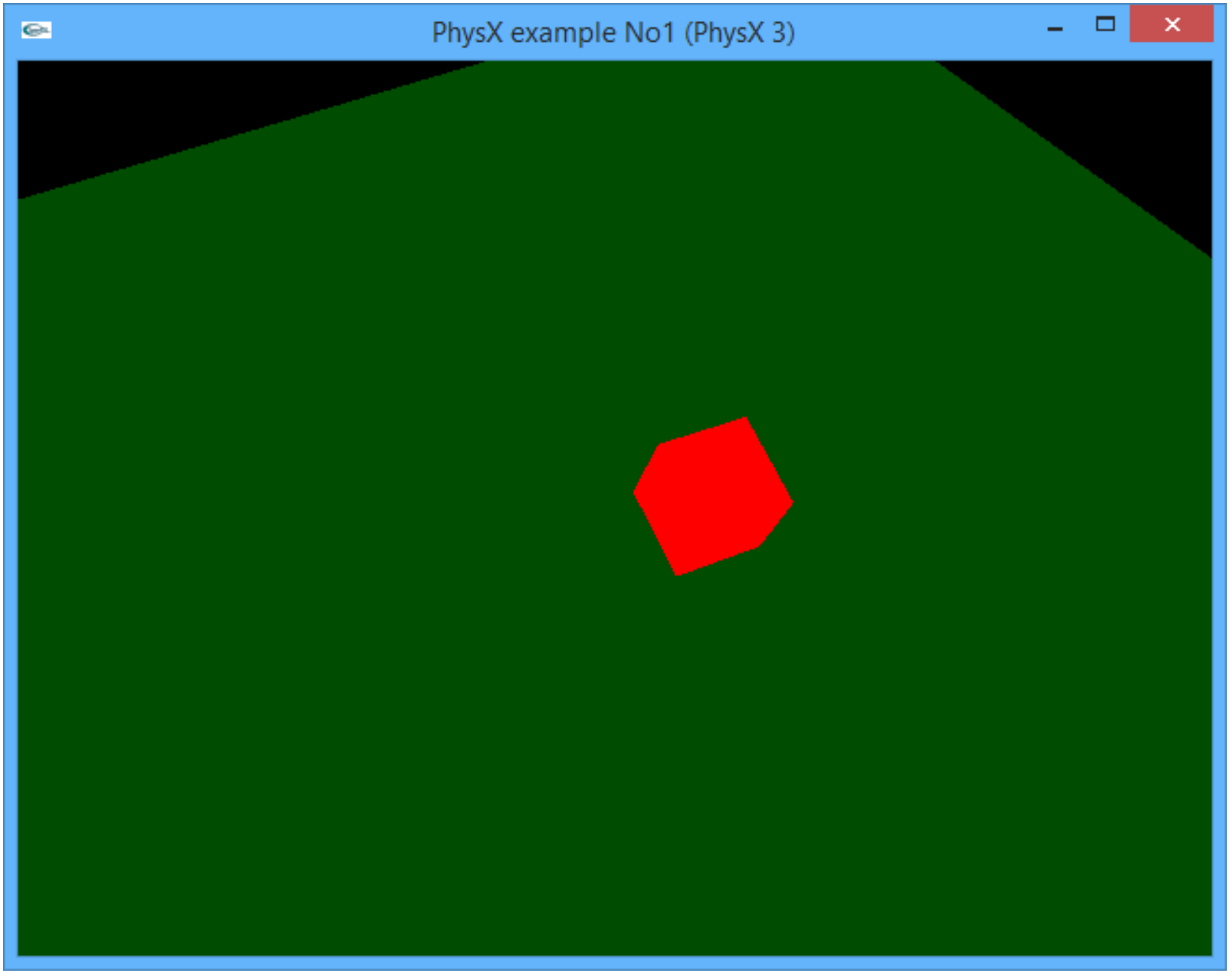
```
void getColumnMajor(PxMat33 m, PxVec3 t, float* mat) {
    mat[0] = m.column0[0];
    mat[1] = m.column0[1];
    mat[2] = m.column0[2];
    mat[3] = 0;

    mat[4] = m.column1[0];
    mat[5] = m.column1[1];
    mat[6] = m.column1[2];
    mat[7] = 0;

    mat[8] = m.column2[0];
    mat[9] = m.column2[1];
    mat[10] = m.column2[2];
    mat[11] = 0;

    mat[12] = t[0];
    mat[13] = t[1];
    mat[14] = t[2];
    mat[15] = 1;
}
```

Przykładowa funkcja rysująca (1/2)

```
void RenderScene() {
    // Wyczyszczenie okna i buforów (OpenGL):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    // Pobranie wyników symulacji (PhysX):
    gsc->fetchResults(true);

    // RENDERING (OpenGL):
    glTranslatef(0, 0, -15);      // Odsunięcie środka sceny
    glRotatef(50, 1, 0, 0);       // Widok z góry
    glRotatef(30, 0, 1, 0);       // Trochę z boku

    // Podłoga przechodząca przez punkt (0, 0, 0) (OpenGL):
    glColor3f(0, 0.3, 0);
    glBegin(GL_QUADS);
        glVertex3f(-10, 0, -10);
        glVertex3f(-10, 0,  10);
        glVertex3f( 10, 0,  10);
        glVertex3f( 10, 0, -10);
    glEnd();
```

Przykładowa funkcja rysująca (2/2)

```
    // Odczytanie aktualnej pozycji klocka (PhysX):
    PxTransform pose = a_box->getGlobalPose();
    glPushMatrix();
        // Odczyt aktualnej pozycji klocka:
        SetupGLMatrix(pose);

        // Klocek na ekran (OpenGL):
        glColor3f(1, 0, 0);
        glutSolidCube(1);
    glPopMatrix();

    // Wyliczenie kolejnej klatka symulacji (PhysX):
    gsc->simulate(SIM_TIME);
    // Zmiana buforów (OpenGL):
    glutSwapBuffers();
}
```


PhysX example No1 (PhysX 3)