Fizyka i silniki fizyczne
nVidia PhysX

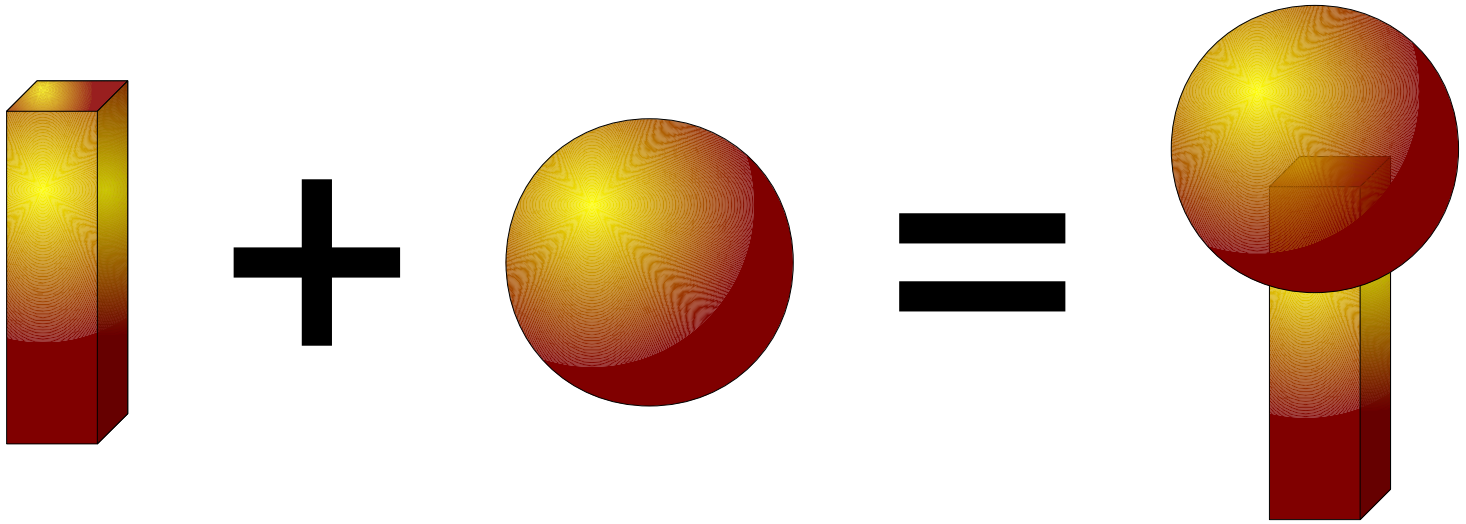
Część 2: Kształty aktorów

Podstawowe kształty aktorów

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

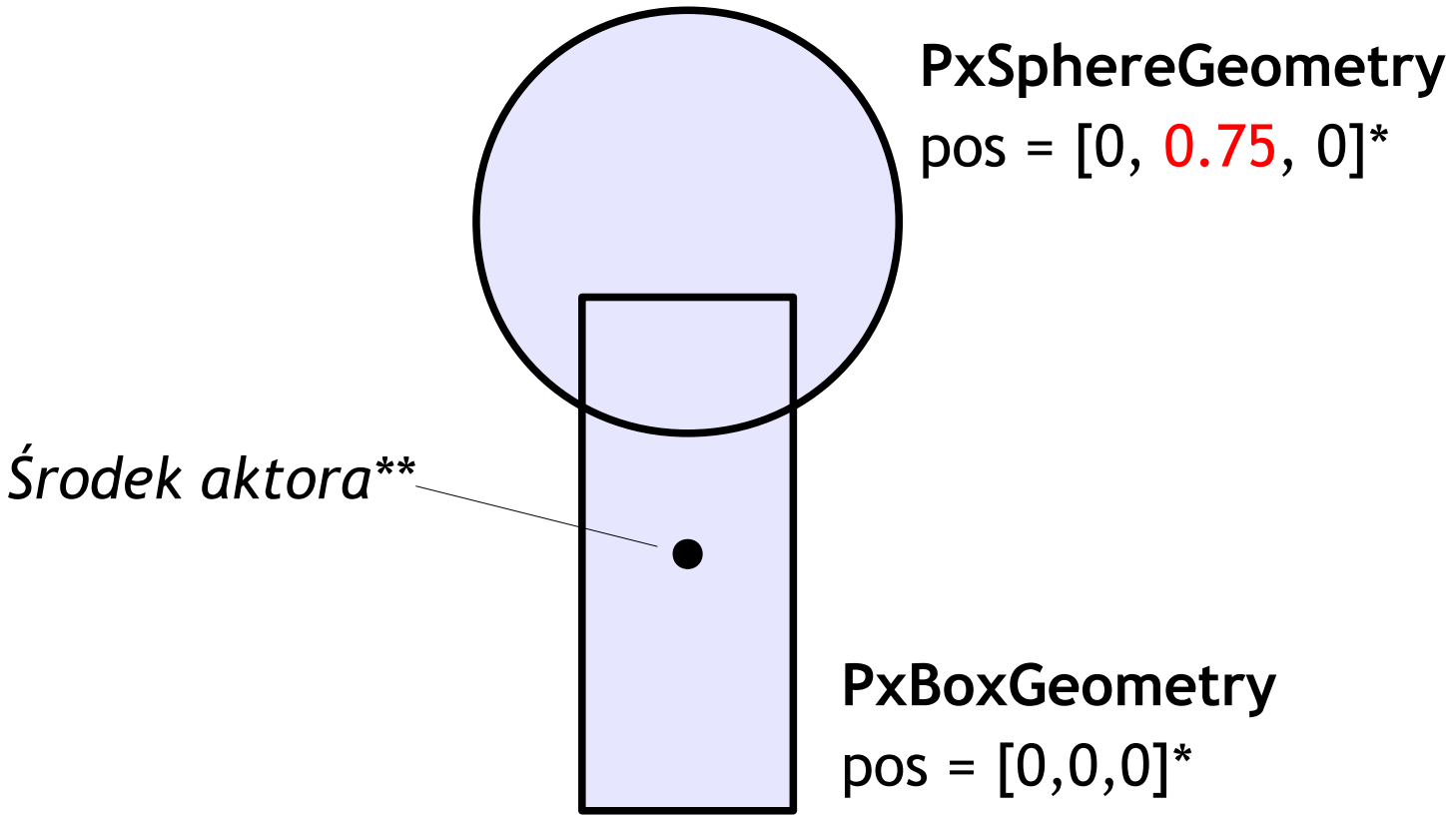
Aktory złożone z wielu prymitywów

Podstawowe kształty można łączyć dodając wielokrotnie prymitywy do jednego aktora.



Wzajemne położenie elementów składowych określone jest przez lokalną pozycję każdego kształtu składowego. Kolejne kształty dodaje się wywołując metodę **createShape()** aktora.

Przykład – kręgle



** Pokazano tylko wektor translacji.*

*** Środek ciężkości to środek masy utworzonej przez kształty składowe.*

Klasa PxRigidBodyExt

Pomocnicza klasa **PxRigidBodyExt** zawiera funkcje pomocnicze związane z klasą **PxRigidBody** i jej pochodnymi. Funkcja **updateMassAndInertia()** z tej klasy wyznacza parametry masy aktora (m.in. środek ciężkości).

```
static bool PxRigidBodyExt::updateMassAndInertia(
           PxRigidBody &body, PxReal density,
           const PxVec3 *massLocalPose = NULL,
           bool includeNonSimShapes = false )
```

- `body` – referencja do aktora dynamicznego/kinematycznego;
- `density` – gęstość aktora;
- `massLocalPose` – położenie środka masy względem środka aktora (opcjonalnie);
- `includeNonSimShapes` – TRUE jeżeli do wyznaczania masy należy wziąć pod uwagę kształty nie podlegające symulacji (opcjonalnie).

Przykład – utworzenie aktora

```
PxRigidDynamic* make_bowl(PxMaterial *mat, float x, float y, float z) {
    // Pozycja kręgla we współrzędnych świata (globalnych):
    PxTransform pose(PxVec3(x, y, z));
    // Podstawą jest prostopadłościan:
    PxBoxGeometry gbox(0.2, 0.8, 0.2);

    // Utworzenie aktora bazowego złożonego z prostopadłościanu:
    PxRigidDynamic *actor = PxCreateDynamic(*pphys, pose, gbox, *mat, 10.0f);

    // U góry "kręgla" jest kula:
    PxSphereGeometry gsph(0.4);
    // Współrzędne kuli we współrzędnych lokalnych aktora:
    PxTransform locpose(PxVec3(0, 0.75, 0));
    // Dołączenie kuli do kształtu aktora:
    PxShape *kula = actor->createShape(gsph, *mat, locpose);

    // Przeliczenie masy:
    PxRigidBodyExt::updateMassAndInertia(*actor, 10.0f);

    // Dodanie do sceny i koniec:
    gsc->addActor(*actor);
    return(actor);
}
```

Przykład – utworzenie aktorów zestawu kręgli

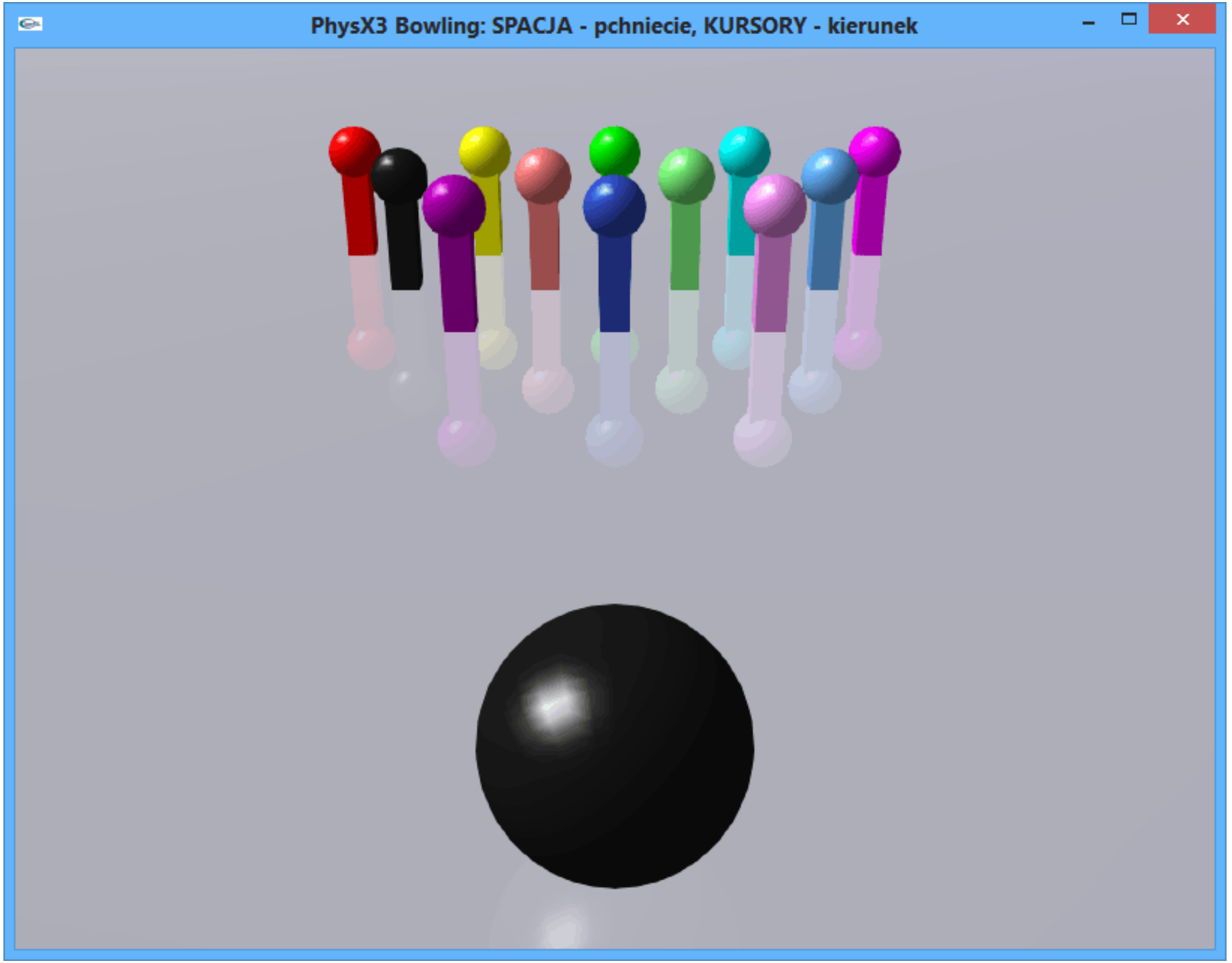
```
// Parametry kręgli:
#define NBOWLS  12
PxRigidDynamic* a_bowl[NBOWLS];

// Kolory predefiniowane:
float bowl_col[NBOWLS][3] = { /* ... */ };
// Pozycje początkowe:
float bowl_pos[NBOWLS][3] = { /* ... */ };

// Globalny materiał:
PxMaterial *glo_mat;

void init_physx() {
    // ...
    // Stojące kręgle:
    for(int i=0;i < NBOWLS;i++)
        a_bowl[i] = make_bowl(glo_mat,
                bowl_pos[i][0], bowl_pos[i][1], bowl_pos[i][2]);
    // ...
}
```

Przykład – narysowanie kręgla w OpenGL

```
// Narysowanie odpowiedniego obiektu pasującego do aktora:
void draw_bowl() {

    // Prostopadłościan:
    glPushMatrix();
        glScalef(1, 4, 1);
        glutSolidCube(0.4);
    glPopMatrix();

    // Sfera z przesunięciem w górę:
    glPushMatrix();
        glTranslatef(0, 0.75, 0);
        glutSolidSphere(0.4, 16, 16);
    glPopMatrix();
}
```

Przykład – narysowanie zestawu kręgli w OpenGL

```
void render_stuff() {

    // ...


    // Kręgle:
    for(int i=0;i < NBOWLS;i++) {
        PxTransform pose = a_bowl[i]->getGlobalPose();

        glPushMatrix();
            // Pozycja do OpenGL:
            SetupGLMatrix(pose);
            // Kręgiel na ekran:
            glColor3fv(bowl_col[i]);
            draw_bowl();
        glPopMatrix();
    }
}
```

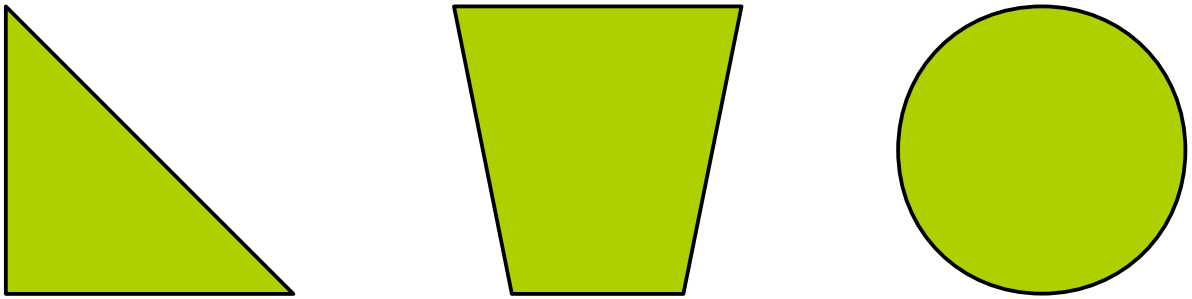

PhysX3 Bowling: SPACJA - pchniecie, KURSORY - kierunek

Bryły wypukłe

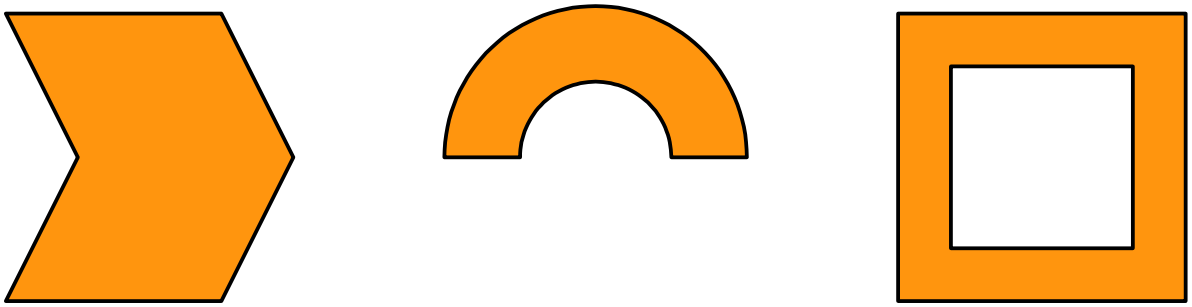
Bryły wypukłe (ang. *convex block*)

Zbiór wypukły – podzbiór przestrzeni euklidesowej o tej własności, że dowolny odcinek którego końce należą do tego zbioru w całości się w nim zawiera.

Zbiory wypukłe:



Zbiory niewypukłe:



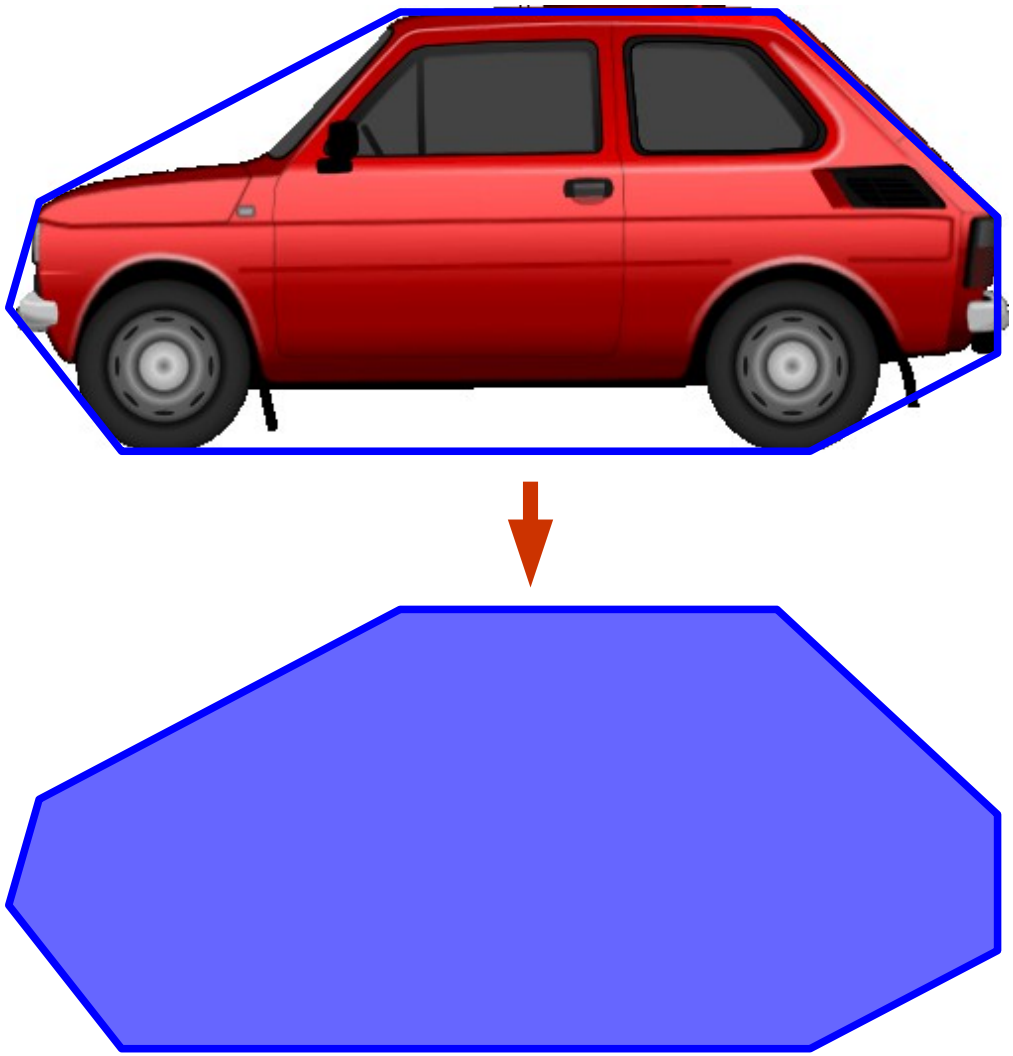
PxConvexMeshGeometry

Aktor typu **convex** opisany jest serią punktów które muszą spełniać następujące warunki:

- muszą być wierzchołkami wielościanu wypukłego,
- nie mogą się powtarzać,
- wielokątów pokrywających powierzchnię obiektu (aktora) nie może być więcej niż 256.

Aktory wypukłe

Aktor typu **convex**, po uproszczeniu siatki, może reprezentować bardziej złożone kształty modeli, np.:



Model 3D
zbiór niewypukły

Obrys modelu 3D
zbiór wypukły

PxCooking

Aktor opisany przez obiekt klasy **PxConvexMeshGeometry**, przed użyciem musi zostać „ugotowany” (ang. *cooked*).

„Gotowanie” (ang. *cooking*) to proces wstępnego przetwarzania siatek opisujących kształty typu **convex**, co pozwala na efektywne wykrywanie kolizji przez PhysX i wielokrotne użycie przetworzonej siatki do tworzenia wielu aktorów.

Mechanizmy „gotowania” umieszczone są w klasie **PxCooking**.

PxCooking.h

```
#include <PxCooking.h>

// W przypadku Visual C++:
#pragma comment(lib, "PhysX3Cooking_x86.lib")

// Zmienna globalna:
PxCooking *pcook;

// ...

// Inicjalizacja "piekarnika":
pcook = PxCreateCooking(PX_PHYSICS_VERSION, *pfund,
            PxCookingParams(pphys->getTolerancesScale()));

if (!pcook) {
   puts("PxCreateCooking ERROR!");
   exit(0);
}
```

Przygotowanie kształtu typu convex

Procedura „gotowania” siatki składa się z następujących kroków:

1) Przygotowanie i wypełnienie danymi obiektu klasy **PxConvexMeshDesc**.

2) Przetworzenie siatki z pomocą metody **cookConvexMesh()** obiektu **PxCooking**, umieszczenie wyników w buforze.

3) Utworzenie obiektu klasy **PxConvexMesh**, z pomocą metody **createConvexMesh()**.

4) Utworzenie aktora korzystając z przygotowanego kształtu i klasy **PxConvexMeshGeometry**.

Przygotowanie kształtu (1/2)

```
// Kształt „cukierka”:
PxConvexMesh* boxoMesh;

PxConvexMesh* init_boxo() {
    // Kształt opisany chmurą wierzchołków bez powtórzeń:
    int npoints = ?;
    float *pts = ?;

    // Dane kształtu:
    PxConvexMeshDesc conMeshDesc;
    // Liczba punktów:
    conMeshDesc.points.count = npoints;
    // Liczba bajtów na definicję jednego punktu:
    conMeshDesc.points.stride = sizeof(float)*3;
    // Wskaźnik na dane (tylko punkty):
    conMeshDesc.points.data = pts;
    // Flaga nakazująca przeliczenie kształtu:
    conMeshDesc.flags = PxConvexFlag::eCOMPUTE_CONVEX;
```

Przygotowanie kształtu (2/2)

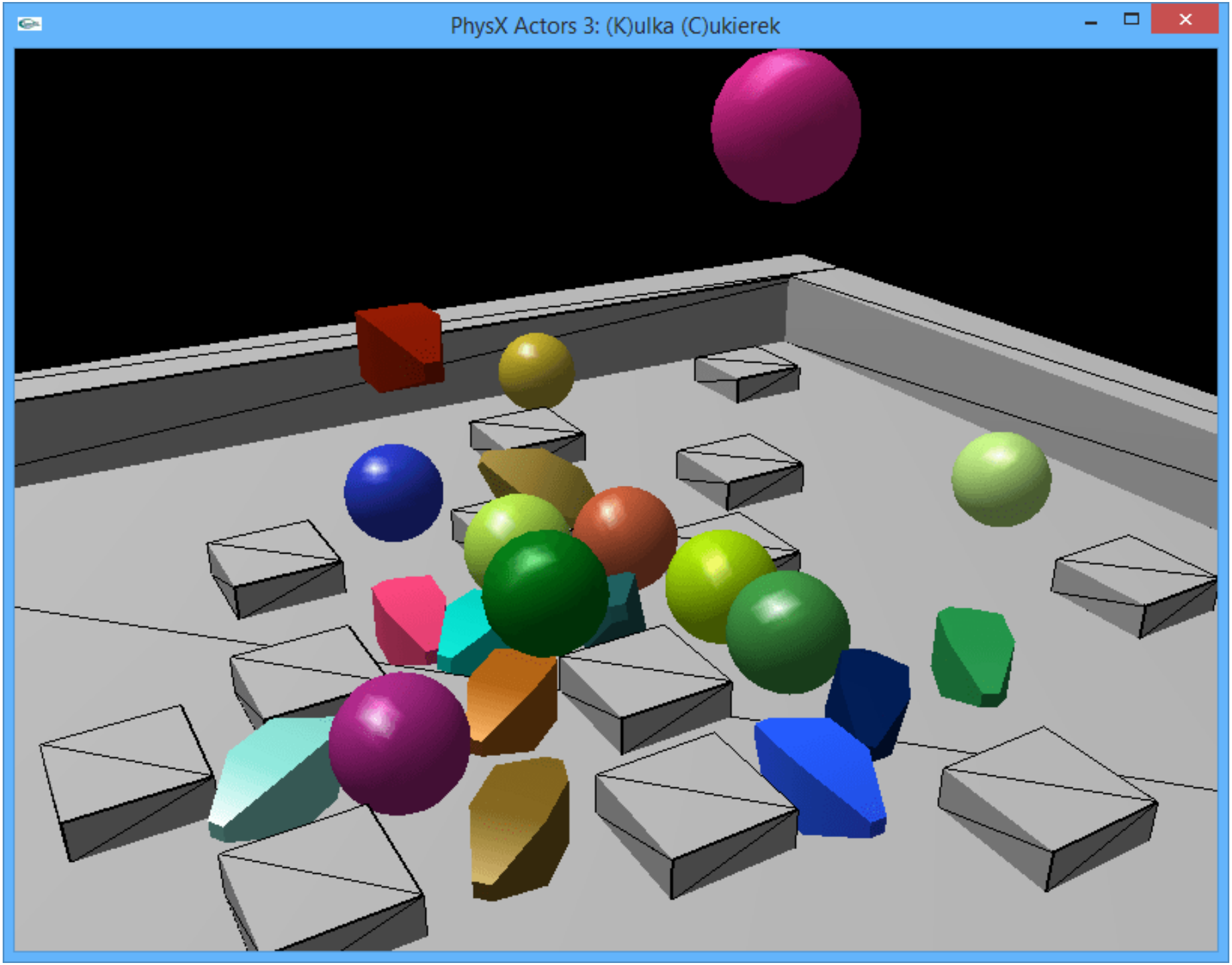
```
    // Pieczenie kształtu:
    PxDefaultMemoryOutputStream writeBuffer;

    bool status = pcook->cookConvexMesh(conMeshDesc, writeBuffer);
    if(!status) {
        puts("cookTriangleMesh() ERROR!");
        exit(0);
    }

    // Zbudowanie siatki kształtu:
    PxDefaultMemoryInputData readBuffer(writeBuffer.getData(),
                                        writeBuffer. getSize());

    // Wygenerowanie obiektu typu PxConvexMesh:
    return pphys->createConvexMesh(readBuffer);
}
```

Utworzenie aktora z użyciem gotowego kształtu

```
// Cukierek (convex):
PxRigidDynamic* make_boxo(PxMaterial *mat,
                                    float x, float y, float z) {
    // Pozycja startowa:
    PxTransform pose = PxTransform(PxVec3(x, y, z));

    // Utworzenie z "upieczonego" kształtu...
    PxRigidDynamic *cukierek = PxCreateDynamic(*pphys, pose,
                    PxConvexMeshGeometry(boxoMesh), *mat, 20.0f);

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

Narysowanie obiektu (Anim8or)

```
void render_stuff() {
    for(int i=0;i < nboxes;i++) {
        // Aktualne położenie:
        PxTransform pose = a_box[i]->getGlobalPose();

        // Narysowanie (OpenGL):
        glPushMatrix();
            SetupGLMatrix(pose);
            // Cukierek jako obiekt Anim8or
            glColor4f(box_col[i][0], box_col[i][1], box_col[i][2], 1);
            an8DrawObject(&object_boxo, false);
        glPopMatrix();
    }

    // Rysowanie pozostałych obiektów...
    // ...
}
```


PhysX Actors 3: (K)ulka (C)ukierek

Statyczne siatki trójkątów

Statyczne siatki trójkątów (PxTriangleMesh)

Siatki trójkątów (ang. *triangle mesh*) mogą reprezentować złożone obiekty zbudowane z wierzchołków i ścian (trójkątów). Mogą to być obiekty dowolnego kształtu, również niewypukłe, z otworami itd.

PhysX pozwala na tworzenie **tylko i wyłącznie statycznych aktorów** tego typu (z powodu m.in. dużej złożoności obliczeniowej wyznaczania kolizji siatek trójkątów).

Statyczne siatki trójkątów

Aktor typu **triangle shape** opisany jest tabelą wierzchołków i tabelą ścian (trójkątów). Tabela ścian składa się z trójek wskazujących wierzchołki (indeksy).

Tak opisany kształt musi spełniać następujące warunki:

- wierzchołki nie mogą się powtarzać,
- wektor normalny każdego trójkąta przekazywany jest niejawnie i wyznaczany jest z iloczynu wektorowego (v1-v0) x (v2-v0).

Aby detekcja kolizji działała poprawnie, wektor normalny powinien wskazywać **przednie strony trójkątów.**

Statyczna siatka trójkątów – przykład (1/3)

```
PxRigidStatic *make_placek(PxMaterial *mat) {
    // Tabela punktów (bez powtórzeń) i ich liczba:
    float pts[] = { … };
    int npoints = …;
    // Tabela ścian (trójkątów) i ich liczba:
    int indices[] = { … };
    int nindices = …;

    // Wypełnienie struktury reprezentującej siatkę trójkątów:
    PxTriangleMeshDesc meshDesc;

    meshDesc.points.count           = npoints;
    meshDesc.points.stride          = sizeof(float) * 3;
    meshDesc.points.data            = pts;

    meshDesc.triangles.count        = nindices / 3;
    meshDesc.triangles.stride       = sizeof(int) * 3;
    meshDesc.triangles.data         = indices;
```

Statyczna siatka trójkątów – przykład (2/3)

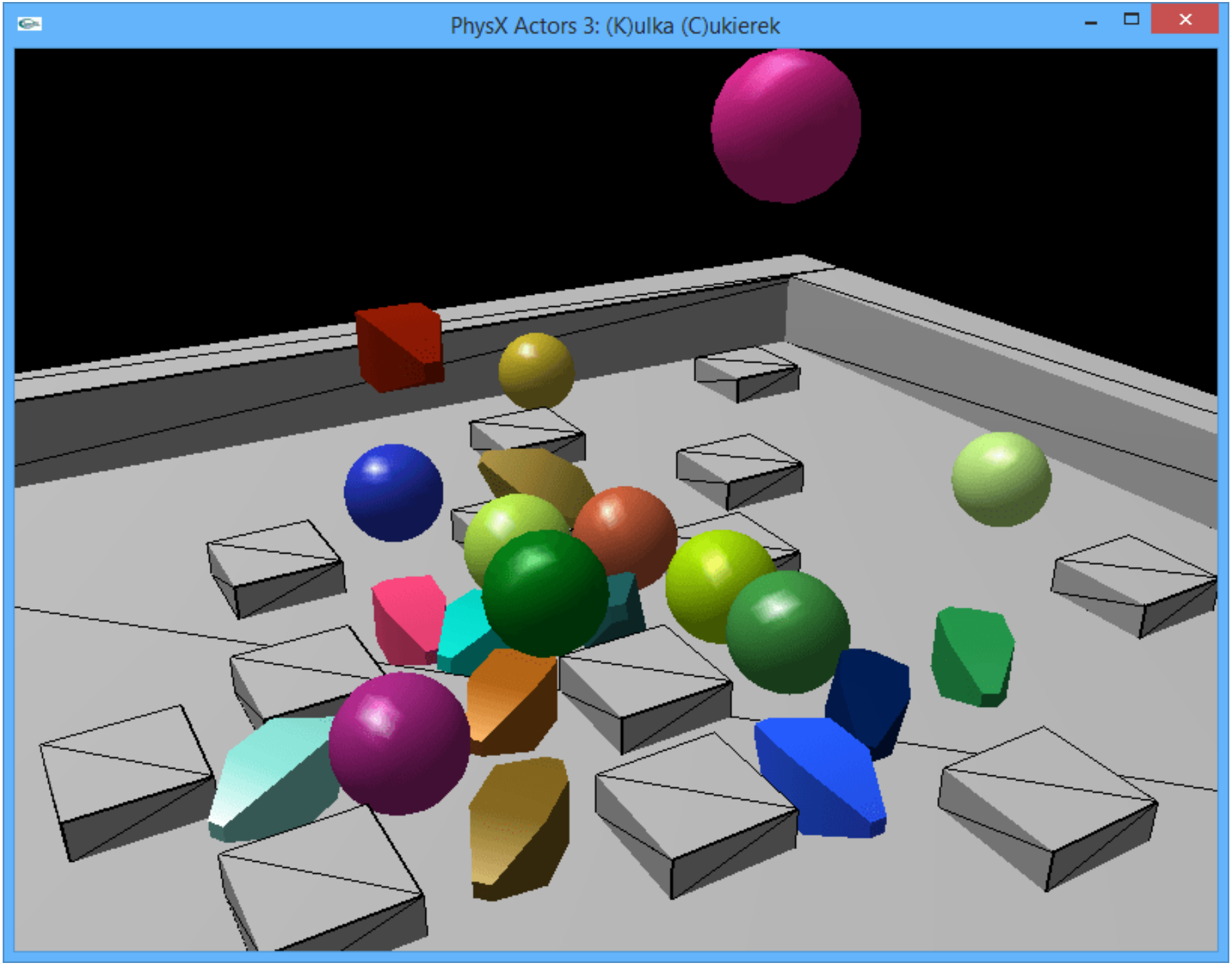
```
PxDefaultMemoryOutputStream writeBuffer;

bool status = pcook->cookTriangleMesh(meshDesc, writeBuffer);
if(!status) {
    puts("cookTriangleMesh() ERROR!");
    return(NULL);
}

PxDefaultMemoryInputData readBuffer(
                    writeBuffer.getData(), writeBuffer.getSize());

PxTriangleMesh *triMesh = pphys->createTriangleMesh(readBuffer);
```

Statyczna siatka trójkątów – przykład (3/3)

```
    // Utworzenie aktora z siatki trójkątów:
    PxTransform pose = PxTransform( PxVec3(0, 0, 0) );

    PxRigidStatic *scenka = pphys->createRigidStatic(pose);

    PxShape* triShape =
        scenka->createShape(PxTriangleMeshGeometry(triMesh), *mat);

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

Statyczna siatka trójkątów – rysowanie (OpenGL)

```
// Główna funkcja rysująca:
void RenderScene() {
    // ...

    // To zwykły obiekt nie zmieniający położenia:
    glColor3f(0, 0, 0);
    an8DrawObject(&object_placek, false, true);  // Druciak

    glColor4f(0.7f, 0.7f, 0.7f, 1.0f);
    an8DrawObject(&object_placek, false);        // Bryła

    // ...
}
```


PhysX Actors 3: (K)ulka (C)ukierek