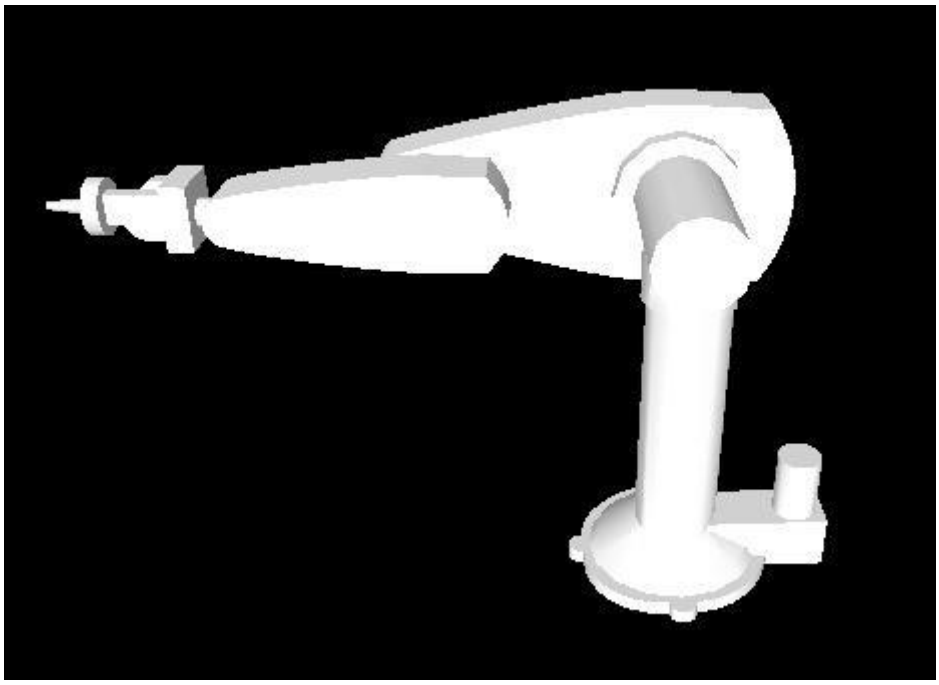
Zadanie „PUMA”

Zadanie polega na zbudowaniu i wyświetleniu przykładowej animowanej sceny przedstawiającej robota spawalniczego typu „PUMA”.

Cały projekt składa się z następujących elementów:
1. Animacja ramion robota przy pomocy rozwiązywania zadania kinematyki odwrotnej
2. Swobodna kamera o 5-ciu stopniach swobody
3. Oświetlenie
4. Bryły cienia
5. System cząstek reprezentujący iskry
6. Odbicie sceny w kawałku blachy

Do zaliczenia wymagane są co najmniej trzy pierwsze punkty. Termin oddania projektu to **08.05**. Pisać można w **C++ lub C#**, z użyciem **DirectX lub OpenGL**.

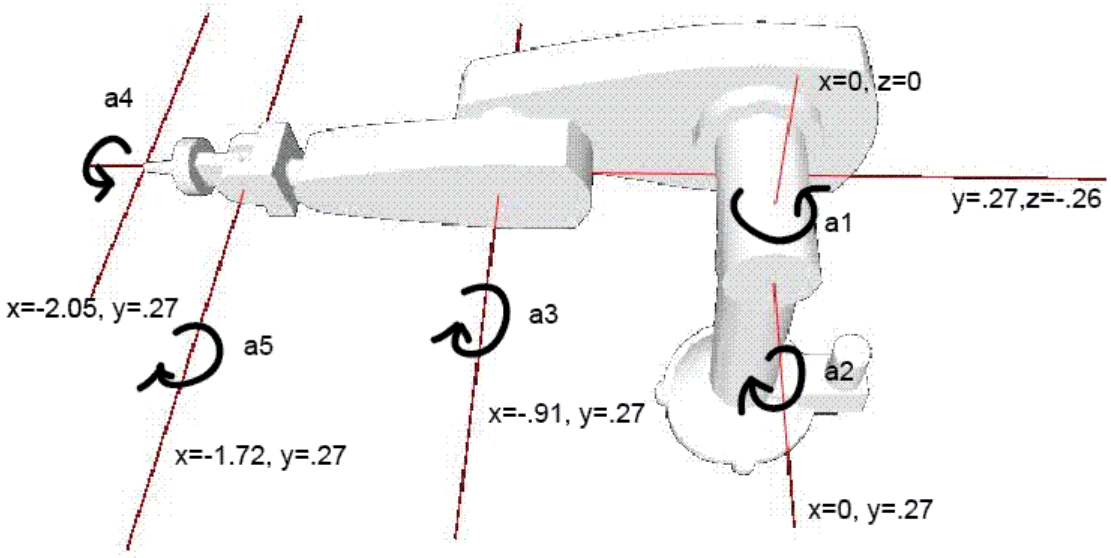
1. ***Animacja ramion robota przy pomocy rozwiązywania zadania kinematyki odwrotnej***



Do zadania dołączone jest 6 plików tekstowych z danymi siatek trójkątów 6-ciu części (ramion) robota: mesh1.txt, ..., mesh6.txt. Pierwsza część, to podstawa, każda kolejna reprezentuje kolejne ramię robota. Po wczytaniu i wyświetleniu siatek w ich lokalnych układach współrzędnych powinno się uzyskać obraz jak na zrzucie ekranu powyżej , a więc wszystkie siatki mają współrzędne wierzchołków i normalnych wyrażone w tym samym układzie współrzędnych. Środek podstawy robota znajduje się w punkcie (0,-1,0), wysokość robota nie przekracza 1,5. Na ekranie powyżej oś z wychodzi z kartki, a robot zwrócony jest w ujemnym kierunku osi x.

Format plików tekstowych:
- W pierwszej linijce pliku tekstowego z opisem siatki znajduje się liczba różnych pozycji wierzchołków „k”. W każdym z „k” kolejnych wierszy znajdują się trzy liczby rzeczywiste – współrzędne kolejnej

pozycji wierzchołka.
- W następnej linijce znajduje się liczba różnych wierzchołków „l”. Liczba różnych wierzchołków jest z reguły większa niż liczba różnych pozycji wierzchołka, ponieważ wzdłuż ostrych krawędzi siatek spotykają się wierzchołki mające te same pozycje w przestrzeni, ale różne wektory normalne. W każdym z kolejnych „l” wierszy znajduje się jedna liczba całkowita – indeks pozycji wierzchołka na liście pozycji wierzchołków (numeracja rozpoczyna się od zera), po której następuje trójka liczb rzeczywistych – współrzędne wektora normalnego (po normalizacji).
- W następnej linijce znajduje się liczba trójkątów „m”. Każdy z kolejnych „m” wierszy zawiera trzy liczby całkowite – indeksy trzech wierzchołków składających się na trójkąt na liście wierzchołków (numeracja rozpoczyna się od zera).
- W następnej linijce znajduje się liczba krawędzi „n”. „n” jest zawsze równe ”m”*3/2, ponieważ siatki nie zawierają dziur i każda krawędź łączy dwa trójkąty. Lista krawędzi przyda się podczas tworzenia brył cienia. W każdym z „n” kolejnych wierszy znajdują się cztery liczby całkowite. Dwie pierwsze, to indeksy pozycji wierzchołków (nie mylić z indeksami wierzchołków), odpowiadających końcom danej krawędzi. Dwie kolejne to indeksy trójkątów na liście trójkątów, które zawierają daną krawędź. Numeracja w obu przypadkach rozpoczyna się od zera.

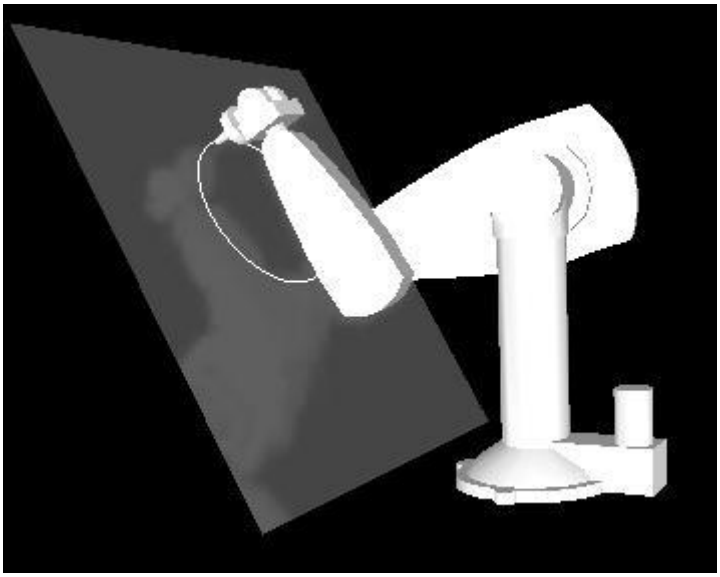


Stopnie swobody robota:
Na rysunku powyżej oznaczona została piątka kątów: a1, ..., a5 (a1 to obrót wokół osi y, a4 wokół x, pozostałe są wokół osi z), parametryzująca daną konfigurację robota. Oznaczone zostały również równania prostych, które są osiami obrotu w pozycji wyjściowej. Należy samodzielnie napisać procedurę, która obliczy odpowiednie macierze przekształceń z układu, w którym zdefiniowane są siatki do układu sceny mając dane kąty obrotów związane z połączeniami ramion robota a1,..., a5.

Rozwiązanie zadania odwrotnego:
Dla danej pozycji w scenie końcówki robota i wektora normalnego wyznaczającego jej orientację, poniższy kod policzy odpowiednie parametry konfiguracji robota, korzystając z biblioteki obliczeń na macierzach i wektorach mtxlib:

```
void inverse_kinematics(vector3 pos, vector3 normal, float &a1, float &a2,
float &a3, float &a4, float &a5)
{
	float l1 = .91f, l2 = .81f, l3 = .33f, dy = .27f, dz = .26f;
	normal.normalize();
	vector3 pos1 = pos + normal * l3;

	float e = sqrtf(pos1.z*pos1.z+pos1.x*pos1.x-dz*dz);
	a1 = atan2(pos1.z, -pos1.x)+atan2(dz, e);

	vector3 pos2(e, pos1.y-dy, .0f);
	a3 = -acosf(min(1.0f,(pos2.x*pos2.x+pos2.y*pos2.y-l1*l1-l2*l2)
/(2.0f*l1*l2)));

	float k = l1 + l2 * cosf(a3), l = l2 * sinf(a3);
	a2 = -atan2(pos2.y,sqrtf(pos2.x*pos2.x+pos2.z*pos2.z))-atan2(l,k);

	vector3 normal1;
	normal1 = vector3(RotateRadMatrix44('y', -a1) *
		  vector4(normal.x, normal.y, normal.z, .0f));
	normal1 = vector3(RotateRadMatrix44('z', -(a2+a3)) *
		  vector4(normal1.x, normal1.y, normal1.z, .0f));
	a5 = acosf(normal1.x);
	a4 = atan2(normal1.z, normal1.y);
}
```

Końcowa animacja:



W scenie należy umieścić z lewej strony robota, w odpowiedniej odległości od niego, nachylony pod pewnym kątem do płaszczyzny x=const (np. 30 stopni) prostokąt reprezentujący kawałek blachy. Niech dany będzie okrąg na powierzchni tego prostokąta. Animacja robota polega na rozwiązywaniu zadania odwrotnego dla punktu poruszającego się po okręgu i orientacji końcówki robota prostopadłej do danego prostokąta.

Pozostała część sceny:
Należy umieścić robota wewnątrz prostopadłościanu (pewne pomieszczenie) oraz dodać leżący walec z tyłu za robotem (tak, żeby na powierzchni walca widać było cień rzucany przez robota – punkt 4.).

2. *Swobodna kamera o 5-ciu stopniach swobody*

Kamera powinna mieć trzy stopnie swobody przemieszczenia i dwa stopnie swobody obrotu. Orientacja kamery powinna być zawsze taka, żeby kierunek w bok kamery leżał w płaszczyźnie równoległej do płaszczyzny y=0 (płaszczyzna pozioma). Orientację wewnątrz programu należy przechowywać przy pomocy dwóch kątów – współrzędnych sferycznych (równoleżnik i południk). Musi istnieć możliwość spojrzenia w innym kierunku poprzez zmianę któregoś z tych kątów. Kamera powinna mieć blokadę obrotu w momencie, gdy jest skierowana dokładnie w górę lub dokładnie w dół. Dla danej orientacji kamery należy zapewnić interfejs przemieszczania punktu obserwatora wzdłuż każdej z trzech osi lokalnego układu kamery dla aktualnego kierunku patrzenia (kierunki w przód-tył, lewo-prawo, góra-dół kamery).

3. *Oświetlenie*

Należy oświetlić scenę z użyciem przynajmniej jednego światła punktowego umieszczonego z lewej strony i powyżej robota. To światło będzie źródłem cienia rzucanego przez robota.

4. *Bryły cienia*

Każda z 6-ciu części robota jest siatką bez dziur, dla której dodatkowo w plikach tekstowych podane zostały dane krawędzi. Należy użyć algorytmu brył cienia do rysowania cieni rzucanych przez robota na resztę sceny. Niech fragment tego cienia będzie widoczny na powierzchni walca leżącego za robotem.

5. *System cząstek reprezentujący iskry*

Punkt, w którym aktualnie znajduje się końcówka robot niech będzie źródłem nowo powstających cząstek. Z każdą cząstką związane są: jej poprzednie i aktualne położenie, wiek, prędkość. Niech kierunek i wartość prędkości każdej nowej cząstki będą generowane losowo według dowolnego rozkładu, dla którego średni kierunek jest prostopadły do płaszczyzny blachy. Po utworzeniu, każda cząstka porusza się po paraboli pod wpływem grawitacji. W każdej klatce animacji uaktualniane są położenie, prędkość i wiek cząstki. Cząstki, których wiek przekracza ustalony okres czasu są usuwane. Na ich miejsce generowane są nowe cząstki w punkcie końcówki robota. Na początek cząstki (iskry) można rysować w postaci linii łączących nowe i poprzednie położenie cząstki. Cząstki powinny być rysowane z użyciem mieszania addytywnego po narysowaniu całej reszty sceny (mieszanie addytywne w OpenGL: glEnable(GL_BLEND); glBlendFunc(GL_SRC_ALPHA, GL_ONE);). Kanał alfa koloru cząstek powinien być tak dobrany, żeby w miarę swojego wieku, stawały się one coraz mniej widoczne. Cząstki nie powinny modyfikować bufora głębokości. Nie trzeba wykrywać kolizji cząstek z otoczeniem.

Jeśli uda się już zrealizować system cząstek reprezentujący iskry spełniający powyższe warunki proponuję ściągnąć ze strony:
http://www1.cs.columbia.edu/CAVE/databases/rain_streak_db/rain_streak.php
dowolny zestaw tekstur dla spadających kropel wody, wybrać jedną lub kilka i wykorzystać jako teksturę dla bilbordów (z kanałem przezroczystości), które zastąpią linie. Bilbord taki najlepiej rysować podając ręcznie pozycje (i współrzędne tekstury) wierzchołków czworokąta. Dwa wierzchołki

są końcami linii, pozostałe dwa powstają przez dodanie do tych dwóch małego przesunięcia w pionie (powstaje równoległobok).

Cząstki podobnie jak reszta sceny powinny się odbijać w powierzchni blachy (punkt 6.)

6. Odbicia sceny w kawałku blachy

Należy renderować odbicia sceny w powierzchni blachy. Trzeba w tym celu wykorzystać bufor szablonu i ten sam schemat postępowania, co w zadaniu z dwunastościanem foremnym. Można pominąć rysowanie cieni w odbiciu lustrzanym. Po narysowaniu odbicia należy jeszcze narysować półprzezroczysty prostokąt blachy.