Animacja szkieletu

Dokumentacja projektu

Szymon Nowak, Piotr Stefaniak, Kurek Wioletta

07.06.2019

1. Autorzy i podział pracy:

Piotr Stefaniak: rysowanie szkieletu,
Szymon Nowak: poruszanie szkieletem,
Kurek Wioletta: GUI, dokumentacja.

2. Opis projektu:

Program „Animacja szkieletu" pozwala na animowanie wygenerowanym szkieletem poprzez poruszanie kontrolerami, które są odpowiedzialne za zmianę kąta pomiędzy poszczególnymi segmentami szkieletu.

3. Założenia wstępne przyjęte w realizacji projektu:

Opis szkieletu

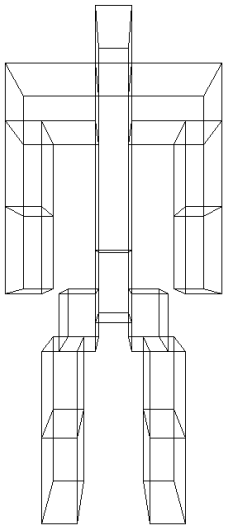
Szkielet składa się z czternastu segmentów przedstawionych na poniższym rysunku:

Rysunek 1: Schemat podziału węzłów szkieletu.

Każdemu węzłowi odpowiadają jeden lub dwa kontrolery w zależności ile dany węzęł posiada stopni swobody (np. zgięcie ręki w łokciu odbywa się za pomocą jednego kontrolera, a sterowania stawem biodrowym za pomocą dwóch) .

Opis aplikacji

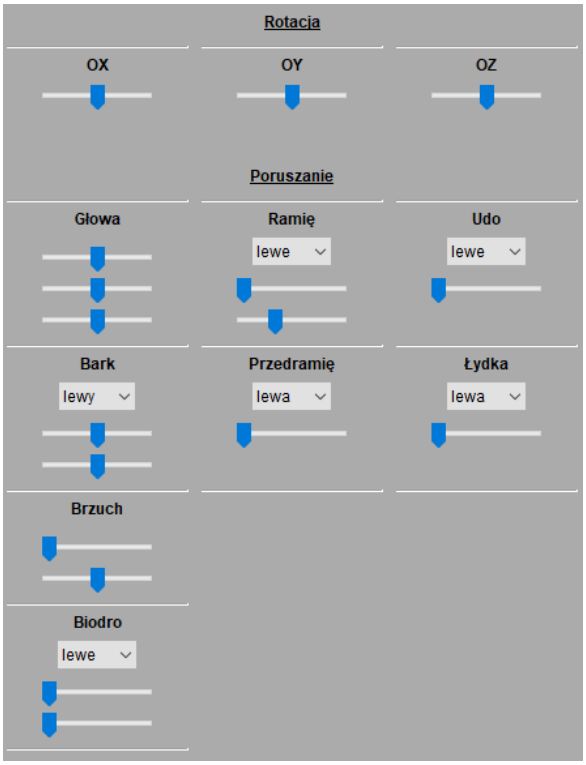
Okno aplikacji podzielone jest na dwie części. Po lewej stronie umieszczono panel z generowanym szkieletem, którym można dowolnie obracać:



Rysunek 2: Lewy panel aplikacji z wygenerowanym szkieletem.

Po prawej stronie znajduje się zestaw suwaków, które pozwalają kontrolować ruchy szkieletu. Są one podzielone na dwie sekcje:

- Rotacja: Zawiera kontrolery odpowiedzialne za rotację szkieletu w osiach x, y, z,
- Poruszanie: Zawiera kontrolery odpowiedzialne za poruszanie poszczególnymi segmentami szkieletu.



Rysunek 3: Prawy panel aplikacji kontrolujący ruchy szkieletu.

4. Analiza projektu:

Dane wejściowe

Jedynym rodzajem danych wejściowych są kąty odchylenia poszczególnych segmentów szkieletu:

- głowa – trzy punkty swobody,
- bark (prawy, lewy)– dwa punkty swobody,
- brzuch – dwa punkty swobody,
- biodro (prawe, lewe) – dwa punkty swobody,
- ramię (prawe, lewe) – dwa punkty swobody,
- przedramię (prawa, lewa) – jeden punkt swobody,
- udo (prawe, lewe) – jeden punkt swobody,
- łydka (prawa, lewa) – jeden punkt swobody,

oraz rotacji wokół osi OX, OY, OZ.

Dane wyjściowe

Wygenerowany szkielet w lewym panelu zmienia pozycję segmentów odpowiednio względem dobranych danych wejściowych dotyczących ich odchylenia. Perspektywa, z której widać szkielet zmienia się w zależności od wybranej rotacji wokół osi OX, OY, OZ.

Struktury przechowywanych danych

Dane potrzebne do rysowania szkieletu:

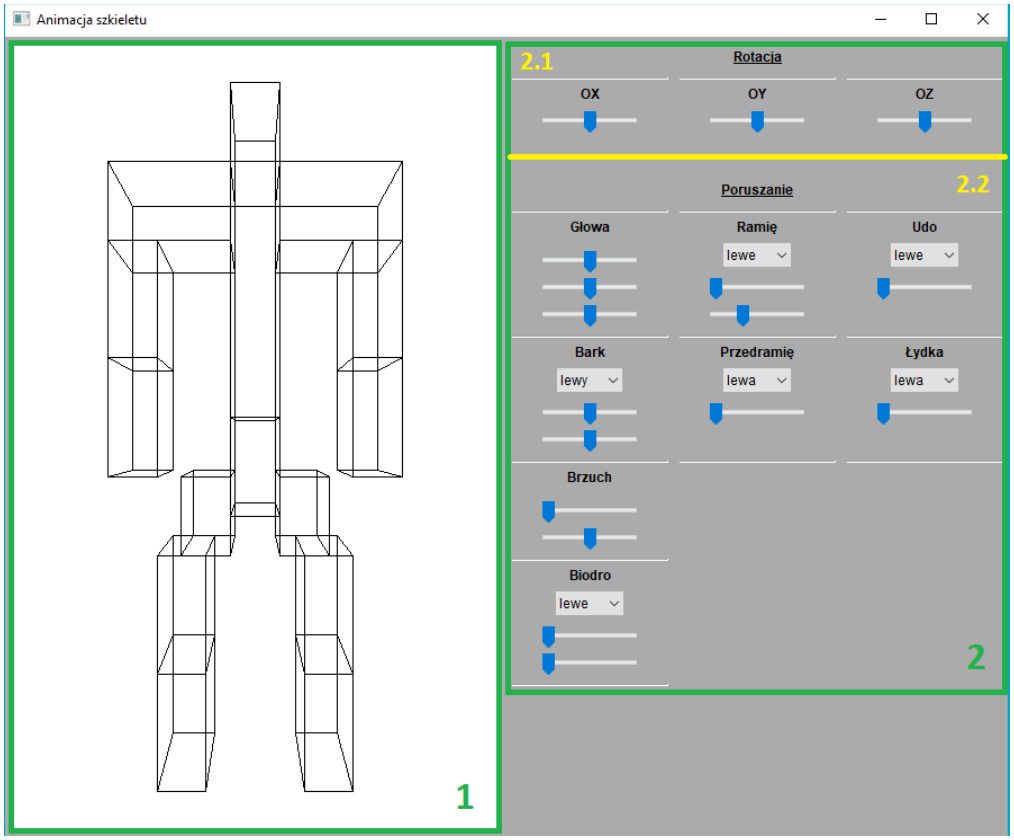
Klasa Point przechowuje współrzędne punktu w przestrzeni, klasa Segment zawiera informacje o rysowanym odcinku. Klasa Part zawiera informacje o pojedyńczej części ciała, które są zapisane z wykorzystaniem klas Point i Segment. Natomiast klasa Skeleton zawiera informacje o budowie całego szkieletu poprzez obiekty klasy Part.

Dane potrzebne do animacji szkieletu:

Dane wejściowe wprowadzone przez użytkownika przechowywane są w postaci macierzy w celu wykonania operacji rotacji i translacji obiektów (segmentów szkieletu).

Specyfikacja interfejsu użytkownika

Interfejs użytkownika został podzielony na dwie części:



Rysunek 4: Interfejs uytkownika podzielony na panele.

1) Panel z wygenerowanym szkieletem:

Służy do wyświetlania wygenerowanego modelu szkieletu, bazując na danych wejściowych wprowadzonych przez użytkownika.

2) Panel kontroli szkieletem.

Zawiera kontrolki podzielone na sekcje:

2.1) Rotacja wokół osi OX, OY, OZ:

W tej sekcji umieszczono suwaki ustawiające kąt rotacji szkieletu wokół osi OX, OY, OZ. Po zmianie pozycji suwaka, wyświetlany model obraca się o zadany kąt wokół wybranej osi.

2.2) *Poruszanie segmentami:*

W tej sekcji umieszczono suwaki odpowiadające kątowi odchylenia poszczególnych segmentów szkieletu, a także kontrolki pozwalające wybrać, którą z kończyn szkielet ma poruszać (prawa, lewa). Po zmianie pozycji suwaka, wyświetlany model porusza wybranym segmentem o zadany kąt.

Wyodrębnienie i zdefiniowanie zadań

Projekt podzielono na trzy mniejsze – zbudowanie interfejsu użytkownika, rysowanie modelu szkieletu oraz animacja poszczególnymi szkieletami.

Zbudowanie interfejsu użytkownika:

Budowa interfejsu, który umożliwił poruszanie wszystkimi segmentami szkieletu oraz jego rotacji w przestrzeni trzywymiarowej.

Rysowanie modelu szkieletu:

Stworzenie funkcjonalności odpowiedzialnej za rysowanie modelu szkieletu składającego się z wszystkich segmentów ujętych w punkcie 3. Założenia wstępne przyjęte w realizacji projektu – Opis szkieletu.

Animacja poszczególnymi segmentami:

Stworzenie funkcjonalności poruszania segmentami szkieletu zgodnie z przyjętymi założeniami i ilością punktów swobody.

Decyzja o wyborze narzędzi programistycznych

Program został napisany w środowisku Visual Studio 2017, w języku C++ oraz skompilowany za pomocą domyślnego w środowisku kompilatora Microsoft C/C++ Compiler. W celu stworzenia interfejsu skorzystano z biblioteki wxWidgets. Formularze aplikacji zostały zbudowane przy użyciu narzędzia wxFormBuilder, a platforma Trello posłużyła do rozplanowania zadań.

Visual Studio 2017:

Wybrano środowisko programistyczne Visual Studio 2017, ponieważ ułatwia ono zdalne pisanie aplikacji dzięki integracji z systemem kontroli wersji GIT oraz możliwością łatwego zarządzania bibliotekami.

wxWidgets:

Zdecydowano się na bibliotekę wxWidgets, ponieważ jest ona międzyplatformowa oraz łatwa w integracji z użytym środowiskiem.

wxFormBuilder:

Aby przyśpieszyć i ułatwić proces budowania widoków aplikacji, skorzystano z narzędzia wxFormBuilder, ponieważ umożliwia ono podgląd widoku formularzy. Dodatkowo jest to narzędzie typu WYSIWYG, dzięki czemu kod jest generowany automatycznie w oparciu o stworzony wygląd formularza.

Trello:

Skorzystano z Trello, ponieważ jest ono ogólnodostępnym i prostym narzędziem do planowania zadań, które doskonale nadaje się przy realizacji mniejszych projektów.

5. Podział pracy i analiza czasowa:

		01.05 - 05.05	08.05 - 09.05	17.05 - 18.05	19.05	27.05	28.05	29.05	01.06	02.06	03.06	04.06	07.06 - 09.06
Przygotowanie	Zapoznanie się z grafiką 3D												
	Określenie wymagań aplikacji												
	Określenie stopni swobody segmentów												
Analiza	Analiza sposobu rozwiązania problemu												
	Analiza ograniczeń aplikacji												
	Analiza sposobu wykonania aplikacji												
Projektowanie	Projektowanie architektury programu												
	Projektowanie interfejsu użytkownika												
	Wybór algorytmów												
Implementacja	Implementacja interfejsu uzytkownika												
	Implementacja rysowania szkieletu												
	Implementacja poruszania szkieletem												
Dokumentacja	Dokumentowanie postępów												
	Sporządzenie analizy projektu												
	Sporządzenie opisu programu												
	Opracowanie i opis algorytmów												

	Cały zespół
	Szymon Nowak
	Piotr Stefaniak
	Kurek Wioletta

6. Opracowanie i opis niezbędnych algorytmów:

W programie zostały użyte dwa główne algorytmy. Poniżej przedstawiono sposób wykonania każdego z nich:

Rysowanie szkieletu:

1. Utworzenie bufora pozwalającego na płynne rysowanie szkieletu
2. Stworzenie dwuwymiarowej tablicy segmentów, które odpowiadają za rysowanie poszczególnych części ciała szkieletu
3. Wykonanie punktu 3 algorytmu "*Animacja segmentem*", który został opisany poniżej
4. Wypełnienie dwuwymiarowej tablicy segmentów współrzędnymi szkieletu
5. Stworzenie i wypełnienie macierzy obrotu wokół osi OX, OY, OZ oraz macierzy rzutowania na płaszczyznę
6. Pobranie punktów początku i końca pojedyńczego segmentu i wykonanie punktów 4-7 algorytmu "*Animacja segmentem*"
7. Przemnożenie punktów przez macierz obrotu oraz ich normalizacja
8. Sprawdzenie czy punkty zawierają odpowiednie współrzędne i w zależności od nich wykonanie odpowiednich operacji skalujących współrzędne punktów oraz ich mnożenie przez macierz rzutowania na płaszczyznę
9. Normalizacja współrzędnych punktów oraz narysowanie linii łączącej wybrane punkty na ekranie.

Animacja segmentem:

1. Definicja jednej macierzy dla każdego pojedynczego punktu osi swobody.
2. Inicjalizacja macierzy w kontruktorze klasy GUIMainFrame,
3. Wywołanie funkcji Move[nazwa segmentu] – rotacjej punkty wokół odpowiedniej osi oraz pobierającej dane z kontrolek, przy starcie metody rysującej szkielet, w celu pobrania wartości domyślnych pozycji.
4. Znalezienie miejsca, w którym rysowane są linie odpowiedniego segmentu.
5. Wykonanie translacji wszystkich punktów stanowiących segment, tak aby oś obrotu pokrywała się z rzeczywistą.
6. Mnożenie punktów przez macierz obrotu.
7. Wykonanie translacji odwrotnej do translacji z pkt. 5.

7. Kodowanie:

Struktura całego programu wygląda następująco:

Klasa *MainFrame* dziedziczy po klasie *wxFrame* i w niej zawarty jest cały wygląd aplikacji. Składa się ona z konstruktora, destruktora, dwóch eventów (jeden z nich jest wirtualny i odpowiada za ponowne rysowanie gdy zostanie wykryta jakakolwiek zmiana, drugi z nich wywoływany jest w momencie wyjścia z aplikacji), wskaźnika na panel oraz masy wskaźników odpowiadających za suwaki i listy pozwalające wybrać lewą bądź prawą część ciała. Wszystkie te suwaki, listy i panel rysujący są odpowiednio ustawiane za pomocą funkcji *void setUpUi()*, którą wywołuje konstruktor. Destruktor zaś rozłącza panel i zwalnia całą zajętą pamięć.

Klasa *GUIMainFrame* dziedziczy po klasie *MainFrame* i to ona odpowiada za całe rysowanie oraz poruszanie szkieletem. Składa się ona z obiektu klasy *Skeleton*, dużej ilości macierzy odpowiedzialnych obsługę ruszania poszczególnych części ciała oraz:

- konstruktora odpowiednio inicjalizującego wartości macierzy,

- domyślnego destruktora,
- metody *Normalization*, która przyjmuje referencję do wektora i normalizuje jego wartości,
- metody *Projection*, która przyjmuje referencję do macierzy i odpowiada za rzutowanie z 3D na 2D,
- metody *Transformation*, która przyjmuje referencję do macierzy i opowiada za obracanie wokół osi OX, OY i OZ,
- metody *DrawSkeleton*, która rysuje cały szkielet na ekranie,
- całej masy metod z przedrostkiem *Move*…, z których każda przyjmuje referencję do macierzy i każda z nich odpowiada za ruszanie daną częścią ciała w odpowiedniej osi,
- eventu *WxPanel_Repaint*, który przysłania event z klasy bazowej i wywołuje on metodę *DrawSkeleton.*

Klasa *SkeletalAnimation*, która dziedziczy po klasie *wxApp* i jest odpowiedzialna za uruchamianie się aplikacji. Posiada ona jedną wirtualną metodę *OnInit*, która zwraca wartość typu bool i pozwala na uruchamianie całego programu.

Klasa *Skeleton* jest klasą odpowiedzialną za budowę całego szkieletu. Zawiera ona dużą liczbę wskaźników na obiekt klasy *Part* oraz konstruktor i destruktor które odpowiednio inicjalizują i zwalniają zawarte w klasie wskaźniki.

Klasa *Part* odpowiada za pojedynczą część ciała szkieletu. Zawiera ona osiem obiektów klasy *Point*, będących krańcami danej kończyny oraz tablicę dwunastu elementów typu *Segment*, które mówią o połączeniach pomiędzy poszczególnymi punktami. Zawiera ona też dwa konstruktory, jeden z nich przyjmuje jeden ciąg znaków będący nazwą kończyny, natomiast drugi przyjmuje nazwę kończyny oraz nazwę strony, po której dana kończyna się znajduje. Pozwalają one na odpowiednią inicjalizację współrzędnych. Destruktor tej klasy jest domyślny.

Klasa *Segment* jest implementacją pojedynczej linii w przestrzeni, która łączy dwa obiekty typu *Point*. Zawiera ona dwa wskaźniki na wyżej wspomniane obiekty oraz domyślny konstruktor i destruktor.

Klasa *Point* jest odpowiednikiem punktu w przestrzeni. Zawiera ona 3 współrzędne: X, Y, Z oraz domyślny konstruktor i destruktor.

Klasa *Matrix4* jest klasą, która implementuje obiekt matematyczny o nazwie macierz o rozmiarach 4x4, z czego każdy jej element jest typu double. Konstruktor wypełnia jej wartości zerami oprócz ostatniego elementu którego wartość wynosi 1. Ma też zdefiniowaną funkcję *Print*, która wyświetla jej zawartość na ekran oraz przeładowany operator mnożenia dla operacji macierzy razy macierz i wektor razy macierz.

Klasa *Vector4* jest klasą, która implementuje wektor o długości 4 elementów typu double. Konstruktor wypełnia jego wartości zerami oprócz ostatniego elementu którego wartość wynosi 1. Ma też zdefiniowaną funkcję *Print* wyświetlającą zawartość wektora na ekran, funkcję *Set* pozwalającą na zapisanie wartości do wektora, metody *GetX*, *GetY*, *GetX* będące getterami do poszczególnych wartości wektora oraz przeładowany operator odejmowania dla operacji wektor minus wektor i operator mnożenia dla operacji wektor razy liczba.

8. Testowanie:

Aplikację przetestowano manualnie. Sprawdzono czy wszystkie kontrolki działają poprawnie oraz czy model jest rysowany i porusza się prawidłowo.

9. Wdrożenie, raport i wnioski:

Po uruchomieniu ostatniej wersji programu sprawdzono jego poprawność działania. Program spełniał przyjęte założenia co do animacji modelem szkieletu. Wszystkie kontrolki działały prawidłowo, a sam model poruszał się odpowiednio do ustawionych danych wejściowych.

Następnym krokiem w celu rozszerzenia programu oraz jego udoskonalenia mogłoby być zbudowanie bardziej realistycznego modelu. Stworzony system rysowania szkieletu w aktualnej wersji programu pozwala na stworzenie segmentu o dowolnym kształcie. Mając odpowiednie współrzędne kształtów, nie byłoby to trudne, a dałoby bardzo zadowalające efekty.