Dokumentacja końcowa

Przedmiot

Projektowanie systemów CAD/CAM

Koordynator przedmiotu

dr inż. Paweł Kotowski

Autorzy

Franciszek Jełowicki

Tomasz Herman

Temat „Dopasowywanie chmury punktów do modelu CAD/CAM”

Iteracyjne dopasowywanie modelu CAD/CAM do chmury punktów metodą gradientów sprzężonych oraz narzędzie do generowania chmury punktów dla dowolnego modelu.

Wydział Matematyki i Nauk Informacyjnych
Politechnika Warszawska

25 czerwca 2021

1 Generowanie chmury punktów

Stworzyliśmy osobne narzędzie do generowania chmury punktów na podstawie modelu w formacie STEP. Program generuje zadaną liczbę punktów rozmieszczoną równomiernie na powierzchni modelu, następnie wygenerowana chmura jest przesuwana i obracana o zadane wartości translacji i rotacji i zapisywana do pliku w formacie STEP.

1.1 Opis interfejsu

Program udostępnia interfejs konsolowy. Sterowanie przebiegiem generowania punktów odbywa się poprzez przekazanie odpowiednich opcji do programu podczas uruchomienia. Dostępne są następujące opcje:

- `-n 1000` - Liczba punktów N która ma zostać wygenerowana. Domyślnie "1000".
- `-d 1.0, --deflection=1.0` - Parametr odpowiadający za dokładność triangulacji. Domyślnie "1.0".
- `-i "input.step", --input="input.step"` - Nazwa pliku wejściowego w formacie STEP z którego ma być stworzona chmura punktów.
- `-o "output.step", --output="output.step"` - Nazwa pliku wyjściowego do którego ma zostać zapisana chmura punktów. Chmura punktów zapisywana jest w formacie STEP. Domyślnie "output.step"
- `-x 0.0, --translationX=0.0` - Przesunięcie chmury punktów względem modelu wzdłuż osi X. Domyślnie "0.0".
- `-y 0.0, --translationY=0.0` - Przesunięcie chmury punktów względem modelu wzdłuż osi Y. Domyślnie "0.0".
- `-z 0.0, --translationZ=0.0` - Przesunięcie chmury punktów względem modelu wzdłuż osi Z. Domyślnie "0.0".
- `-X 0.0, --rotationX=0.0` - Obrót chmury punktów w stopniach względem modelu w okół osi X (Kąty eulera ZYX). Domyślnie "0.0".
- `-Y 0.0, --rotationY=0.0` - Obrót chmury punktów w stopniach względem modelu w okół osi Y (Kąty eulera ZYX). Domyślnie "0.0".
- `-Z 0.0, --rotationZ=0.0` - Obrót chmury punktów w stopniach względem modelu w okół osi Z (Kąty eulera ZYX). Domyślnie "0.0".
- `-h, --help` - Wyświetla pomoc programu.

Przykładowe wywołanie:
`pcgen -i "model.step" -o "points.step" -n 5000 -x 20 -y 10 -X 120`
wygeneruje chmurę 5000 punktów z modelu "model.step" przesuniętą o 20 wzdłuż osi X i 10 wzdłóż Y i obróconą o 120 stopni w okół X i zapiszę tą chmurę do pliku "points.step".

1.2 Opis pliku wyjściowego

Plik wyjściowy będący chmurą punktów jest modelem w formacie STEP. Model złożony jest z N krawędzi w których pierwszy punkt odpowiada pozycji a drugi punkt odpowiada wektorowi normalnemu. Żeby odczytać tak zapisany model do wektora par (pozycja, wektor normalny) można posłużyć się następującym kodem w C++:

```cpp
#include <iostream>
#include <vector>

#include <STEPCAFControl_Reader.hxx>
#include <TopoDS_Shape.hxx>
#include <BRep_Tool.hxx>
#include <TopExp_Explorer.hxx>
#include <TopoDS.hxx>
#include <TopoDS_Edge.hxx>
#include <TopoDS_Vertex.hxx>

int main(int argc, char* argv[]) {
    STEPControl_Reader reader;
    IFSelect_ReturnStatus stat = reader.ReadFile(argv[1]);
    if (stat != IFSelect_RetDone)
        throw std::runtime_error("Error reading model");

    Standard_Integer num = reader.TransferRoots();
    std::vector<TopoDS_Shape> points(num);
    for (int i = 0; i < num; i++) points[i] = reader.OneShape();

    std::vector<std::pair<TopoDS_Vertex, TopoDS_Vertex>> edges;
    for (const auto& point : points) {
        TopoDS_Vertex vv;
        int i = 0;
        for (TopExp_Explorer vertexExplorer(point, TopAbs_VERTEX);
             vertexExplorer.More(); vertexExplorer.Next(), i++) {
            const auto& vertex = TopoDS::Vertex(vertexExplorer.Current());
            if (vertex.IsNull()) continue;
            if (i % 2 == 0) vv = vertex;
            else edges.emplace_back(vv, vertex);
        }
    }
    // edges zawiera teraz listę par pozycja, wektor normalny
}
```

1.3 Algorytm generujący chmurę punktów

Algorytm generujący chmurę punktów wygląda następująco:

1. Wczytaj model dany na wejściu
2. Wygeneruj triangulację wczytanego modelu.
3. Oblicz pole powierzchni każdego trójkąta i policz w wektorze skumulowaną sumę pól powierzchni.
4. Powtórz N razy:
   - Wylosuj liczbę z przedziału 0 do całkowitej powierzchni modelu.
   - Wybierz trójkąt któremu odpowiada ta liczba na podstawie wektora skumulowanych sum.
   - Wylosuj punkt na wybranym trójkącie i dodaj go do wyniku razem z wektorem normalnym tego trójkąta.
5. Obróć i przesuń wszystkie punkty.
6. Zapisz wygenerowane punkty do pliku.

2 Dopasowywanie chmury punktów do modelu

2.1 Program

Główny moduł składa się z dwóch części. Pierwszy program *„cadcam"* jest odpowiedzialny za obliczenie wartości translacji i rotacji. Oczekuje on podanie dwóch ścieżek do plików STEP zawierających model oraz chmurę punktów. Wynikiem działania programu jest plik matrices.txt zawierający listę translacji i rotacji w kolejnych iteracjach. Składa się on z wielu wierszy, każdy zawiera siedem liczb $T_xT_yT_zR_xR_yR_za$, gdzie T to translacja, R - oś obrotu a a - kąt.

Drugi program - *„gui"* jest odpowiedzialny za wizualizację wyników. Oczekuje on trzech ścieżek - do chmury punków, pliku wynikowego pierwszego programu oraz ztriangulowanego modelu w formacie STEP. Po uruchomieniu program wyświetla chmurę punków i model oraz pozwala przełączać się między kolejnymi iteracjami przy użyciu strzałek na klawiaturze.

2.2 Algorytm

Aby dopasować chmurę punktów do modelu zastosowaliśmy metodę gradientu prostego. Dla każdego punktu obliczany jest gradient dla translacji oraz obrotu zgodnie ze wzorami:

$$d = 2\alpha N$$

$$\omega_x = 2\alpha(N_z\Delta_y - N_y\Delta_z) + 2(1-\alpha)(n_yN_z - n_zN_y)$$

$$\omega_y = 2\alpha(N_x\Delta_z - N_z\Delta_x) + 2(1-\alpha)(n_zN_x - n_xN_z)$$

$$\omega_z = 2\alpha(N_y\Delta_x - N_x\Delta_y) + 2(1-\alpha)(n_xN_y - n_yN_x)$$

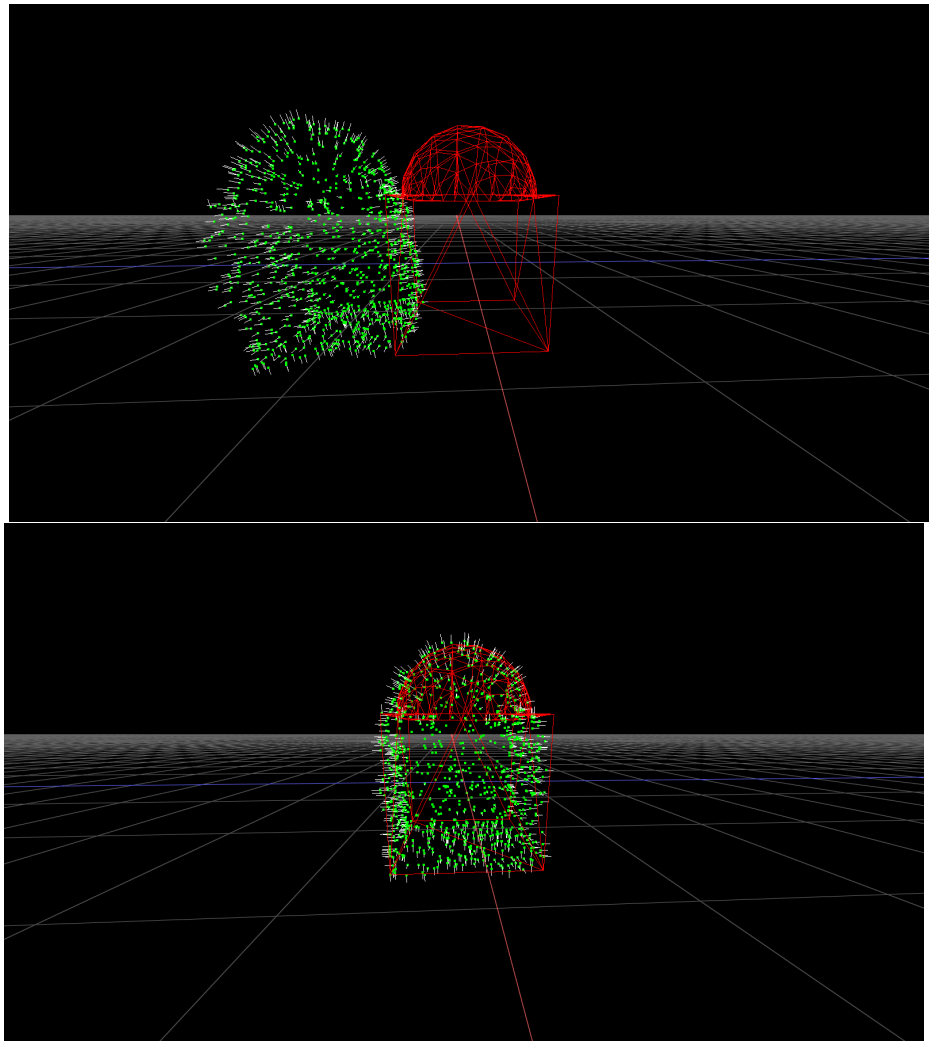
Gdzie α jest współczynnikiem określającym stosunek obrotu do rotacji, Δ oznacza różnicę współrzędnych danego punktu oraz najbliższego punktu na module, n oznacza wektor normalny punktu zaś N wektor normalny najbliższego punktu na modelu.

Następnie wartości z każdego punktu w chmurze są uśredniane. Uśrednione d to wartość translacji, ω wyznacza oś obrotu zaś jej długość kąt. Chmura punktów jest przesuwana zgodnie z wyznaczonymi wartości, po czym krok jest powtarzany tak długo, aż algorytm jest zbieżny, tzn. aż średniokwadratowa odległość punktów do modelu maleje.

W powyższym wzorze wektor normalny jest traktowany jako przybliżenie odpowiednich pochodnych cząstkowych. Z tego powodu rotacja może być prawidłowa jedynie dla niewielkich kątów, tak aby móc stosować przybliżenie $sin\alpha = \alpha$.

3 Rezultaty

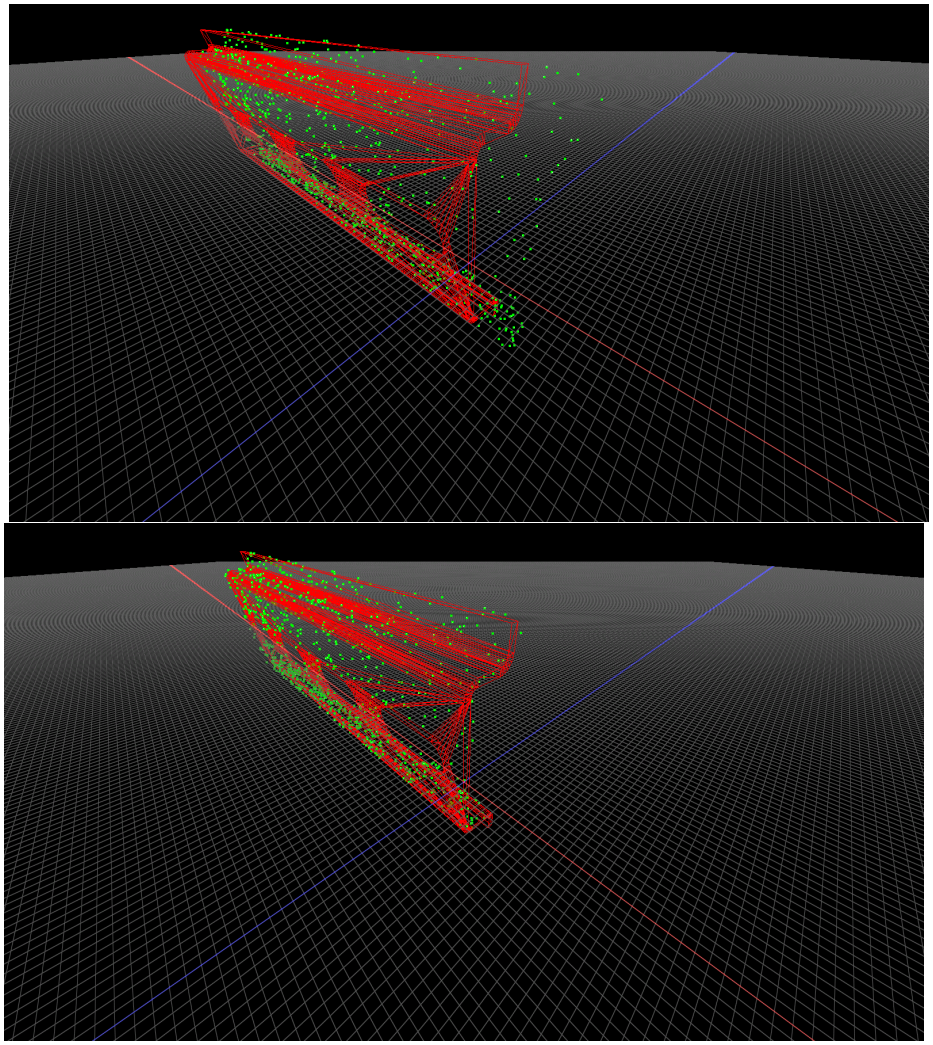
Podstawowym modele na którym testowaliśmy program była „kostka z kopułą” - sześcian którego jedna ze ścian została zastąpiona półsferą. Dla tego modelu program uzyskiwał dobre wyniki - niezależnie od translacji, dla niewielkich kątów rotacji (mniejszych niż 10 stopni) oraz zaburzeń punktów algorytm był zawsze zbieżny a wyniki satysfakcjonujące.



Rysunek 1: Dopasowanie „kostki z kopułą” - 1000 punktów, translacja oraz rotacja wokół osi X a także zaburzenia 0,05. Błąd średniokwadratowy dopasowania - 0,07.

Problemem z którym mierzyliśmy się przy prawdziwych modelach były bardzo duże wartości kąta obrotu. W efekcie cała metoda nie zbiegała. Aby to poprawić wprowadziliśmy ograniczenie obrotu w jednej iteracji do 0,02 radiana. Dzięki temu dla niektórych modeli zaczęliśmy otrzymywać dobre wyniki. Nadal jednak występuje szereg problemów:

- Możliwość wpadnięcia w minimum lokalne - zdarza się szczególnie w modelach zawierających otwory, wypustki czy wypełnienie.
- Błędny obrót - powiązane z pierwszym - przy jednoczesnej translacji i rotacji istnieje możliwość, że obrót zostanie wykonany w złym kierunku gdy chmura będzie odległa od modelu a następnie taka źle obrócona chmura będzie dopasowywana i trafi w minimum lokalne.
- Wydajność - aby metoda była poprawna konieczne jest znajdowanie najbliższego punktu na modelu dla każdego punktu. Oznacza to, że złożoność jednej iteracji wynosi nmb, gdzie n to liczba punktów, m - liczba powierzchni na modelu zaś b to złożoność wyszukiwania odległości między jednym punktem a jedną powierzchnią. Im bardziej skomplikowany model tym więcej punktów powinna zawierać chmura w celu uzyskania dokładnego wyniku. W związku z tym dla nieco bardziej złożonych modeli obliczenia zajmują dużo czasu



Rysunek 2: Dopasowanie modelu metalowej szyny montażowej. U góry translacja w X na dole w Z. W pierwszym przypadku algorytm wpadł w minimum lokalne, w drugim dopasowanie jest bardzo dobre.

4 Przemyślenia dotyczące Open Cascade

Osobiste przemyślenia z użytkowania biblioteki. Biblioteka używana była na systemach z rodziny Linux.

4.1 Dołączanie do projektu

Dołączanie do projektu również było proste w naszym przypadku. Do budowania systemu budowania używaliśmy CMake, w którym wystarczyło podlinkować do naszych plików wykonalnych bibliotekę. Biblioteka składa się z dużej ilości modułów, co z jednej strony może poprawić optymalizację, z drugiej utrudnia stworzenie minimalnego działającego projektu - co chwila czegoś brakuje i trzeba szukać odpowiedniego modułu (a ostatecznie dołączyliśmy wszystkie).

4.2 Dokumentacja

Wszystkie funkcjonalności są opisane w oficjalnej dokumentacji. Jest to dokumentacja generowana automatycznie z kodu źródłowego. Z tego powodu często brakuje dokładnego opisu funkcjonalności oraz przykładów użycia. Na przykład gdy chcieliśmy sprawdzić dostępne algorytmy dla `BRepExtrema_DistShapeShape` nie miały one dokumentacji: `https://dev.opencascade.org/doc/refman/html/_extrema___ext_algo_8hxx.html`. Bardzo pomocne w szukaniu przykładów kodu okazywało się forum opencascade, na którym często można było znaleźć gotowe rozwiązania problemów. Również przydatnym źródłem okazał się blog `https://techoverflow.net/category/opencascade/` oraz związana z nim biblioteka OCCUtils, gdzie można było znaleźć przydatne fragmenty kodu.

4.3 Funkcjonalność

Biblioteka zawierała wszystkie potrzebne przez nasz projekt funkcjonalności do operowania na modelach cadowych. Ciekawe okazało się narzędzie Draw, które pozwala testować funkcjonalności biblioteki w interpreterze komand bazującym na TCL. Narzędzie to pozwoliło nam stworzyć prosty model, wywołać na nim funkcję, jak również coś narysować.

Sporym utrudnieniem w korzystaniu z biblioteki jest konwencja nazewnictwa, nieprzystająca do nowoczesnych projektów. Każda nazwa zaczyna się od przedrostka związanego z modułem/działem w jakim znajduje się dana klasa/funkcja. Jako że biblioteka jest napisana w c++ można byłoby to rozwiązać w dużo elegantszy sposób przy pomocy przestrzeni nazw (namespaces). Ogólnie problemem biblioteki jest niestosowanie funkcji nowoczesnego c++ (c++11 i nowszych). Jest to zrozumiałe ze względu na wiek biblioteki, jednak bardzo utrudnia jej używanie w nowych projektach, osobom przyzwyczajonym do tych funkcjonalności (jak wiadomo między c++ przed wersją 11 i po niej występują ogromne różnice).

4.4 Opinia

Użycie gotowych rozwiązań z biblioteki zaoszczędziło nam dużo czasu. Własna implementacja triangulacji, czy znajdowania pary najbliższych punktów między obiektami byłaby trudna i czasochłonna. Mimo, że dokumentacja nie zawiera wielu przykładów użycia, to często można znaleźć pomoc na forum albo w innych źródłach. Biblioteka jest otwartoźródłowa, ale nie jest to zbyt pomocne, biblioteka jest zbyt rozbudowana i skomplikowana, żeby można było zrozumieć kod źródłowy. Na pewno warto znać tą bibliotekę, gdyż nie brakuje w niej funkcjonalności i jest bezpłatna. Użycie biblioteki wydaje się uzasadnione jeżeli zakres projektu tego wymaga, jeżeli jednak potrzeba tylko pojedynczych funkcjonalności warto rozejrzeć się za alternatywami.

5 Wnioski

Prezentowana metoda, mimo że niewątpliwie poprawna, daje niezadowalające wyniki w praktycznych zastosowaniach. Próba użycia modelu zawierającego dziury lub innego rodzaju nieregularności powoduje, że algorytm wpada w minimum lokalne. Z drugiej strony osiągane wyniki można poprawić poprzez zastosowanie różnych heurystyk. Przykładem takowej było sztywne ograniczenie kąta obrotu do 0,02 radiana, które pozwoliło na znaczną poprawę w przypadku niektórych modeli. Znalezienie heurystyki na wstępne dopasowanie mogłoby pozwolić uniknąć większości problemów z jakimi boryka się aktualny program.