VJEŽBA 7: ITERATIVNI ALGORITAM NAJBLIŽE TOČKE

I. Cilj vježbe: *Naučiti primijeniti iterativni algoritam najbliže točke.*

II. Opis vježbe:

Iterativni algoritam najbliže točke je algoritam koji se upotrebljava za određivanje najbolje transformacije *T* koja koja preslikava jedan skup u drugi. Transformacija se može odrediti na sljedeći način:

1. Svaka točka polazišnog skupa se sparuje s točkom odredišnog skupa.
2. Pronalazi se transformacija *T* koja minimizira funkciju troška.
3. Sve točke polazišnog skupa se transformiraju pomoću dobivene transformacije *T*.
4. Prva tri koraka se ponavljaju dok promjene transformacije *T* ne budu unutar zadanog praga.

U vježbi se upotrebljava iterativni algoritam najbliže točke i VTK biblioteka. Potrebno je odrediti optimalne parametre za najbolju registraciju za dane trodimenzionalne modele.

III. Rad na vježbi:

a) Implementirati registraciju oblaka 3D točaka priloženih modela primjenjujući iterativni algoritam najbliže točke implementiran u VTK biblioteci.
b) Utvrditi optimalne parametre (broj iteracija, maksimalni broj korespondencija) za vizualno najbolju registraciju koju je moguće postići u što kraćem vremenu.
c) Vizualizirati registraciju te napisati izmjerena vremena i upotrijebljene parametre.

Prilog A:

Iterativni algoritam najbliže točke

Neka su *A* i *B* skupovi 3D točaka. Transformacija **T** koja skup *A* transformira na način da se što bolje podudara sa skupom *B* može se odrediti na sljedeći način:

1. Odredi skup parova $W = \{(i, j)\}$ takvih da je za svaku točku P_i *A* točka $P_j' \in B$ najbliža od svih točaka skupa *B*, tj. $P_j' = \mathrm{N}(P_i, B)$.
2. Pronađi **T** koja minimizira funkciju troška

$$\Im(\mathbf{T}) = \sum_{(i,j)\in W} \left\| \mathbf{p}_j' - \mathbf{R}\,\mathbf{p}_i + \mathbf{t} \right\|^2,$$

gdje je

$$\mathbf{T} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}^{3\times 1} & 1 \end{array}\right],$$

a **p** označava vektore koordinata točaka P_i *A* odnosno $P_j' \in B$.

3. Transformiraj sve točke skupa *A* pomoću transformacije **T**.
4. Ponovi postupak od koraka 1, sve dok promjene transformacije **T** ne budu manje od nekog zadanog praga.

Implementacija iterativnog algoritma najbliže točke u VTK biblioteci se nalazi u klasi ***vtkIterativeClosestPointTransform*** (vtkmodules.vtkCommonDataModel). Rezultat rada te klase je transformacijska matrica **T** koju je potrebno primijeniti koristeći klasu ***vtkTransformPolyDataFilter*** (vtkmodules.vtkFiltersGeneral) koja radi kao filter pošto na osnovu ulaznih podataka (***vtkPolyData***) generira nove podatke (također ***vtkPolyData***) koji predstavljaju transformirani originalni objekt (koordinate zapisane u ***vtkPoints*** objektu su transformirane).
Primjer korištenja:

```
icp = vtkIterativeClosestPointTransform()
icp.SetSource(sourcePD)      #Ulazni objekt (početna poza objekta)
icp.SetTarget(targetPD)   #Konačni objekt (željena poza objekta)
icp.GetLandmarkTransform().SetModeToRigidBody() #Potrebni način rada je transformacija za kruta tijela
icp.SetMaximumNumberOfIterations(20)   #Željeni broj iteracija
icp.SetMaximumNumberOfLandmarks(1000)   #Koliko parova točaka da se koristi prilikom minimiziranja cost funkcije
icp.Update()        #Provedi algoritam

icpTransformFilter  = vtkTransformPolyDataFilter()
icpTransformFilter.SetInputData(source)   #Objekt s početnim koordinatama
icpTransformFilter.SetTransform(icp)   #transformiramo na novi položaj koristeći transformacijsku matricu
icpTransformFilter.Update()

icpResultPD = icpTransformFilter.GetOutput() #Transformirani (novi) objekt
```

Prilog B:

Učitavanje PLY datoteka

Učitavanje se vrši korištenjem klase ***vtkPLYReader*** (vtkmodules.vtkIOPLY).
Primjer korištenja:

```
plyReader  = vtkPLYReader()
plyReader.SetFileName("bunny.ply") #Putanja do željene datoteke
plyReader.Update()  #Učitaj
```

```
inputPD = plyReader.GetOutput() #Učitana geometrija se nalazi u vtkPolyData objektu
```

Prilog C:

Mjerenje vremena u Pythonu

```
import time

start_time = time.time()
(...)
end_time = time.time()
diff = end_time – start_time #in seconds
```

Prilog D:

Osnovne VTK klase potrebne za vježbu

- ***vtkRenderer*** (vtkmodules.vtkRenderingCore) - Objekt koji kontrolira proces renderiranja na ekranu, odnosno proces iscrtavanja 2D slike nekog 3D objekata uzimajući u obzir osvjetljenje, položaj kamere u prostoru i sl. Vodi računa o transformaciji koordinata 3D prostora u koordinate slike. Sadrži popis svih objekata i njihovih stanja na sceni.
  Primjer korištenja:
  ```
  renderer = vtkRenderer()
  renderer.SetBackground(1.0, 1.0, 1.0)  #Bijela pozadina
  renderer.AddActor(sphereActor) #Dodaj neki objekt na scenu
  renderer.ResetCamera()  #Centriraj kameru tako da obuhvaća objekte na sceni
  ```

- ***vtkRenderWindow*** (vtkmodules.vtkRenderingCore) - Predstavlja prozor unutar kojeg će se crtati grafičko sučelje i prikazivati renderirane slike scene. Mora biti povezan s najmanje jednim ***vtkRenderer*** objektom. Definira standardne parametre prozora kao što je naziv, veličina i sl.
  Primjer korištenja:
  ```
  window = vtkRenderWindow()
  window.AddRenderer(renderer) #Moguće je dodati i više renderera na jedan prozor
  window.SetSize(800, 600)  #Veličina prozora na ekranu
  window.SetWindowName("Scena")  #Naziv prozora
  window.Render() #Renderaj scenu
  ```

- ***vtkRenderWindowInteractor*** (vtkmodules.vtkRenderingCore) - Klasa koja omogućava upravljanje događajima kao što je klik miša na prozor ili pritisak neke tipke na tipkovnici (klasa ***vtkCallbackCommand***). Mora biti povezan s nekim ***vtkRenderWindow*** objektom da bi radio.
  Primjer korištenja:

  ```
  interactor = vtkRenderWindowInteractor()
  interactor.SetRenderWindow(window)
  (...)
  interactor.Start() #Pokretanje interaktora, potrebno kako se vtk prozor ne bi odmah zatvorio
  ```

- ***vtkPolyData*** (vtkmodules.vtkCommonDataModel) - Objekti ove klase predstavljaju geometrijske strukture u prostoru. Te strukture mogu biti točke, linije, trokuti, trake trokuta i poligoni. Pored geometrijskih struktura i položaja točaka u prostoru, ovi objekti također mogu sadržavati i dodatne podatke kao što su boja po ćeliji ili točki, normale ili bilo koji drugi skalarni ili vektorski podaci.

- ***vtkPLYReader*** (vtkmodules.vtkIOPLY) **–** Služi za učitavanje PLY datoteka, kao što je ranije naznačeno. Objekt ove klase ima metodu *GetOutput()* koja vraća objekt klase **vtkPolyData**.

- ***vtkPolyDataMapper*** (vtkmodules.vtkRenderingCore) - Ova klasa olakšava iscrtavanje grafičkih primitiva (točka, trokut i sl.) na ekran. Olakšava posao ***vtkRenderer*** klasi da vizualizira neki objekt. Zahtjeva objekt klase ***vtkPolyData*** kao ulazni podatak. Postoje različiti „mapperi“ za različite tipova podataka koje se žele vizualizirati, a ovaj je za ***vtkPolyData*** tip podataka.
  Primjer korištenja:
  ```
  mapper = vtkPolyDataMapper()
  mapper.SetInputData(pd) #Ulazni podatak je objekt tipa vtkPolyData
  ```

- ***vtkActor*** (vtkmodules.vtkRenderingCore) - Objekt ove klase predstavlja objekt (geometrija s ostalim svojstvima) na 3D sceni. Veže se na objekt tipa npr. ***vtkPolyDataMapper*** („mapper“ ne mora nužno biti definiran za „polydata“ tip podataka). Sadrži dodatna svojstva kao što je položaj u prostoru, tekstura, veličina točaka (prilikom iscrtavanja na ekran) i sl. Objekt ove klase se dodaje u ***vtkRenderer*** objekt radi dodavanja u scenu i posljedičnog iscrtavanja na ekran.
  Primjer korištenja:
  ```
  actor = vtkActor()
  actor.SetMapper(mapper)   #Povezujemo ga s mapperom za određeni tip podataka
  actor.GetProperty().SetColor(1.0, 0.0, 0.0)  #Objekt će biti obojan u crveno
  actor.GetProperty().SetPointSize(5)   #Veličina će biti 5x5 piksela po točci
  renderer.AddActor(actor)   #Dodajemo ga na popis objekata na sceni
  ```

Prilog E:

Kako bi program ispravno radio, potrebno je dodati sljedeću liniju na početak python datoteke:

```
import vtk
```