FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Diplomski studij računarstva

Računalna geometrija i robotski vid

Laboratorijska vježba 7

ITERATIVNI ALGORITAM NAJBLIŽE TOČKE

Ivan Gudelj, DRB

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:

A. Svaka točka polazišnog skupa se sparuje s točkom odredišnog skupa.
B. Pronalazi se transformacija T koja minimizira funkciju troška.
C. Sve točke polazišnog skupa se transformiraju pomoću dobivene transformacije T.
D. 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.

RJEŠENJE :

Analiziramo rješenje tako što pokrenemo program više puta s različitim parametrima kako bi se prošlo kroz sve parametre i moglo vidjeti uz koju kombinaciju parametara dobijamo vizualno najbolju registraciju koju je moguće postići u što kraćem vremenu. Parametri kroz koje se prolazi su NUMBER_OF_ITERATIONS (broj iteracija) i NUMBER_OF_LANDMARKS (maksimalni broj korespondencije).

```
import vtk
import time
from vtkmodules.vtkCommonDataModel import vtkIterativeClosestPointTransform
from vtkmodules.vtkFiltersGeneral import vtkTransformPolyDataFilter
from vtkmodules.vtkIOPLY import vtkPLYReader
from vtkmodules.vtkRenderingCore import vtkRenderer
from vtkmodules.vtkRenderingCore import vtkRenderWindow
from vtkmodules.vtkRenderingCore import vtkRenderWindowInteractor
from vtkmodules.vtkCommonDataModel import vtkPolyData
from vtkmodules.vtkRenderingCore import vtkPolyDataMapper
from vtkmodules.vtkRenderingCore import vtkActor
from vtkmodules.vtkCommonCore import vtkLookupTable
from vtkmodules.vtkRenderingCore import vtkTextActor
from vtkmodules.vtkRenderingAnnotation import vtkScalarBarActor
from vtkmodules.vtkInteractionWidgets import vtkScalarBarWidget

NUMBER_OF_ITERATIONS=100
NUMBER_OF_LANDMARKS=1000
```

Učitavanje 3D modela napravljeno pomoću vtkPolyData - 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- služi za učitavanje poligonalnih podataka u formatu datoteke PLY.

```
#Učitavanje modela odlomka (Taj treba fitat)
##########################################
partialPlyReader = vtkPLYReader()
partialPlyReader.SetFileName('modeli/bunny_t5_parc.ply')
partialPlyReader.Update()
inputPartialPlyReader = partialPlyReader.GetOutput()
##########################################

#Učitavanje modela cijelog zeca
##########################################
fullPlyReader = vtkPLYReader()
fullPlyReader.SetFileName('modeli/bunny.ply')
fullPlyReader.Update()
targetedinputPlyReader = fullPlyReader.GetOutput()
##########################################
```

vtkPolyDataMapper - 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.

vtkActor - 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.

```
#CRVENI ZEC - POČETNA POZICIJA ODLOMKA
########################################
polyDataMapper = vtkPolyDataMapper()
polyDataMapper.SetInputData(inputPartialPlyReader)

primaryActor = vtkActor()
primaryActor.SetMapper(polyDataMapper)
primaryActor.GetProperty().SetColor(1.0, 0.0, 0.0)
primaryActor.GetProperty().SetPointSize(5)
########################################

#ZELENI ZEC - CILJANA POZICIJA NA CIJELOM ZECU
########################################
secondaryPolyaDataMapper = vtkPolyDataMapper()
secondaryPolyaDataMapper.SetInputData(targetedinputPlyReader)

secondaryActor = vtkActor()
secondaryActor.SetMapper(secondaryPolyaDataMapper)
secondaryActor.GetProperty().SetColor(0.0, 1.0, 0.0)
secondaryActor.GetProperty().SetPointSize(5)
########################################

#PLAVI ZEC - REZULTAT TRANZICIJE
########################################
theraryPolyDataMapper = vtkPolyDataMapper()
theraryPolyDataMapper.SetInputData(icpResult)

theraryActor = vtkActor()
theraryActor.SetMapper(theraryPolyDataMapper)
theraryActor.GetProperty().SetColor(0.0, 0.0, 1.0)
theraryActor.GetProperty().SetPointSize(5)
########################################
```

Ispis podataka kao što su skala udaljenosti poklapajućih točaka (trenutni odsnos između točaka osnovnog modela i parcijalnih dijelova) se vrši koristeći se vtkTextActorom i vtkScalarBarActorom.

```
#Pokušaj textActora
########################################
textActor = vtkTextActor()
textActor.SetInput("Number Of iterations is :" + str(NUMBER_OF_ITERATIONS) + "\nNumber of Landmarks is:" +
                   str(NUMBER_OF_LANDMARKS) + "\nTime Needed for matching:" + str(timeDifference))
textActor.GetTextProperty().SetFontSize(20)
textActor.SetPosition2(10,40)
textActor.GetTextProperty().SetColor(0.5,0.5,0.5)
########################################
```

```
#Iscrtavanje vektora poklapanja
################################################
scalar_bar = vtkScalarBarActor()
scalar_bar.SetOrientationToHorizontal()
scalar_bar.SetLookupTable(lut)

scalar_bar_widget = vtkScalarBarWidget()
scalar_bar_widget.SetInteractor(originalInteractor)
scalar_bar_widget.SetScalarBarActor(scalar_bar)
scalar_bar_widget.On()
################################################
```

Interactor odnosno metoda za prepoznavanje promjene pogleda od strane korisnika (promjena pogleda na modele) se vrši pomoću vtkRenderMonitorInteractor-a.

```
#Postavljanje interactora za korisnički pregled
############################################
originalInteractor = vtkRenderWindowInteractor()
originalInteractor.SetRenderWindow(showingWindow)
originalInteractor.Initialize()
showingWindow.Render()
############################################
```

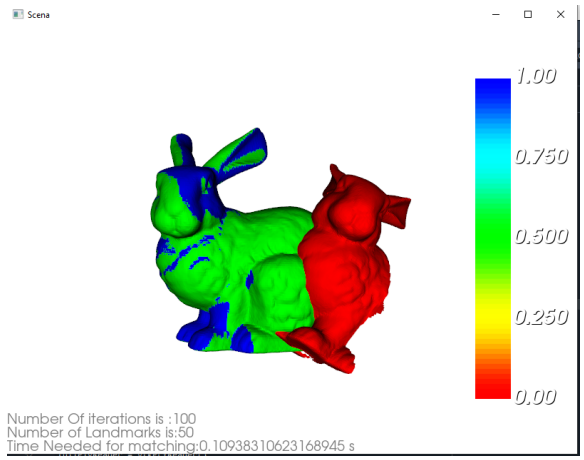
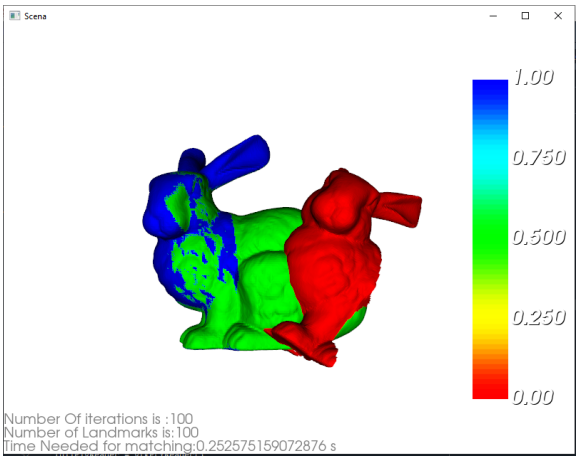
Renderiranje na ekranu je omogućeno preko VTK klase vtkRenderer - Objekt koji kontrolira proces renderiranja na ekranu, odnosno proces iscrtavanja 2D slike nekog 3D objekata uzimajući u obzir osvjetljenje, položaj kamere u prostoru i sl. Vodi računa o transformaciji koordinata 3D prostora u koordinate slike. Sadrži popis svih objekata i njihovih stanja na sceni. vtkRenderWindow - Predstavlja prozor unutar kojeg će se crtati grafičko sučelje i prikazivati renderirane slike scene.

```
#Učitavanje scene
############################################
realRenderer = vtkRenderer()
realRenderer.SetBackground(1.0, 1.0, 1.0)
realRenderer.AddActor(primaryActor)
realRenderer.AddActor(secondaryActor)
realRenderer.AddActor(theraryActor)

realRenderer.AddActor2D(textActor)


showingWindow = vtkRenderWindow()
showingWindow.SetSize(800, 600) #Veličina prozora na ekranu
showingWindow.SetWindowName("Scena")
showingWindow.AddRenderer(realRenderer)
############################################
```

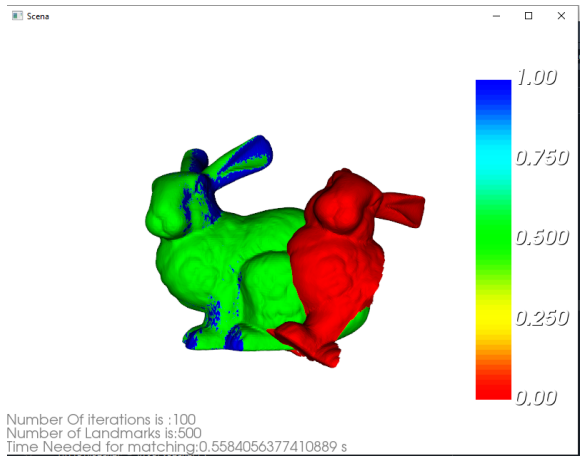
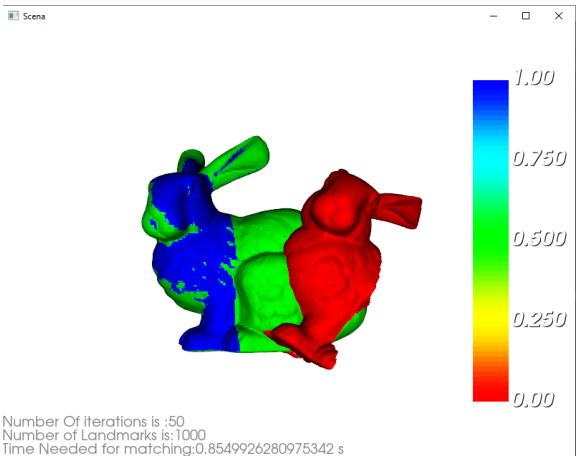
Sam prikaz rezultata modela na ekranu kao i prikaz podataka kaoi što su broj iteracija, broj korespodencije i vrijeme potrebno se vrši pomoću sljedećih naredbi.

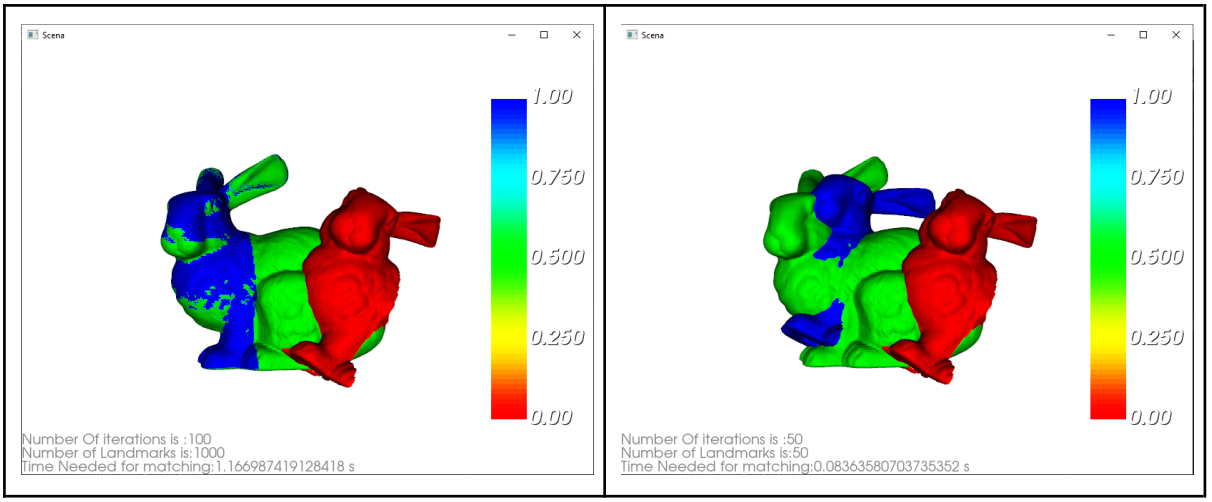
```
#Prikaz prozora
########################################
showingWindow.Render()
showingWindow.Render()
originalInteractor.Start()
########################################
```

Nakon provedenih svih kombinacija broja iteracija(10, 50, 100, 500, 1000) i broja korespodencije(10, 50, 100, 500, 1000) kao najoptimalnije rješenja dobivamo sljedeće:

Iterations= 100 Landmarks=50 Time[s]=0.1093	Iterations= 100 Landmarks=100 Time[s]=0.2526
	

Također više nasumičnih primjera u nastavku:


Scena
1.00
0.750
0.500
0.250
0.00
Number Of iterations is :100
Number of Landmarks is:1000
Time Needed for matching:1.166987419128418 s
Scena
1.00
0.750
0.500
0.250
0.00
Number Of iterations is :50
Number of Landmarks is:50
Time Needed for matching:0.08363580703735352 s