|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU** |  |

Nikola Berdić (E2-66/2015)

Autorigovanje 3D

modela pomoću „Pinocchio“

programske biblioteke

Seminarski rad

- iz predmeta Multimedijalni sistemi -

Mentor:

Prof. dr Dragan Ivetić

**1. Uvod**

U kompjuterskoj animaciji, skeletalna animacija je često korišćena kao intuitivna metoda za modelovanje. U slučaju animacije skeleta, meš modela je predstavljen sa odgovarajućim skeletom, koji određuje na koji način će se meš model promeniti u slučaju kretanja. Proces kreiranja i uklapanja skeleta sa modelom naziva se rigovanje i uglavnom se radi ručno, pomoću softvera za 3D modelovanje, kao što su Maya ili Blender.

Ručno rigovanje može biti obeshrabrujuće za početnike, ali i vrlo vremenski zahtevno za eksperte zato što svaki model mora biti rigovan odvojeno. U radu biće proučena programska biblioteku za automatsko rigovanje meš modela „Pinocchio“. Glavna namena programske biblioteke je da omogući automatsko rigovanje bez ikakve pomoći korisnika.

Zadatak seminarskog rada je da prouči „Pinocchio“ programsku biblioteku, da da ocenu njene upotrebljivosti i da je proširi tako da se obezbedi jednostavnije okruženje (sa naprednim elementima interakcije) za njeno korišćenje.

Upotrebljivost „Pinocchio“ programske biblioteke biće odrađena tako što će se proučiti sledeća tri aspekta programske biblioteke:

* Preformanse – izvršavanje procesa rigovanja pomoću programske biblioteke bi trebao da traje manje od ručnog rigovanja,
* Samostalnost – izvršavanje procesa rigovanje pomoću biblioteke bi trebao što više da smanji umešanost korisnika,
* Uopštenost – upotrebljovost biblioteke bi trebala da bude na različitim tipovima modela.

U prvom poglavlju seminarskog rada je opisan način na koji funkcioniše „Pinocchio“ programska biblioteka. U drugom poglavlju su proučena tri aspekta „Pinocchio“ programske biblioteke. U trećem poglavlju seminarskog rada opisan je programski kod koji omogućuje korišćenje „Pinocchio“ programske biblioteke i njegova funkcionalnost aplikacije. Na kraju seminarskog rada doneti su zaključci o „Pinocchio“ programskoj biblioteci uz konstatacije kako se sve ova biblioteka može primeniti.

**2. Opis „Pinocchio“ programske biblioteke**

Programska biblioteka „Pinocchio“ je kreirana 2007 godine i njeno pokretanje se vrši iz komandne linije. Ovaj pristup znatno otežava korišćenje programske biblioteke korisnicima sa prosečnim znanjem o računaru. To je razlog zbog kojeg programska biblioteka mora da se prilagodi radu na Windows 10 operativnom sistemu i da se proširi grafičkim korisničkim interfejsom koji bi omogućio lakšu konfiguraciju parametara za rigovanje. Rad sa programskom bibliotekom „Pinocchio“ je vrlo zahtevan i težak za korisnika sa prosečnim znanjem računara. Takođe, na ovaj način nije moguće izvršiti rigovanje više meš modela od jednom. Razvijanje grafičkog korisničkog interfejsa bi olakšao podešavanje parametara i dalo mogućnost rigovanja većeg broja meš modela.

Metoda rigovanja koja je iskorišćena može da se podeli u dva uzastopna koraka. Generisanje i postavljanje skeleta unutar modela i „skinning“ modela, što predstavlja kako će se model menjati skeletnom strukturom.

**2.1. Generisanje skeleta i njegovo postavljanje**

Prvi korak prilikom rigovanja je generisanje i postavljanje skeleta. Ovaj proces se vrši na jedan od dva načina, ugrađivanjem ili izvlačenjem skeleta iz 3D modela [1]. Prilikom ugrađivanja skeleta, postoji skelet šablon koji je potrebno ugraditi u model na najoptimalniji način. Kod izvlačenja skeleta, unutrašnjost modela se proučava i na osnovu proučene unutrašnjosti skelet za taj model se izvlači. Glavna razlika između ova dva procesa je ta da za proces ugrađivanje skeleta potreban je ulazni (šablon) skelet. Samim tim ovaj način zahteva inicijalnu interferenciju korisnika.

„Pinocchio“ biblioteka koristi prvi pristup za generisanje i postavljane skeleta, a to je ugrađivanje skeleta u 3D model [1]. Ugrađivanje skeleta zahteva već kreirane skelete koji će biti ugrađeni u meš, kao i funkciju koja bi odredila optimalni način da se to uradi. Funkcija stavlja volumen rigovanja meša u drugi plan da bi računica bila efikasnija, dok je finalno postavljanje skeleta određeno metodom optimizacije optimalnih margina.

Da bi se ugradio skelet na optimalan način, prvo se izoluje postojeći volumen, tako da se ne uzima ceo prostor unutar meša kao mogućnost postavljanja skeleta. Na kraju ovog koraka pronađena je medijalna, srednja površina meša. Srednja ili medijalna površina je isto što i medijalna osa, samo u 3D prostoru. Tačka pripada medijalnoj osi ukoliko ima više od jedne najbliže tačke na površini modela. Logično, medijalna površina se sastoji od tačaka koje centrirane u modelu, gde bi se očekivala skeletalna struktura. Slika 1. skroz levo prikazuje medialnu površinu. Idealno bi bilo da se nađe prostor koji bi bio odvojeniji od medijalne površine. Zbog toga je iskorišćena metoda za kreiranje konačnog grafa vrlo slična pakovanju sfera. Svaka tačka medijalne površine, počevši od najdalje od meša, dodaje sferu gde je centar sfere ta tačka, a radijus najbliža udaljenost do meša. Sfera se ne dodaje ukoliko sadrži centar neke druge sfere. Upakovane sfere unutar meša su prikazane na slici 1., u sredini. Finalni graf je onda konstruisan, centri sfera postaju čvorovi, a grane su postavljene između preklapajućih sfera, ili ukoliko je ta grana esencijalna za dalji nastavak. Svaki čvor je potencijalna pozicija zgloba skeleta, a svaka grana je potencijalna konekcija, kost izmedju dva zgloba.



Slika 1. Izolovanje postojećeg volumena unutrašnjosti meša. Levo je prikazana medialna površina, sredina prikazuje pakovanje sfera, desno je prikazan graf. [1]

Nakon ovog procesa, potrebno je nekako „ugurati“ skelet u dobijeni graf. Da bi se našao najbolji način potrebno je na neki način negativno bodovati potencijalne zglobove i kosti. Mogući opisi nepoželjnih delova skeleta bi bile kratke kosti, neprimerena orientacija među zglobovima, dužina kosti koje su označene kao simetrične, kosti koje dele više od jednog zgloba (u skeletu ne bi trebalo da bude petlji), kost označena kao „noge“ a udaljena od poda, neprimerena orientacija kosti, kao i zglobovi preblizu jedan drugome, a udaljeni od skeleta[1]. Da bi se našao optimalni način negativnog bodovanja, uči se maksimalni marginalni linearni klasifikator, koji pokušava da maksimizuje margine između najboljeg dobrog i najboljeg lošeg dela skeleta. Formula 1 predstavlja maksimalni marginalni linearni klasifikator. [1]

(1)

Gde je **p** i **q** vektor sa bodovima (sastoji se od negativnih bodova koji su malo pre spomenuti) dobrih i loših delova respektivno. Г je vektor koji sadrži težinu, a **n** i **m** su brojevi dobrih i loših delova skeleta, respektivno. Idealno bi bilo da izbor Г bude takav da je jednačina 1 maksimizovana, tako da se napravi očigledna razdvojenost dobrih i loših potencijalnih delova skeleta.

Sama procedura učenja se zasniva na Nedler-Mid metodi. Započinje se sa nasumičnom vrednošću za težine, koje su ocenjivane jednačinom 1, što bi trebalo da se maksimizuje na setu podataka za treniranje. Ovaj deo se radi nekoliko puta sa različitim početnim težinama, jer metoda Nedler-Mid može da se zaustavi (zaglavi) u lokalnom minimumu.

Sada je moguće izračunati optimalni način pripajanja skeleta u meš. Da bi se prostor potrage smanjio još više, zglobovi koji povezuju dve kosti (kao što su kolena i laktovi) se uklanjaju iz šablonskog skeleta. Rezultat svakog parcijalnog potencijalnog pripajanja skeleta mešu je suma negativnih bodova između dva međusobno spojena zgloba dodata na negativne bodove prva dva zgloba dodata na negativne bodove na prva tri zgloba i tako nadalje sve do poslednjeg pripojenog zgloba. Algoritam staje u slučaju da je pripajanje završeno, i na taj način je osigurano da je izabrano najoptimalnije rešenje. Kada je nađeno konačno rešenje, spojeni zglobovi se ponovo dodaju tako što se razdvoje na nađene ivice u proporcije odgovarajuće originalnom skeletu. Na slici 2 prikazan je tok algoritma „Pinocchio“ programske biblioteke.

Slika 2. Tok algoritma „Pinocchio“ programske biblioteke

**2.2. „Skining“ modela**

„Skining“ modela je proces prijanjanja teksture na 3D model i definiše na koji način bi se geometrijska struktura menjala prilikom promene poze skeleta. Ovaj proces može biti jedan od zahtevnijih procesa prilikom kreiranja 3D avatara, zato što je potrebno da se kompleksna 3D površina pretoči u 2D sliku (teksturu) tako da se svaka tačka mapira.

„Skining“ modela u „Pinocchio“ programskoj biblioteci je odrađen korišćenjem tehnike zvanom LBS (Linear Blend Skinning [1]), jer je jedna od najrasprostranjenijih metoda u ovoj oblasti. LBS kao ulazne podatke očekuje oblik meš modela u standardnoj pozi, transformaciju kosti i težinu „skining“-a [2]. Meš model u standardoj, odnosno početnoj, pozi predstavljen je nizom trodimenzionalnih tačaka. Transformacija kosti je niz transformacionih matrica koje opisuju način pokretanja skeleta. Težina „skinning“-a predstavlja jednodimenzionalni niz težina koji odgovara odgovarajućoj tački u meš modelu. Težina opisuje koliko kost utiče na neku tačku. Ukoliko je **vj** pozicija tačke ***j****,*  **T i** je transformacija ***i-te*** koske, a ***w ij*** je težina ***i-te*** koske za tačku ***j*,** LBS nam daje poziciju transformisane tačke ***j***  kao ***∑i w ij T i (vj).*** Na slici 3 prikazan je meš model u prvobitnom stanju. Na slici 4 je prikazan isti taj meš model kako se kreće. Kao što se vidi na laktu desne ruke došlo je do deformacije meš modela, način na koji se meš transformiše određen je LBS tehnikom.

LBS funkcioniše vrlo dobro u slučajevima kada se transformacije **T i** ne razlikuju mnogo. Problemi se javljaju ukoliko je potrebno iskoristiti transformacije koje se značajno razlikuju u rotacionoj komponenti. Poznato je da linearna kombinacija rotacija više nije rotacija[3].

|  |  |
| --- | --- |
| Slika 3. Meš model u stanju mirovanja (početnom) | Slika 4. Meš model koji se kreće |

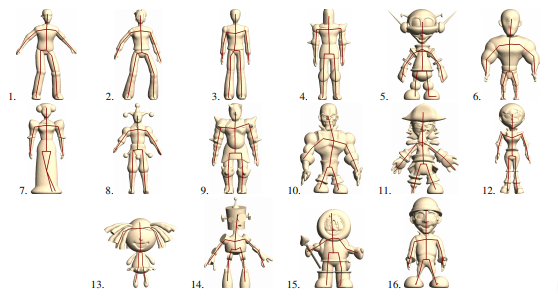
**3. Upotrebljivost „Pinocchio“ programske biblioteke**

Upotrebljivost „Pinocchio“ programske biblioteke je ocenjena na proceni preformansi, samostalnosti i uspešnosti. Pod **preformansom** biblioteke smatra se brzina koja je potrebna da bi se prošao ceo proces autorigovanja. Idealno, proces bi trebao da bude brži od ručnog autorigovanja. Algoritam koji je korišćen u biblioteci, testiran je na 16 različitih modela. Najkraće potrebno vreme bilo je 12.6 sekundi za model koji sadrži 19,001 tačku. Najduže vreme je 77.1 sekundu za model od 56,865 tačaka[1]. Najviše vremena oduzima pretprocesiranje, zbog kalkulacije polja distanci. Samo „ugrađivanje“ traje od prilike jednu petinu celoukupnog vremena. Rigovanje meš modela trenutno oduzima dosta vremena, čak i osobama koje to rade dugo i bave se time profesionalno. Vreme potrebno za rigovanje prostih karaktera i meš modela je 2-3 sata. Za srednje kompleksne karaktere i meš modele vreme koje je potrebno za rigovanje je 1 – 1.5 dana. Dok za najkompleksnije, koji zahtevaju, mimiku lica, savijanje prsta, utrošeno vreme može biti i preko mesec dana.[4]

**Samostalnost** bi idealno trebala da bude što veća, jer na taj način konačni korisnik bi trebao uz minimum dodatnog truda da popravi greške algoritma. Restrikcije ili dodatna uputstva kroz proces rigovanja trebao bi biti minimalan, jer na taj način korisnik bi mogao da izvrši širok spektar mogućnosti uz minimalan broj koraka. Restrikcije kod biblioteke su vrlo velike, jer zahteva početni skelet, koji mora da se slaže sa modelom. Potrebno je da se model dosta poklapa sa nekim od šablon skeleta. Ukoliko bi ovo bilo drugačije broj negativnih bodova koji se dodeljuju kostima i zglobovima bi bio ogroman. Takođe, biblioteka očekuje da obe noge budu na dnu modela. Na slici 4 prikazan je meš model koji je orientisan shodno tome što očekuje „Pinocchio“ programska biblioteka. Kao što se vidi iz priloženog, model je vrlo dobro izrigovan. Svaki zglob je na dobrom mestu. Slika 6 prikazuje isti meš model kao i na slici 5, samo što je u ovom slučaju meš model zarotiran za 90 stepeni oko svoje *z* ose. Pošto programska biblioteka „Pinocchio“ očekuje da noge modela budu najbliže podu, ona je u desnu ruku meš modela smestila desnu nogu skeleta, u desnoj nozi meš modela smestila je levu nogu skeleta, u levoj nozi meš modela smestila je levu ruku skeleta, u levoj ruci meš modela smestila je glavu skeleta i u glavi meš modela smestila je desnu ruku skeleta. Kao što se vidi na slici, biblioteka ne može da detektuje rotaciju modela pa da shodno njoj rotira i skelet.

|  |  |
| --- | --- |
| Slika 5. Meš model u validnoj rotaciji i rezultat njegovog rigovanja | Slika 6. Meš model rotiran za 90 stepeni oko njegove *z* ose i rezultat njegovog rigovanja |

**Uopštenost** metode je vrlo visoka. Idealno bi bilo da je moguće primeniti način rigovanja na bilo koji model. „Pinocchio“ je testiran nad 16 modela koji su prikazani na slici 7. Većina modela su tačno rigovana, sem modela 7, 10 i 13. Ali njihova ispravka bi bila vrlo jednostavna, korisnik bi trebao samo da pokaže još jedan dodatni zglob. Biblioteka „Pinochio“ funkcioniše vrlo dobro u ovom pogledu, ali je vrlo ograničena na čovekolike modele. Ovaj problem bi mogao da se reši na način da se unos šablon skeleta poveća sa jednog na više, samim tim, skelet koji dobije najmanje negativnih ocena bio bi vraćen kao optimalni rezultat. Ovaj način nadograđivanja biblioteke bi, naravno, imao svoje mane. Prvo, problem bi predstavljalo vreme izvršavanja. Svako dodatni šablon skelet bi značajno uvećao vreme izvršenje algoritma, iako je rečeno da sam proces prilagođavanja skeleta oduzima samo petinu vremena celog procesa, ali u najgorem slučaju i taj proces bi mogao da se oduži i značajno da utiče na preformanse algoritma. Drugi problem je trenutna funkcija koja boduje delove skeleta. Iako je trenutna funkcija jednostavna i upotrebljiva nad različitim šablonima skeleta, bodovanje malih kosti negativno utiče na loše prilagođavanje skeleta kada su u pitanju modeli kao što su životinje sa repovima, zmije, oktopodi, ili bilo koji drugi karakter koji se sastoji od kratkih kosti. Naravno, ovo je rešivo ignorisanjem dužine prilikom traženja kosti koje su označene kao *rep*. Poslednji i najveći problem koji je nerešiv zbog same metode i načina na koji funkcioniše, je taj da šta ukoliko je najbolji mogući šablon skelet ipak nedovoljno dobar za model? Kao što je i rečeno, sama metoda „ugrađivanja“ skeleta je ograničena na šablon skelet koji mu se daje kao ulaz, i samim tim prilikom korišćenja ove biblioteke potreban je dodatni napor krajnjeg korisnika koji bi ispravio nedostatke.

Slika 7. Rezultati rigovanja 16 modela korišćenjem biblioteke.

**4. Korišćenje i proširivanje „Pinocchio“ programske biblioteke**

Programska biblioteka „Pinocchio“ je slobodna za korišćenje (open source). Ceo kod je slobodan i moguć za modifikaciju. Aplilkacija koja dolazi uz programsku biblioteku „Pinocchio“ služi za prikaz rezultata meš modela nakon rigovanja zadatim parametrima. Aplikaciju je moguće izbildovati isključivo ukoliko na računaru ima prethodno instalirana FLTK biblioteka [5] verzije 1.1.x ili starije. Bildovanje aplikacije kod Linux i ostalih Unix sistema potrebno je samo izvršiti naredbu „make“ nad „makefiles“. Dok kod Windows operativnog sistema potrebno je definisati FLTKDIR globalnu Windows variablu (environment variable) da bi program „video“ fajlove zaglavlja (header files). Zatim je potrebno pokrenuti Visual Studio 2005 ili neki stariji uz pomoć kojeg bi se projekti izbildovali. Zarad ovog seminarskog rada aplikacija je izbildovana u Visual Studio 2015. Pošto je programska biblioteka „Pinocchio“ razvijena 2007 godine, i to u Visual Studio 2005, ona unutar svog source koda koristi tipove podataka koji su zastareli za novije verzije Visual Studia. Zastarele tipove podataka koje koristi su <hash\_map> i <hash\_set>. Trenutna alternativa ovim podacima su <unordered\_map> i <unordered\_set>. Da bi se ovo automatski primenilo potrebno je u svakom fajlu gde se navedeni zastareli tipovi podataka koriste definiše #define \_SILENCE\_STDEXT\_HASH\_DEPRECATION\_WARNINGS 1 , time se izbegavaju potencijalne greške prilikom bildovanja projekta.

Programska biblioteka „Pinocchio“ sadrži dosta metoda koje je moguće vrlo dobro iskoristiti u oblasti autorigovanja. Da bi se biblioteka upotrebila na najprostili način, moguće je izvršiti u četiri koraka koraka. Kod 1 prikazuje najprostiji mogući način da se izvrši rigovanje nekog modela. Glavni deo koda je funkcija PinocchioOutput autorig(const Skeleton &s, const Mesh &m). Ova funkcija sadrži sve korake koje je potrebno izvršiti da bi se model izrigovao. Funkcija kao ulazne parametere uzima Mesh &m , što je model koji je potrebno izrigovati. Drugi parametar je Skeleton &s koji predstavlja tip skeleta koji će model poprimiti na kraju. Klasa Skeleton predstavlja apstraktnu klasu i treba da bude ulazni skelet koji će se pripojiti modelu. Tipovi skeleta koji su definisani u biblioteci i mogu se iskoristiti kao ulazni skelet su: „CentaurSkeleton“, „FileSkeleton“, „HorseSkeleton“, „HumanSkeleton“ i „QuadSkeleton“.

|  |
| --- |
| #include<stdlib.h>  #include"pinocchioApi.h"    void main(int argc, char\*\* argv)  {  Skeletons = HumanSkeleton();  Mesh m = Mesh("model\_path");  autorig(s, m);  } |
| Kod 1. Najjednostavniji način koišćenja biblioteke „Pinocchio“ |

Funkcija „autorig“ predstavlja kombinaciju svih koraka koji su opisani u poglavlju 2.1. Sve funkcije koje su iskorišćene unutar „autorig“ funkcije moguće je izvršiti pojedinačno , sve to dolazi zajedno sa bibliotekom „Pinocchio“. Kod 2 predstavlja funkciju „autorig“ i koraci kojima se ona izvršava.

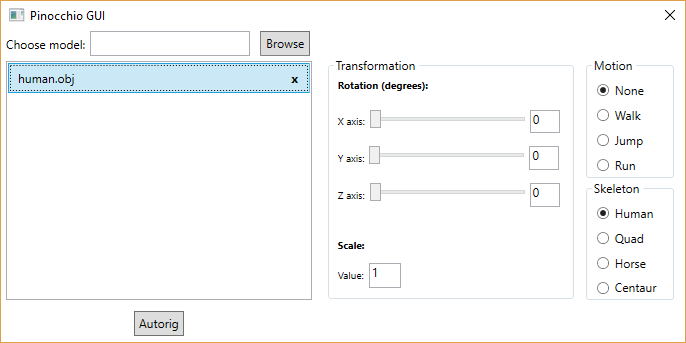
|  |
| --- |
| PinocchioOutput autorig(const Skeleton& given,const Mesh &m)  {  int i;  PinocchioOutput out;  Mesh newMesh= prepareMesh(m);  if(newMesh.vertices.size() == 0)  return out;  //kreiranje polja distanci mes modela  TreeType\* distanceField= constructDistanceField(newMesh);  //kreiranje medijalne povrsine  vector<Sphere> medialSurface=sampleMedialSurface(distanceField);  //pakovanje sfera  vector<Sphere> spheres=packSpheres(medialSurface);  //spajanje dobijenih sfera i distance  PtGraph graph=connectSamples(distanceField,spheres);  vector<vector<int>>possibilities = computePossibilities(graph, spheres, given);  vector<int> embeddingIndices = discreteEmbed(graph, spheres, given, possibilities);  if(embeddingIndices.size() == 0)  {  //greska  delete distanceField;  return out;  }  //razdvajanje  vector<Vector3> discreteEmbedding = splitPaths(embeddingIndices, graph, given);  //poboljsanja povrsine  vector<Vector3> medialCenters(medialSurface.size());  for(i=0; i<(int)medialSurface.size(); ++i)  medialCenters[i] = medialSurface[i].center;  out.embedding = refineEmbedding(distanceField, medialCenters, discreteEmbedding, given);  //attachment  VisTester<TreeType>\* tester = newVisTester<TreeType>(distanceField);  out.attachment = newAttachment(newMesh, given, out.embedding, tester);  //ciscenje resursa  delete tester;  delete distanceField;  return out;  } |
| Kod 2. Autoriging funkcija |

Trenutno pokretanje programske biblioteke se vrši iz komandne linije očekuje jedan uslovni argument, a to je putanja do meš modela koji je potrebno izrigovati. Pored uslovnog argumenta, putanje do modela, moguće je dodati i opcione argumente. Opcioni argumenti mogu biti rotacija meš modela za navedene stepene, skaliranje njihovog skeleta nakon rigovanja, ulazni skelet kao i pokrete koje bi avatar primenio nakon završetka procesa rigovanja. Primer pokretanja programske biblioteke sa uslovnim i opcionim argumentima prikazan je u komandnoj liniji 1. Putanja do meš modela „**data\cheb.obj“**  je prvi i uslovni argument do meš modela. Rotacija meš modela se menja dodavanjem argumenta **-rot 0 1 0 90** , u ovom slučaju meš model će biti zarotiran oko *y* ose za 90 stepeni. Ukoliko se ne navede ulazni skelet je uvek čovekoliki, ali moguće je navesti neki drugi ukoliko je potrebno izrigovati neki model koji nema telesnu strukturu čoveka. Na primer, meš model neke četvoronoge životinje bi kao ulazni skelet zahtevao skelet koji je naveden u komandoj liniji jedan, **-skel quad.**  Ukoliko je potrebno skaliranje konačnok skeleta dodaje se argument **-scale 2** , koji u ovom slučaju duplira veličinu konačnog skeleta. Poslednji opcioni argument naveden u komandnoj liniji 1 je dodavanje pokreta krajnje izrigovanom avataru, **-motion motion\jumpAround.txt**. U ovom slučaju, avatar bi skakutao.

**DemoUI.exe data\cheb.obj -rot 0 1 0 90 -skel quad -scale 2** **-motion motion\jumpAround.txt**

Komandna linija 1. Primer pokretanja programske biblioteke sa uslovnim i opcionim argumentima

Aplikacja koja je kreirana radi lakšeg korišćenja programske biblioteke „Pinocchio“ i predstavlja jednostavan i user-friendly interfejs kako bi prosečan korisnik mogao da je upotrebi. Na slici 8 prikazana je aplikacija, dok kod 3 prikazuje xaml fajl u kome je interfejs aplikacije napisan. Sadrži osnovne parametre koje su potrebne korisniku da bi započeo rigovanje nekog meš modela. Koisnik je u mogućnosti da odabere meš model koji bi želeo da izriguje na dva načina, jedan je ukucavanjem pune adrese do modela i pritiskom „Enter“ tastera ili klikom „Browse“ dugmeta koje bi mu otvorilo prozor za odabir jednog ili više meš modela. Zatim, korisnik odabirom meš modela iz liste može da modifikuje parametre svakog meš modela zasebno. Meš model je moguće transformisati pomoću jedne od dve transformacije, a to su rotacija i skaliranje. Rotaciju je moguće izvršiti nad sve tri ose modela. Dok se skaliranje vrši nad autorigovanim skeletom, a ne nad modelom.



Slika 8. Interfejs aplikacija za autorigovanje modela pomoću Pinocchio biblioteke

|  |
| --- |
| <Window x:Class="PinocchioInterface.MainWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:PinocchioInterface"  xmlns:controls ="clr-namespace:PinocchioInterface.Controls"  xmlns:viewmodel="clr-namespace:PinocchioInterface.ViewModel"  mc:Ignorable="d"  Title="Pinocchio GUI" Height="350" Width="650"  ResizeMode="NoResize">  <Window.DataContext>  <viewmodel:MainWindowViewModel/>  </Window.DataContext>  <Window.Resources>  <controls:EnumBooleanConverter x:Key="EnumBooleanConverter"/>  </Window.Resources>  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="1\*"/>  <RowDefinition Height="10\*"/>  <RowDefinition Height="1.5\*"/>  </Grid.RowDefinitions>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="3\*"/>  <ColumnDefinition Width="2\*"/>  <ColumnDefinition Width="1\*"/>  </Grid.ColumnDefinitions>  <StackPanel Orientation="Horizontal" HorizontalAlignment="Left" VerticalAlignment="Center">  <Label Content="Choose model:"/>  <TextBox x:Name="tbModelPath" ToolTip="{Binding RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}" Width="160" KeyUp="tbModelPath\_KeyUp">  <TextBox.Text>  <Binding Path="ModelPath" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged">  <Binding.ValidationRules>  <local:FileExistsValidator/>  </Binding.ValidationRules>  </Binding>  </TextBox.Text>  </TextBox>  <Button Margin="10 0 0 0" Name="btnBrowse" Click="btnBrowse\_Click" Width ="50" Height="25" Content="Browse" />  </StackPanel>  <ListBox x:Name="lbModels" Grid.Row="1" Margin="5" HorizontalContentAlignment="Stretch" ItemsSource="{Binding RiggingModels}">  <ListBox.ItemTemplate>  <DataTemplate>  <Grid>  <Grid.ColumnDefinitions>  <ColumnDefinition Width="270"/>  <ColumnDefinition Width="20"/>  </Grid.ColumnDefinitions>  <Label Grid.Column="0" ToolTip="{Binding Path=Path}">  <TextBlock Text="{Binding Path=Name}"/>  </Label>  <Label Grid.Column="1" MouseLeftButtonUp="Label\_MouseLeftButtonUp" Content="x" FontWeight="Bold" HorizontalAlignment="Right" ToolTip="Remove from the list"/>  </Grid>  </DataTemplate>  </ListBox.ItemTemplate>  </ListBox>  <GroupBox Grid.Row="1" Grid.Column="1" Header="Transformation" Margin="10 0 5 5">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="1\*"/>  <RowDefinition Height="1\*"/>  <RowDefinition Height="1\*"/>  <RowDefinition Height="1\*"/>  <RowDefinition Height="1\*"/>  <RowDefinition Height="1\*"/>  </Grid.RowDefinitions>  <Label Grid.Row="0" Content="Rotation (degrees):" FontSize="10" FontWeight="Bold"></Label>  <StackPanel VerticalAlignment="Top" Grid.Row="1" Orientation="Horizontal">  <Label FontSize="10" Content="X axis:"/>  <Slider Maximum="359" Minimum="0" Width="160" Value="{Binding ElementName=lbModels, Path=SelectedItem.XRot, Mode=TwoWay}"/>  <TextBox ToolTip="{Binding RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}" Width="30" x:Name="tbXrot">  <TextBox.Text>  <Binding ElementName="lbModels" Path="SelectedItem.XRot" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged">  <Binding.ValidationRules>  <local:DegreesValidator/>  </Binding.ValidationRules>  </Binding>  </TextBox.Text>  </TextBox>  </StackPanel>  <StackPanel VerticalAlignment="Top" Grid.Row="2" Orientation="Horizontal">  <Label FontSize="10" Content="Y axis:"/>  <Slider Maximum="359" Minimum="0" Width="160" Value="{Binding ElementName=lbModels, Path=SelectedItem.YRot, Mode=TwoWay}"/>  <TextBox ToolTip="{Binding RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}" Width="30" x:Name="tbYrot">  <TextBox.Text>  <Binding ElementName="lbModels" Path="SelectedItem.YRot" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged">  <Binding.ValidationRules>  <local:DegreesValidator/>  </Binding.ValidationRules>  </Binding>  </TextBox.Text>  </TextBox>  </StackPanel>  <StackPanel VerticalAlignment="Top" Grid.Row="3" Orientation="Horizontal">  <Label FontSize="10" Content="Z axis:" />  <Slider Maximum="359" Minimum="0" Width="160" Value="{Binding ElementName=lbModels, Path=SelectedItem.ZRot, Mode=TwoWay}"/>  <TextBox ToolTip="{Binding RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}" Width="30" x:Name="tbZrot" >  <TextBox.Text>  <Binding ElementName="lbModels" Path="SelectedItem.ZRot" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged">  <Binding.ValidationRules>  <local:DegreesValidator/>  </Binding.ValidationRules>  </Binding>  </TextBox.Text>  </TextBox>  </StackPanel>  <Label VerticalAlignment="Bottom" Grid.Row="4" Content="Scale:" FontWeight="Bold" FontSize="10"></Label>  <StackPanel Grid.Row="5" Orientation="Horizontal">  <Label FontSize="10" VerticalAlignment="Center" Content="Value:"/>  <TextBox ToolTip="{Binding RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}" Width="32" Height="25" x:Name="tbScaleFactor">  <TextBox.Text>  <Binding ElementName="lbModels" Path="SelectedItem.ScaleFactor" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged">  <Binding.ValidationRules>  <local:ScaleFactorValidator/>  </Binding.ValidationRules>  </Binding>  </TextBox.Text>  </TextBox>  </StackPanel>  </Grid>  </GroupBox>  <StackPanel Margin="5 0 10 0" Grid.Row="1" Grid.Column="2" Orientation="Vertical">  <GroupBox x:Name="gbMotion" Header="Motion">  <GroupBox.Style>  <Style TargetType="GroupBox">  <Style.Triggers>  <DataTrigger Binding="{Binding ElementName=lbModels, Path=SelectedIndex}" Value="-1">  <Setter Property="IsEnabled" Value="False"/>  </DataTrigger>  </Style.Triggers>  </Style>  </GroupBox.Style>  <StackPanel Orientation="Vertical">  <RadioButton GroupName="MotionGroup" x:Name="rbNone" Content="None" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Motion, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Motion.None}}" />  <RadioButton GroupName="MotionGroup" x:Name="rbWalk" Content="Walk" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Motion, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Motion.Walk}}"/>  <RadioButton GroupName="MotionGroup" x:Name="rbJump" Content="Jump" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Motion, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Motion.Jump}}"/>  <RadioButton GroupName="MotionGroup" x:Name="rbRun" Content="Run" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Motion, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Motion.Run}}"/>  </StackPanel>  </GroupBox>  <GroupBox x:Name="gbSkeleton" Margin="0 0 0 5" Header="Skeleton">  <GroupBox.Style>  <Style TargetType="GroupBox">  <Style.Triggers>  <DataTrigger Binding="{Binding ElementName=lbModels, Path=SelectedIndex}" Value="-1">  <Setter Property="IsEnabled" Value="False"/>  </DataTrigger>  </Style.Triggers>  </Style>  </GroupBox.Style>  <StackPanel Orientation="Vertical">  <RadioButton GroupName="SkeletonGroup" x:Name="rbHuman" Content="Human" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Skeleton, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Skeleton.Human}, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>  <RadioButton GroupName="SkeletonGroup" x:Name="rbQuad" Content="Quad" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Skeleton, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Skeleton.Quad}, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}">  <RadioButton.Style>  <Style TargetType="RadioButton">  <Style.Triggers>  <DataTrigger Binding="{Binding ElementName=rbNone, Path=IsChecked}" Value="False">  <Setter Property="IsEnabled" Value="False"/>    </DataTrigger>  </Style.Triggers>  </Style>  </RadioButton.Style>  </RadioButton>  <RadioButton GroupName="SkeletonGroup" x:Name="rbHorse" Content="Horse" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Skeleton, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Skeleton.Horse}, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" >  <RadioButton.Style>  <Style TargetType="RadioButton">  <Style.Triggers>  <DataTrigger Binding="{Binding ElementName=rbNone, Path=IsChecked}" Value="False">  <Setter Property="IsEnabled" Value="False"/>  </DataTrigger>  </Style.Triggers>  </Style>  </RadioButton.Style>  </RadioButton>  <RadioButton GroupName="SkeletonGroup" x:Name="rbCentaur" Content="Centaur" Margin="5 10 0 0" IsChecked="{Binding ElementName=lbModels, Path=SelectedItem.Skeleton, Converter={StaticResource EnumBooleanConverter}, ConverterParameter={x:Static local:Skeleton.Centaur}, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" >  <RadioButton.Style>  <Style TargetType="RadioButton">  <Style.Triggers>  <DataTrigger Binding="{Binding ElementName=rbNone, Path=IsChecked}" Value="False">  <Setter Property="IsEnabled" Value="False"/>    </DataTrigger>  </Style.Triggers>  </Style>  </RadioButton.Style>  </RadioButton>  </StackPanel>  </GroupBox>  </StackPanel>    <Button Grid.Row="3" Name="btnAutorig" Click="btnAutorig\_Click" Width ="50" Height="25" Content="Autorig" VerticalAlignment="Center" HorizontalAlignment="Center">  <Button.Style>  <Style TargetType="Button">  <Setter Property="IsEnabled" Value="False"/>  <Style.Triggers>  <MultiDataTrigger>  <MultiDataTrigger.Conditions>  <Condition Binding="{Binding Path=(Validation.HasError), ElementName=tbXrot}" Value="False"/>  <Condition Binding="{Binding Path=(Validation.HasError), ElementName=tbYrot}" Value="False"/>  <Condition Binding="{Binding Path=(Validation.HasError), ElementName=tbZrot}" Value="False"/>  <Condition Binding="{Binding Path=(Validation.HasError), ElementName=tbScaleFactor}" Value="False"/>  </MultiDataTrigger.Conditions>  <Setter Property="IsEnabled" Value="True"/>  </MultiDataTrigger>  <DataTrigger Binding="{Binding ElementName=lbModels, Path=Items.Count}" Value="0">  <Setter Property="IsEnabled" Value="False"/>  </DataTrigger>  </Style.Triggers>  </Style>  </Button.Style>  </Button>  </Grid>  </Window> |
| Kod 3. Interfejs aplikacija za autorigovanje modela pomoću Pinocchio biblioteke |

Korisnik može da odabere i ulazni skelet rigovanja, zavisno od meš modela. Takođe korisnik ima opciju da odabere i pokret svog lika nakon rigovanja. Pokreti su trenutno mogući isključivo za meš modele za koje ulazni skelet ljudskog oblika. Naravno, ovo nije potrebno, nego bi samo demonstriralo pokretanje modela sa prethodno definisanim pokretima. Trenutno su navedena tri: skakanje, hodanje i trčanje. Nakon pritiska dugmeta „Autorig“ počinje proces autorigovanja koji će kao rezultat prikazati željene modele. Proces autorigovanja nije dug, ne traje duže od minut, a i modeli su dosta dobro rigovani.

Prilikom odabira putanje kreira se novi objekat klase RiggingModel . Klasa RiggingModel sadrži sve potrebne informacije vezane za učitani meš model. Svaka promena na interfejsu na prethodno opisan način reflektuje promenu u odgovarajućem RiggingModel objektu. Kod 4 prikazuje celu klasu RiggingModel i parametre koje ona sadrži.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace PinocchioInterface  {  public class RiggingModel : INotifyPropertyChanged  {  private int \_xRot;  private int \_yRot;  private int \_zRot;  private double \_scaleFactor;  private string \_path;  private string \_name;  private Motion \_motion;  private Skeleton \_skeleton;  public RiggingModel(string path)  {  XRot = 0;  YRot = 0;  ZRot = 0;  ScaleFactor = 1;  Path = path;  Motion = Motion.None;  Skeleton = Skeleton.Human;  }  /// <summary>  /// Degrees value for rotation aroun X axis.  /// </summary>  public int XRot  {  get { return \_xRot; }  set  {  \_xRot = value;  NotifyPropertyChanged("XRot");  }  }  /// <summary>  /// Degrees value for rotation aroun Y axis.  /// </summary>  public int YRot  {  get { return \_yRot; }  set  {  \_yRot = value;  NotifyPropertyChanged("YRot");  }  }  /// <summary>  /// Degrees value for rotation aroun Z axis.  /// </summary>  public int ZRot  {  get { return \_zRot; }  set  {  \_zRot = value;  NotifyPropertyChanged("ZRot");  }  }  /// <summary>  /// Skeleton scaling factor  /// </summary>  public double ScaleFactor  {  get { return \_scaleFactor; }  set  {  \_scaleFactor = Math.Round(value, 2);  }  }  /// <summary>  /// Path to the model file  /// </summary>  public string Path  {  get { return \_path; }  set  {  \_path = value;  Name = System.IO.Path.GetFileName(Path);  }  }  /// <summary>  /// Name of the model file, extracted from Path property  /// </summary>  public string Name  {  get  {  return \_name;  }  set  {  \_name = value;  }  }    /// <summary>  /// Motion  /// </summary>  public Motion Motion  {  get  {  return \_motion;  }  set  {  \_motion = value;  if (\_motion != Motion.None)  Skeleton = Skeleton.Human;  }  }  /// <summary>  /// Skeleton  /// </summary>  public Skeleton Skeleton  {  get { return \_skeleton; }  set { \_skeleton = value; NotifyPropertyChanged("Skeleton"); }  }  public string GetCommandLineArguments(string motionFolder)  {  string[] parameters = new string[7];  parameters[0] = GetPathForCmd();  parameters[1] = GetXRotForCmd();  parameters[2] = GetYRotForCmd();  parameters[3] = GetZRotForCmd();  parameters[4] = GetScaleFactorCmd();  parameters[5] = GetMotionForCmd(motionFolder);  parameters[6] = GetSkeletonCmd();  return String.Join(" ", parameters);  }  private string GetMotionForCmd(string motionFolder)  {  switch (Motion)  {  case Motion.Jump:  return "-motion " + System.IO.Path.Combine(motionFolder, "jumpAround.txt");  case Motion.Walk:  return "-motion " + System.IO.Path.Combine(motionFolder, "walk.txt");  case Motion.Run:  return "-motion " + System.IO.Path.Combine(motionFolder, "runAround.txt");  }  return "";  }  private string GetSkeletonCmd()  {  return "-skel " + Skeleton.ToString().ToLower();  }  private string GetScaleFactorCmd()  {  return "-scale " + ScaleFactor;  }  private string GetZRotForCmd()  {  return "-rot 0 0 1 " + ZRot;  }  private string GetYRotForCmd()  {  return "-rot 0 1 0 " + YRot;  }  private string GetXRotForCmd()  {  return "-rot 1 0 0 " + XRot;  }  private string GetPathForCmd()  {  return "\"" + Path + "\"";  }  public event PropertyChangedEventHandler PropertyChanged;  private void NotifyPropertyChanged(string propertyName = "")  {  if (PropertyChanged != null)  {  PropertyChanged(this, new PropertyChangedEventArgs(propertyName));  }  }  }  public enum Motion  {  None,  Walk,  Run,  Jump  }  public enum Skeleton  {  Human,  Quad,  Horse,  Centaur  }  } |
| Kod 3. RiggingModel klasa |

**5. Zaključak**

U ovom seminarskom opisana je jedna od starijih programskih biblioteka za automatsko rigovanje meš modela koja se zasniva na ugrađivanju već predefinisanog skeleta unutar meš modela. Prvo je prikazan način na koji funkcioniše biblioteka i kakve algoritme koristi. Zatim je ocenjena na tri prethodno definisana aspekta, preformanse, uopštenost i samostalnost. Potom je opisan i dat primer njene primene i opisani zaključci do kojih se došlo.

„Pinocchio“ programska biblioteka u stanju kakvom je trenutno je vrlo nezahvalna za rad, pogotovo za korisnika sa prosečnim znanjem računara. Da bi se skelet ugradio korisnik mora da zada startne podatke, to nije poželjno, jer uvek postoji mogućnost korisničke greške. Takođe, model koji se prosleđuje kao ulazni parametar potrebno je da bude orijentisan uspravno, jer u slučaju da je pogrešno rotiran došlo bi do greške u rigovanju. Preformanse koje je programska biblioteka prikazala su vrlo dobre i dosta skraćuje mukotrpan proces ručnog rigovanja meš modela. Sama biblioteka i proces njene primene je zahtevan iz razloga što je dosta stara i neke od metoda i tipova podataka se više ne koriste, pa je zbog toga potrebno izmeniti kod i ponovo izbildovati. Ipak, kada se prevaziđu te prepreke, korišćenje i implementacija koda je relativno laka.

Programsku biblioteku je moguće primeniti u oblastima u kojima je potrebno rigovanje jednog tipa meš modela. Primena u industriji koja sadrži isključivo čovekolike meš modele je vrlo moguća i uz minimalne modifikacije dala bi značajne rezultate. U tom slučaju bi ulazni skelet model uvek bio isti. Aplikacija koja je kreirana zarad ovog seminarskog rada olakšala je korišćenje programske biblioteke. Korisnik sa prosečnim znanjem računara bi sada mogao nesmetano da koristi programsku biblioteku i riguje za koje postoji već predifinisani skelet.

**6. Literatura**

[1] Ilya Baran and Jovan Popovi´c. Automatic rigging and animation of 3d characters. ACM Trans. Graph., 26(3), July 2007.

[2] Ladislav Kavan Part I: Direct Skinning Methods and Deformation Primitives SIGGRAPH Course 2014 — Skinning: Real-time Shape Deformation

[3] ALEXA, M., AND MÜLLER, W. 2000. Representing animations by principal components. In Computer Graphics Forum, vol. 19, Wiley Online Library, 411–418.

[4] <http://forums.cgsociety.org/archive/index.php?t-929284.html>

[5] http://www.fltk.org/index.php