Kinect Driven 2D-Mesh Animation

Sarthak Ahuja*
2012088 - B.Tech Computer Science, IIIT-Delhi

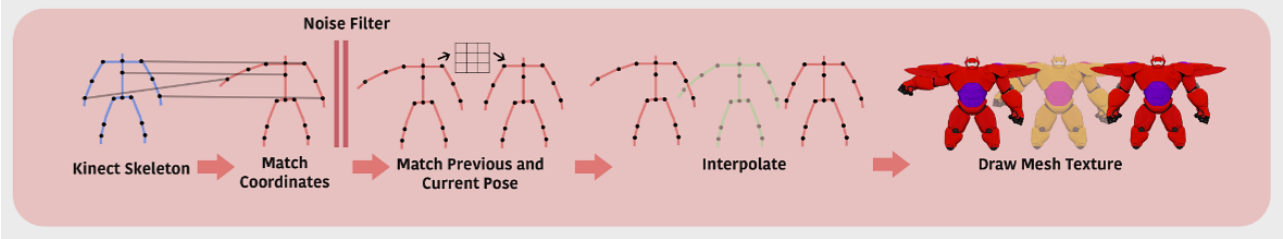


Figure 1: *An overview of the algorithm implemented*

Abstract

In this project we focus on animating a 2D mesh character and driving its actions through kinect. It has been a little more than five years since the first kinect was launched and even today it continues to draw a lot of attention and applications in various fields. In this approach we use Visual Studio to interact with our kinect device and fetch the skeleton data captured in the form of coordinates. We then create our 2D mesh in QT and use these coordinates to mimic the skeleton. The project follows a bone based skeleton model to construct our 2D mesh character. This projects finds applications in the animation industry for simplifying a lot of processes involved in animating 2D characters. The project can seamlessly by ported to work for 3D characters as well.

Keywords: kinect, mesh, animation, 2D, skeleton

1 Introduction

A 2D mesh is a collection of vertices which combine to form fragments which are ultimately texture mapped to depict some a character. A top-down approach to animate this mesh is to fit a skeleton into it and map these vertices with some weight to each of the neighbouring k bones of the skeleton. In this project we try an alternate bottoms-up approach where we have the skeleton structure and bones and we map piece-wise parts of the mesh on to each bone. This gives a rather puppetry feel to the created character. Next we use kinect to animate this character and mimic the person standing infront of the kinect. Some straight-up assumptions made in this project are as follows:

1. Movements along the z-axis are not dealt with.
2. Mapping between kinect skeleton and mesh skeleton are provided.
3. The entire scene remains in 2D.
4. The entire body remains in front of the kinect device.
5. There are no occlusions in the scene.
6. Assumption of character coherence and volume conservation - No change in length of bones.

*e-mail:sarthak12088@iiitd.ac.in

2 Introduction to Kinect

2.1 History

Kinect was initially introduced by Microsoft (based on the technology developed by primesense, Israel) in 2010. It is a infrared based depth sensor which uses the patterns of change in a known speckle pattern when projected on the scene. Development can be done on the kinect in two ways - either through the microsoft kinect SDK or opensource libraries such as OpenNI and libfreenect. While the MS SDK runs only on windows, it receives great support from microsoft and is easier to operate on. With the Kinect 2.0 coming in support from other open source libraries has greatly declined and MS SDK has in-evidently become the first choice for development.

Figure 2: *Kinect: Different Parts*

2.2 Working

Figure 1 shows the different parts of a kinect camera. The primary parts are the infra red projector (which projects the pattern on the scene) and infrared sensor (which interprets the disparity in the pattern to calculate the depth as depicted in figure 2).By the similar triangle property we find our selves with a single missing constraint

in the following equation which gives us the depth of a pixel based on it's disparity:

$$Z_o = \frac{Z_r}{1 + \frac{Z_r}{fb} d} \quad (1)$$

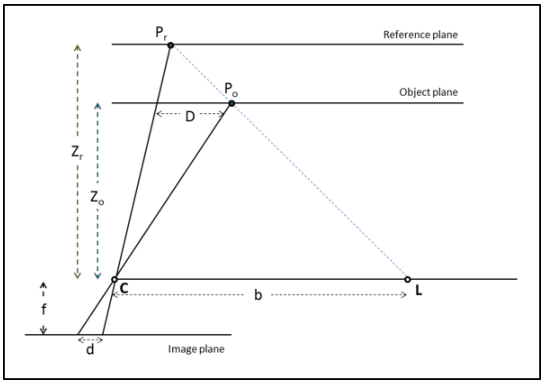


Figure 3: *Calculation of Depth in Kinect*

2.3 Application

Another special thing about the kinect which is of immense importance to this project is it's capability to detect humans. With the microsoft kinect SDK comes bundled a very useful set of libraries - one of them is the "Body Part Detector" which is basically a very high accuracy trained Random Decision Forest. In this project we directly use this library to obtain joint coordinates from the frame. The figure 2 depicts the steps internally followed by kinect to obtain the joint coordinates of a human in a frame.

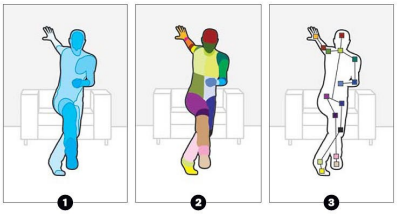


Figure 4: *Body Part and Joint Detection in Kinect*

3 Related work

3.1 Animation of 3D Characters from Single Depth Camera

[Mian Ma] In this approach a simple depth camera was used to capture and was the first case where the assumption of coherence was found - volume of character remains constant. It uses the rodriguez axis angle formula to calculate the alignment of two bones. This is the basis of this reports approach as well - in the 2D context.

3.2 Human Pose Tracking in Monocular Sequence Using Multilevel Structured Models

[Lee] This basically consisted of three steps:

1. Human Blob Detection - Based on traversing a trained color histogram.
2. Body Part Recognition - Identify face, skin using image analysis.
3. 3D Interpretation of Body Parts - Use belief networks to place joints between detected bodyparts.

Though it had some positives such as it was doing automatic rigging and was mostly immune to occlusions, it did not run in real time due to it's complexity.

3.3 Automatic Rigging and Animation of 3D Characters

[Baran and Popovi] This was an introductory paper on Automatic Rigging of animated 3D meshes to obtain the skeleton. The mapping and animating algorithm resembled the work by [Mian Ma].

3.4 Shape your body: Control a Virtual Silhouette

[Leite and Orvalho] This was an application made specifically for silhouettes using 14 joints. It involved manual rigging but was based on the same algorithm as [Mian Ma].

3.5 Augmented Mirror: Interactive Augmented Reality System Based on Kinect

[Luca Vera] This application used kinect as a primary sensor but fused it with a host of multiple sensors as well which is something out of the scope of this project.

3.6 Real-time physical modelling of character movements with Microsoft Kinect

[Shum and Ho] This paper by microsoft used a trained database of poses to match the current pose to these learned actions in a supervised fashion. Again this proved to be out of scope for this project.

3.7 Animation of 3D Human Model Using Markerless Motion Capture

[Shingade1 and Ghotkar] Same approach as [Mian Ma].

4 Milestones

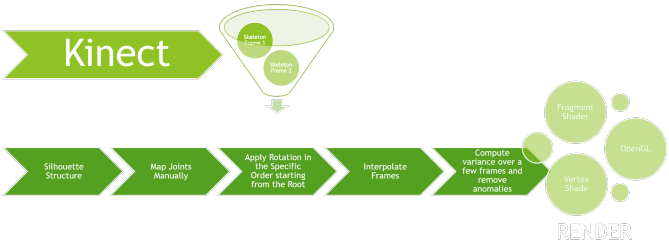


Figure 5: *Algorithm Pipeline*

The following milestones were decided upon on the onset of the project:

1. Literature Survey - Get up to date with the State-of-the-art and plan goals.

2. Independent Implementation - Independently setup the QT Skeleton and Obtain data from the kinect device.
3. Integration - Integrate the QT application and kinect to make the QT mesh mimic the kinect skeleton.
4. Optimization - Improve the output received on integration with filtering and carefull texture mapping.

5 Skeleton Coding

5.1 N-ary Tree

Figure 3 depicts the Joint tree used.

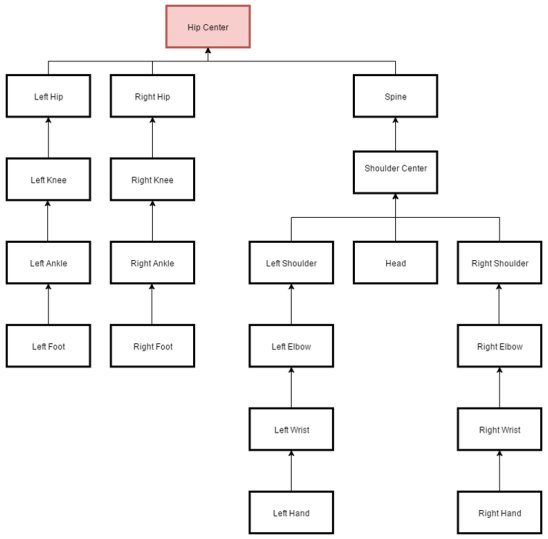


Figure 6: *Joint Tree with the Hip Center as the Root*

6 Kinect Integration

6.1 Bind Pose

Figure 3 depicts the skeleton constructed from the joint coordinates. These joints are numbered as shown and are mapped to specific Kinect joints. This mapping remains constant throughout.

6.2 Implementation

Figure 3 depicts the implementation of the code. Visual Studio communicates with the kinect to get the skeleton coordinates which are save as files in the buffer folder. The QT application picks up the frames from the folder (deleting them simultaneously) and uses them as direct input from kinect.

7 Animation

7.1 Forward Kinematics

The bone being a part of the joint tree transforms recursively to match the kinect skeleton. The phenomena being followed is forward kinematics where the matching begins from the root of our joint tree i.e. the hip center. The bones originating from the hip center are matched to the skeleton first and all their children suffer

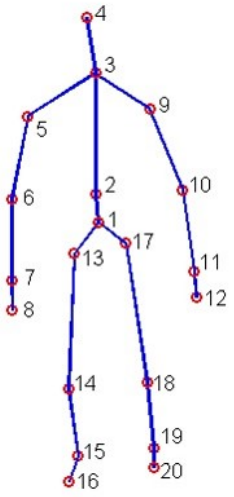


Figure 7: *Bone Structure Created in QT*

Figure 8: *Implementation Pipeline*

this transformation. Then we move lower down the tree and recursively follow the same procedure to map the bones in the kinect skeleton to our mesh skeleton. For this we find the directed angle of rotation using the axis angle method of computing the dot and cross product between 2 2D vectors. Figure 3 depicts the same.

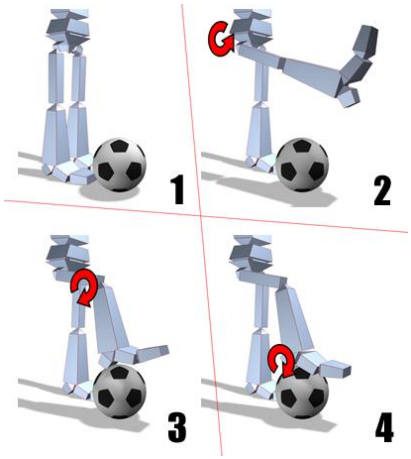


Figure 9: *Forward Kinematics*

7.2 Noise Filtering

A mean filter is added with customizable window and step size to smooth the joint coordinate data being received from the kinect device. This was a crucial add-on as it removed a very pertinent problem of jerkiness in the animation. Figure 4 depicts the effect of filtering on a signal. The incoming joint coordinates can be treated

like a signal and the noise can be removed by a simple convolution of a mean filter window over a buffer collected.

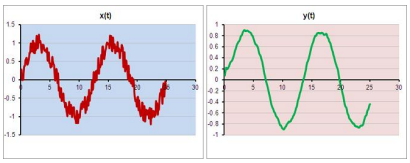


Figure 10: *Noise Filtering*

7.3 Interpolation

Further simple linear interpolation along theta is done to break large rotations into smaller ones. This gives another element of smoothness to the animation.

7.4 Constraint Setting

Each joint has a set of constraints bound to it which do not allow rotation, translation or scaling beyond a certain extent. In this implementation time has not been spent on designing each and every constraint and a generic constraint of not allowing more than 180 degree rotation is imposed.

7.5 Rigid Mesh Transformation

Vertices are computed based on the bone in a fashion that the quadrilateral is almost a rectangle in the direction of the bone. The actual transform is done on the bone as suggested above by forward kinematics.

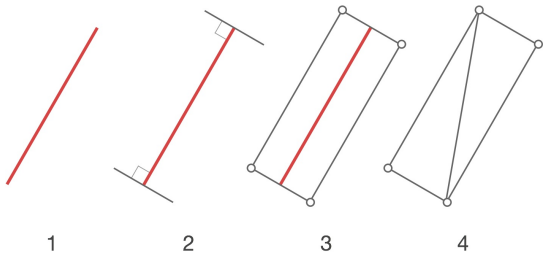


Figure 11: *Making vertices over bones*

8 Designing Baymax

Baymax as depicted in figure 4 is a very popular character from the movie ”Big Hero 6”. It has a very distinct body structure making it ideal for this kind of project. Creating Baymax from scratch involved breaking the image into body parts and mapping them to the bones of the skeleton. It was not at all as simple as it sounds as a number of hacks had to be made to create the ideal Baymax character in qt.

8.1 Texture Mapping

For texture mapping a few things had to be considered - First, the image parts have transparent elements and thus require some sort of blending. For this we have to use the GL_BLEND functionality. The following set of commands achieve exactly that. (Need to load the image by specifying the RGBA format)

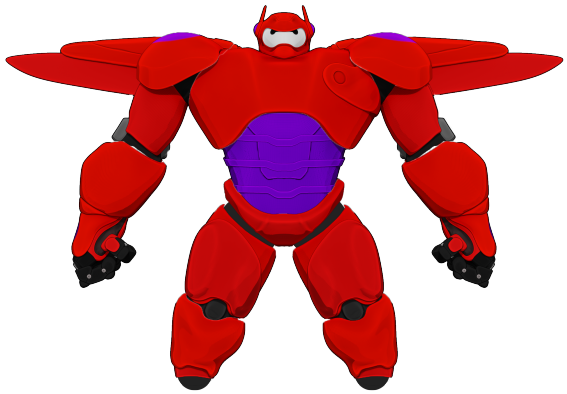


Figure 12: *Baymax!*

```
glBlendFuncSeparate(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

The first and second line signal open_gl to replace the transparent spaces in the mapped texture with the contents of current buffer rendered on the screen. The third line is used in cases whenever the pixel being textured maps to an area greater than one texture element. GL_Linear returns the weighted average of the four texture elements that are closest to the center of the pixel being textured.

8.2 Improvement Hacks

Some hacks that were made to give the character a more realistic touch were as follows:

1. All bones were mapped with aligned quadrilateral planes i.e. 4 mesh vertices mapped to each bone creating one texture quadrilateral(parallelogram to be specific). These planes were then texture mapped with images containing transparency on regions where no texture was present in the body part. Figure 4 depicts the same.
2. Order of Rendering: The ordering was done such that the theoretically nearer parts are rendered last.
3. Texture Overlay: Besides just mapping the texture across the bone on a quadrilateral, the quadrilateral was made modifiable by extending it in necessary directions along the bone to cover up the gaps between the bones as depicted in Figure 4.

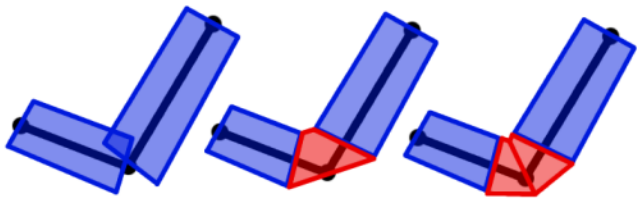


Figure 13: *Extend texture in direction of the bones to cover gaps*

To create the quadrilateral we follow:

$$dx = x1 - x0; \tag{2}$$

$$dy = y1 - y0; \tag{3}$$

$$linelength = sqrt(dx * dx + dy * dy); \tag{4}$$

$$dx/ = linelength; \tag{5}$$

$$dy/ = linelength; \tag{6}$$

$$delta = 5.0f; \tag{7}$$

$$dx = 0.5f * delta * (-dy); \tag{8}$$

$$dy = 0.5f * delta * dx; \tag{9}$$

We follow all 4 combinations of the above two dx and dy to create four vertices of the mapping quadrilateral as follows:

$$(x0 - dx, y0 + dy); \tag{10}$$

$$(x1 - dx, y1 + dy); \tag{11}$$

$$(x1 + dx, y1 - dy); \tag{12}$$

$$(x0 + dx, y0 - dy); \tag{13}$$

To extend the line we simply use it's slope as follows:

$$x = old_x + length * cos(alpha); \tag{14}$$

$$y = old_y + length * sin(alpha); \tag{15}$$

9 Evaluation

9.1 Data

We found a very useful database on the Microsoft Research Page which contained data of 16 activities performed by 30 subjects. This came in very handy while evaluating the code.

Figure 14: *Snapshot from the MSR database*

9.2 Live

Apart from testing on databases we often performed live testing on subjects.

10 Results

10.1 User Interface

Figure 5 depicts the interface created in QT. It allows one to independently control all bones of the character as well as perform all functionality implementations at the click of a button. A toggle to modify the size of the window filter is also provided.

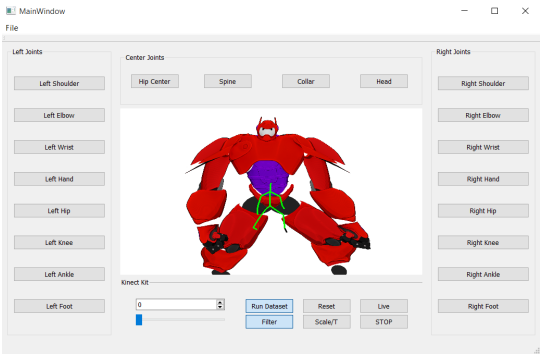


Figure 15: *User Interface in QT*

10.2 UML

The UML for the code is depicted in figure 7. The code contains primarily the following classes:

1. Scene - Contains the Canvas, Datasets and Character.
2. Human/Baymax - Contains the Joint Tree. Implements Character Class.
3. Character - Abstract Class to create the character.
4. Joint - Abstract Class to create character body parts.
5. Thigh/Ankle/etc - Implements Joint functions to display Texture.
6. QPointF - In-built QT pair structure to store joint coordinates.
7. Transformation - 3x3 Rigid Transform matrices.
8. Dataset - Has a dual implementation to collect live data from the kinect buffer folder and to collect data from the available datasets.
9. Filter - Performs Mean filter on a collection of frames.

11 Discussions

11.1 Evaluation 1

After evaluation 1 the discussions revolved around setting tangible milestones and discussion on the approaches suggested based on the literature survey.

11.2 Evaluation 2

Post the second evaluation the point of discussion was the integration of kinect with qt. Kinect posed a lot of issues due to unavailable or limited support for ubuntu and incompatibility of the SDK with QT. This was resolved by using Visual Studio to communicate with the kinect device.

11.3 Evaluation 3

Suggestions received after evaluation 3 were mainly about adding a noise filter and making a choice between rigid and non-rigid transforms. The noise filter was successfully created as a part of this evaluation. Owing to the available time the former approach of having only rigid transforms was adopted.

12 Future work

12.1 Non-Rigid Mesh Transformation

Figure 6 depicts the two approaches that were pointed out in the beginning. In this project we have implemented the former Bottoms-up approach which has certain limitations such as the effort that has to be put in dividing an image into smaller parts to map the bones. The latter approach would in many ways improve the visual element of the project besides adding to the ease of creating an animated character.

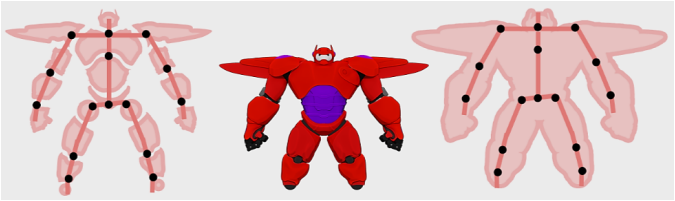


Figure 17: *(L)Bottoms-Up Approach to (R)Top-Down Approach*

12.2 Extend to 3D

Movement along the Z-axis is not dealt with in this iteration. Designing the structure in 3D and projecting it on a 2D image plane will solve and remove the assumption of non-uniform scaling.

12.3 Improve Scaling and Translation

In this iteration of the project a clean implementation of Scaling and Translation was not accomplished in a successful fashion. Though in 2D scene it does not carry as much importance as it does in the 3D scenario, this would be a task to complete later on.

Acknowledgements

I would like to thank Dr. Ojaswa Sharma under whose guidance I completed this project.

References

BARAN, I., AND POPOVI, J. Automatic rigging and animation of 3d characters.

LEE, M. W. Human pose tracking in monocular sequence using multilevel structured models.

LEITE, L., AND ORVALHO, V. Shape your body: Control a virtual silhouette using body motion.

LUCA VERA, JESS GIMENO, I. C. M. F. Augmented mirror: Interactive augmented reality system based on kinect.

MIAN MA, FENG XU, Y. L. Animation of 3d characters from single depth camera.

MING ZENG, ZHENGCUN LIU, Q. M. Z. B. H. J. Motion capture and reconstruction based on depth information using kinect.

SHINGADE1, A., AND GHOTKAR, A. Animation of 3d human model using markerless motion capture applied to sports.

SHUM, H., AND HO, E. S. Real-time physical modelling of character movements with microsoft kinect.

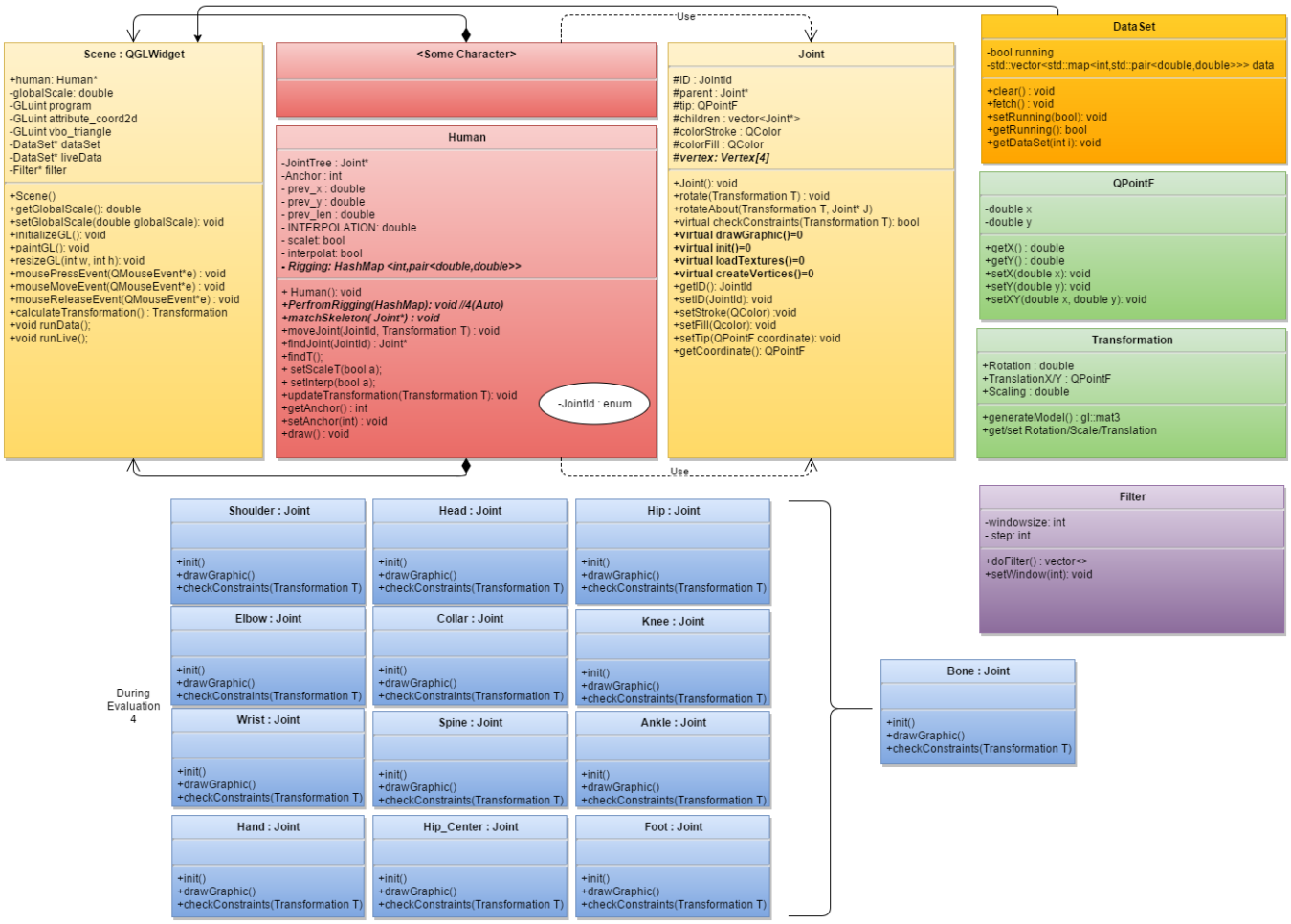


Figure 16: *UML for the Code*

INDRAPRASTHA INSTITUTE *of* INFORMATION TECHNOLOGY **DELHI**

CSE 533/333 Computer Graphics, IIIT-Delhi

Kinect Driven 2D Mesh Animation

Sarthak Ahuja (2012088)

Under the guidance of Dr. Ojaswa Sharma

INTRODUCTION

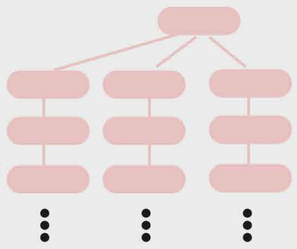
Kinect uses it's infrared sensor to calculate depth images of a scene. It further is capable of detecting human s in the frame and identify their joint coordinates. This project is aimed at animating a 2D character using these joint coordinates provided by kinect. Kinect 2.0 is used in this project to provide the skeleton which is mapped to our character in QT and rendered using OpenGL.

Current Constraints:

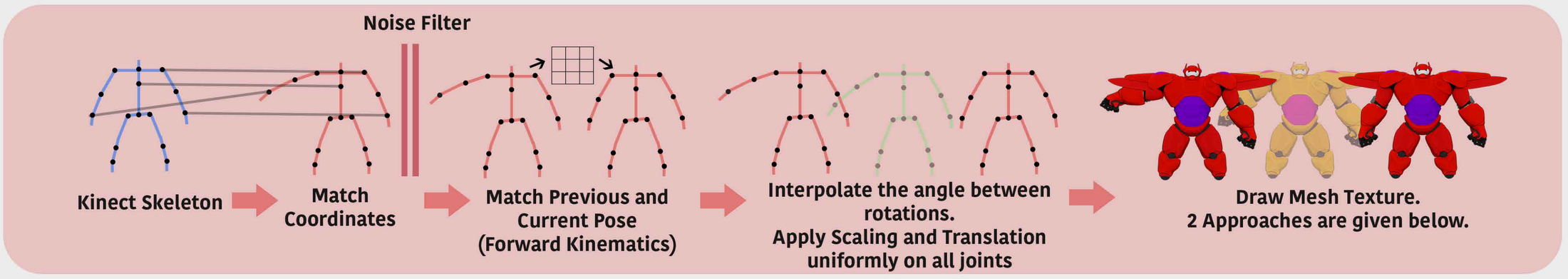
- Rigging between mesh skeleton and kinect joints is available.

UNDERLYING STRUCTURE

- The Character is setup as a **N-ary Tree**.
- The Nodes of the tree represent **joints**.
- These joints store **transformation matrices**.
- Each joint matrix affects it's children. (**Recursive**)
- The root node in our case is the **hip center**.
- In the current model we have **20** joints



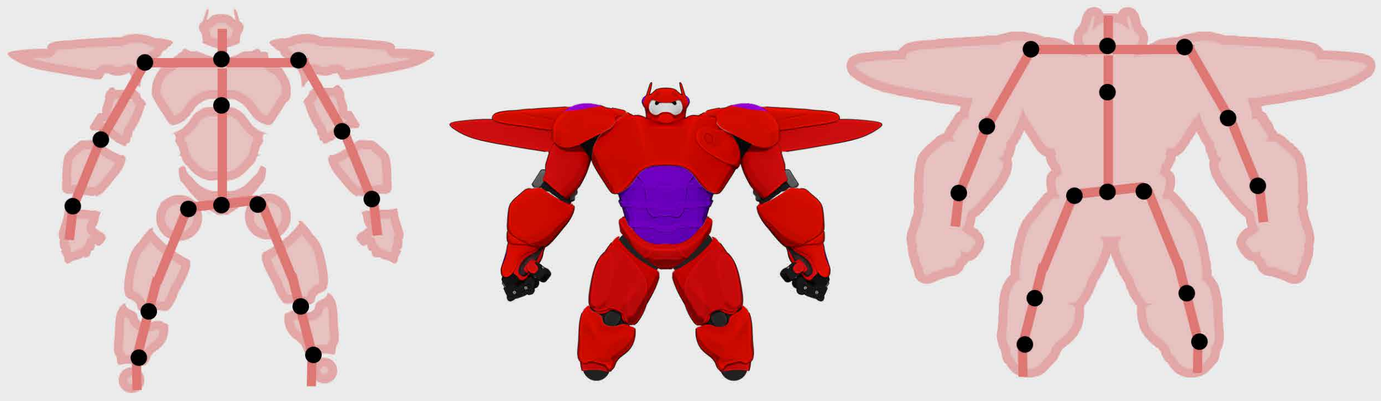
PIPELINE



Performing Mesh Transforms

Approach 1
Rigid Transforms

- **Separate Mesh** for every Body Part.
- Distance between two points in a mesh remains same throughout
- Gives **Puppet/Silhoutte** feel.
- Unable to clearly cover joints with mesh during animation.
- **Easy to implement**



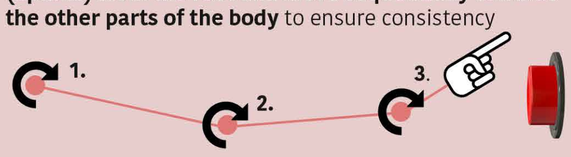
Approach 2
Non-Rigid Transforms

- **One Single Mesh** for entire skeleton.
- Transformation of mesh points as a weighted function of the neighbouring joints.
- Gives **Actual Character** feel.
- Able to cover joints with mesh during animation with reasonable clarity
- **Tricky to implement**

IMPLEMENTATION DETAILS

Forward Kinematics

- To function correctly we need to **begin rotations (update) from the root and move sequentially towards the other parts of the body** to ensure consistency



- Constraints
All body parts have certain constraints associated with their parents, so as to how much they can be rotated, moved or scaled. This is necessary to maintain structure and character.

Implementation Pipeline

- Data from Kinect is subscribed using Visual Studio and the joint coordinates are published as files in a buffer folder.
- The QT application picks up the files in sequence (simultaneously deleting them) and uses them as the input Kinect Skeleton Joint coordinates.

Classes and Code Structure

- Scene:
This is the main class containing the canvas and character.
- Character
This is the generic character class which contains the joint tree.
- Dataset
This is the class used to buffer in frames.
- Joint
This is the node of the created tree which forces children to implement the draw function.
- Transformation
This is the class used to create Transformation matrices.
- MainWindow
This is the main UI class.

- Baymax - Implementation of **Character**
- Thigh, Head ... - Implementation of **Joint**

FUTURE WORK

- Automatic Skeleton Generation and Mapping
As of now we are assuming we have a matched skeleton

- Extend the system to 3D
As of now we cannot deal with Scaling along z-axis.

CALCULATIONS AND OPTIMIZATION

- Noise Removal: A median filter is applied to remove noise in the joint coordinates received from kinect.
- Interpolation: Linear Interpolation is performed on the rotation angle.

EVALUATION

Besides live testing, evaluation done on the MSR skeleton dataset
http://research.microsoft.com/en-us/um/people/zliu/actionrecorsrc/
16 activities, 30 subjects

REFERENCES

- Real-time physical modelling of character movements with Microsoft Kinect, **Shum and Ho**
- Motion Capture Applied To Sports, **Ashish Shingade and Archana Ghotkar**
- Animation of 3D Characters from Single Depth Camera, **M.May, F.Xu Y. Liu**
- Shape-aware MLS deformation for line handles, **Sharma, Ojaswa, and Ranjith Tharayil**