Geometric Image Processing Lab

LiveCap Cover

By Izo & Shai

1. Overview

- We describe what we did. More detailed description of the original work of LiveCap can be found in the report.

- This presentation builds on the ideas bottom up, first we describe the individual components, then how they are joined together.

Road map:

1. Overview
2. Problem Description
3. Human 3D Modeling
4. Non-Linear Least Squares (NLLS) Optimization
5. Image Processing
6. Combining it all
7. Implementation
8. Experiments
9. Main Differences From The Original Work
10. Conclusions & Future Work

2. Problem Description

Real Time Motion Capture

Our task is to capture the movement of a human from a video.

<u>To Capture The Movement</u> in our context means to recreate a 3d model that moves similarly to the human observed in the images.

Different uses: game industry, medical world...

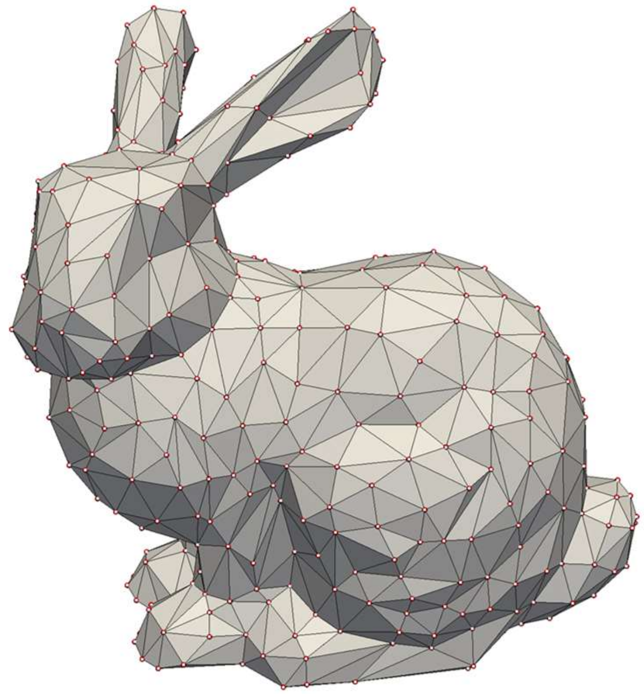
Left to right
1. The model in the rest pose
2. Input image
3. Model without texture
4. Model with texture
5. Textured and untextured model from different views

Motion capture - from livecap paper

3. Human 3D modelling

Representing Geometrical Shapes

- This is done by approximating the shape with many small polygons - a **Mesh**.
- The polygons here are called **Faces**, and are here triangles.
- Each face is composed out of **vertices**, connected by **edges.**
- Vertices are points in 3d space.

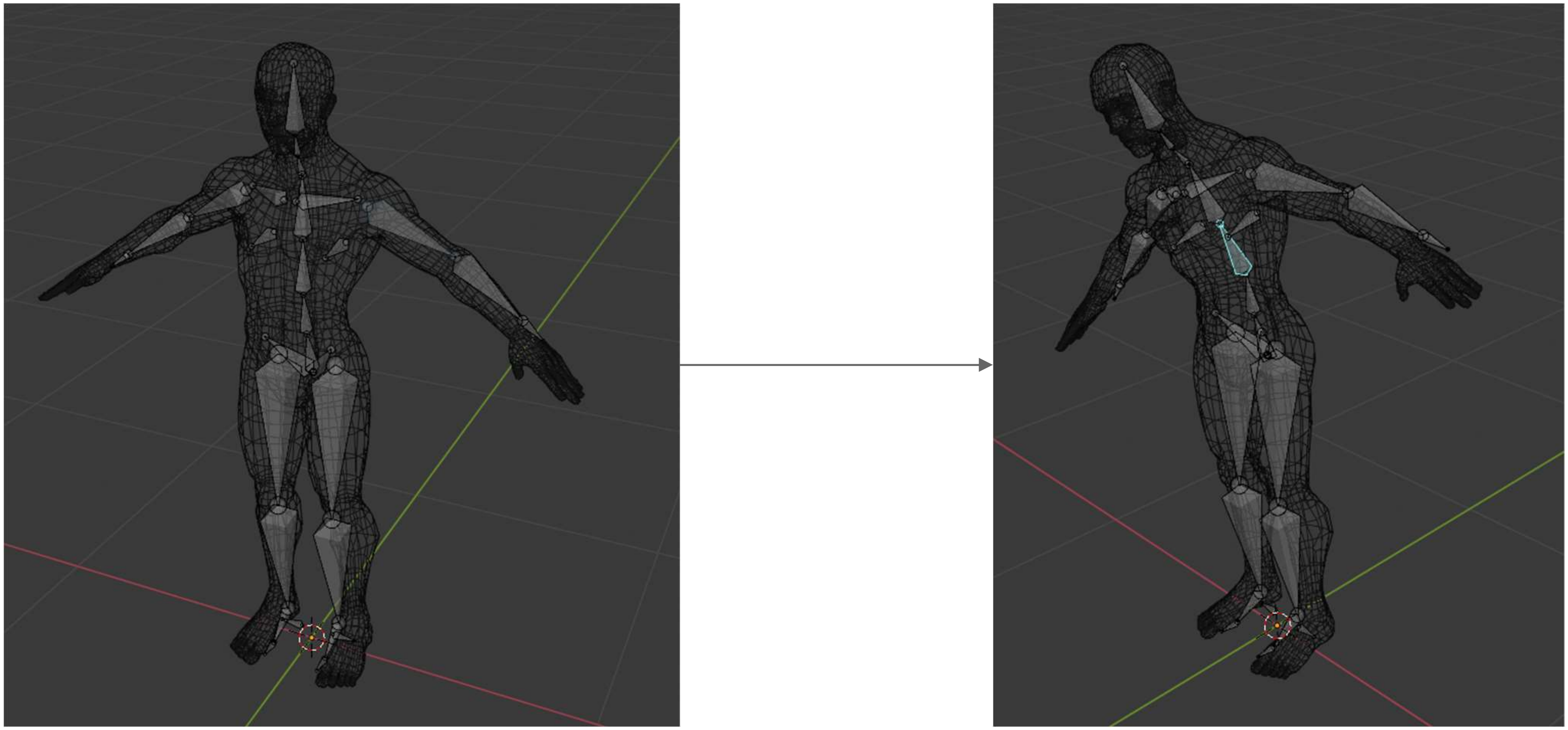


3D mesh - can see vertices, faces, edges - from pyvista docs

Movement Model

- We use a movement model called **Linear Blend Skinning (LBS)**.
- We have a **skeleton** that is an hierarchical set of **joints**.
- Each joint has a **parent** (or is the root), and a set of transformations.

- **T_joint_to_parent** is the rigid transformation from a joint to its parent joint.
- **T_joint_to_model** is a transformation from the joint space to the model space.
- **T_model_to_joint** is a transformation from the model space to joint space.

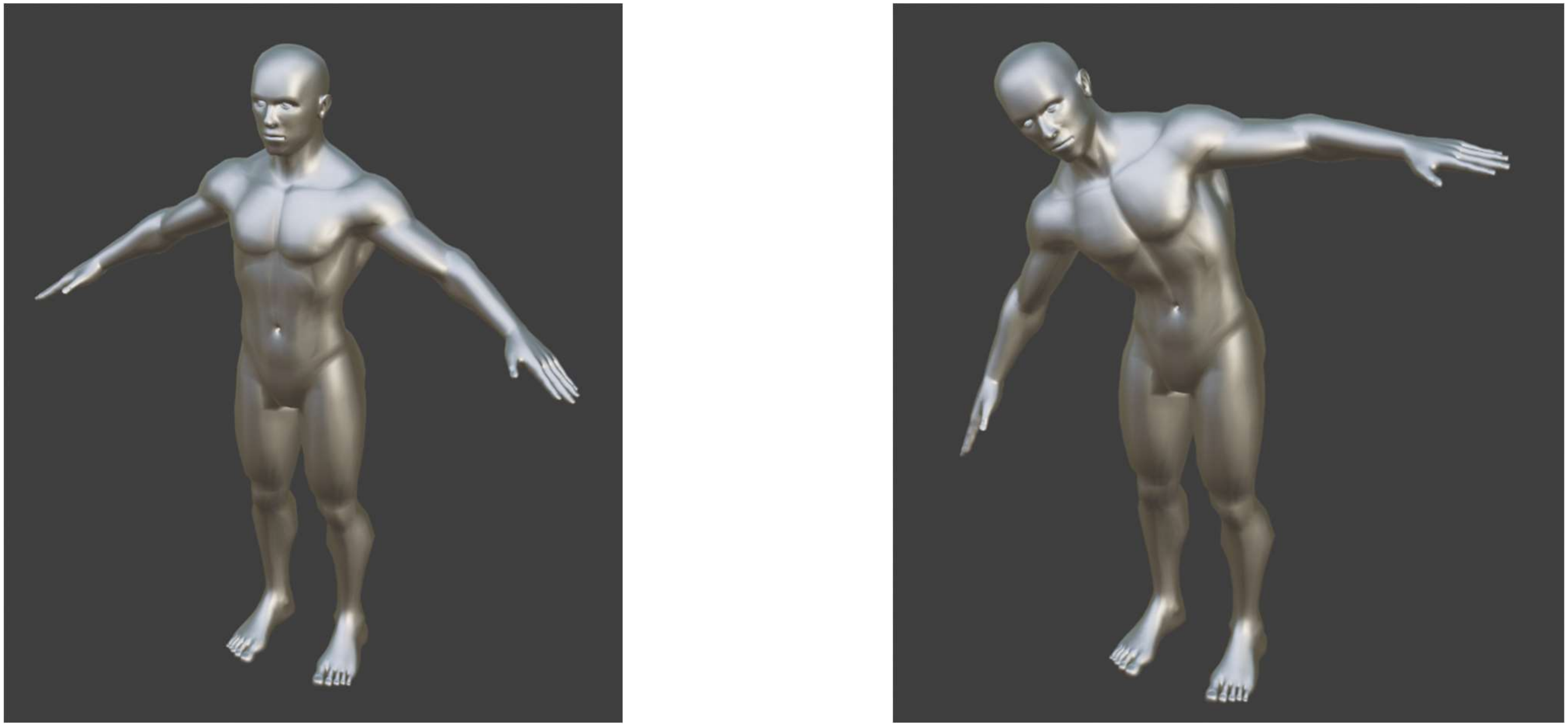


Skeleton Movement - a single joint movement(spine) affects all its children - blender

Movement Model (cont.)

- Each **vertex**, **joint** pair has an associated weight, s.t. All of the weights associated with that vertex are summed up to 1.
- ***weight(i,j)*** means, how much vertex ***v_i*** is affected by joint ***j***.
- Most of the weights are 0.

$$\mathbf{v}_i' = \sum_{j=1}^{m} w_{i,j} \mathbf{T}_j \mathbf{v}_i$$



Same model shape change after the rotation of the spine joint - blender

4. Non-Linear least squares optimization

NLLS

- Least Squares is a minimization problem, where we have some vector function ***f***, that we try to find the argument to this functions that yields the minimum value in terms of the L2 norm:

$$f : \mathbb{R}^m \to \mathbb{R}^n$$
$$\theta^* = \underset{\theta \in \mathbb{R}^m}{argmin} ||f(\theta)||_2^2$$

NLLS (cont.)

- When ***f*** is a linear function, we have a closed form solution.
- When ***f*** is **non-linear**, we usually iteratively solve for the optimal parameters, by using first(linear) or second (quadratic) approximations.
- There are many methods for solving NLLS, the paper uses Gauss-Newton method, we use **Levenberg-Marquardt**.
- To use NLLS solvers, we need to formulate our problem with a vector valued **cost function**, that is ***f***.

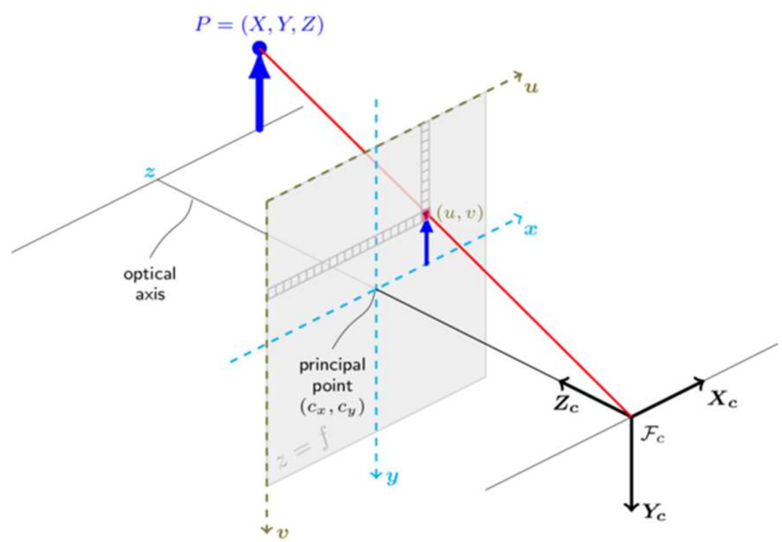
5. Image processing

Projection and camera calibration

- Camera model parameter
- The projection operator

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{R|t} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

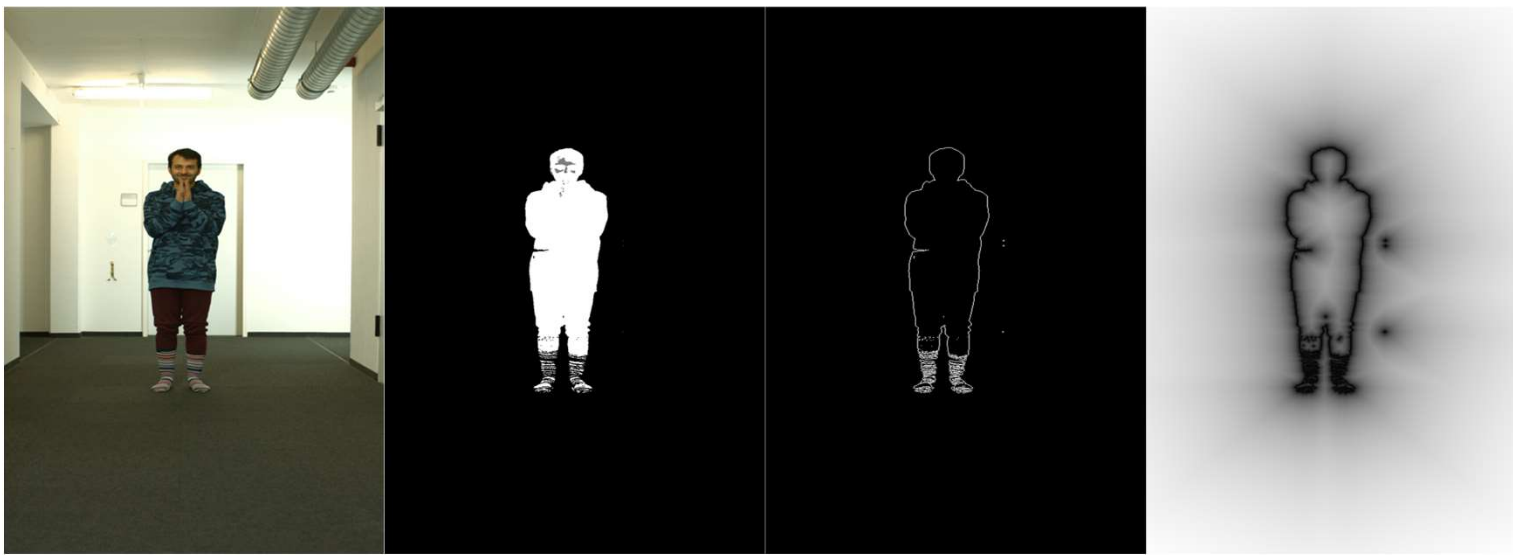
A - intrinsic parameters, camera matrix
R - extrinsic parameters



- **Camera calibration:** finding A

Silhouette extraction

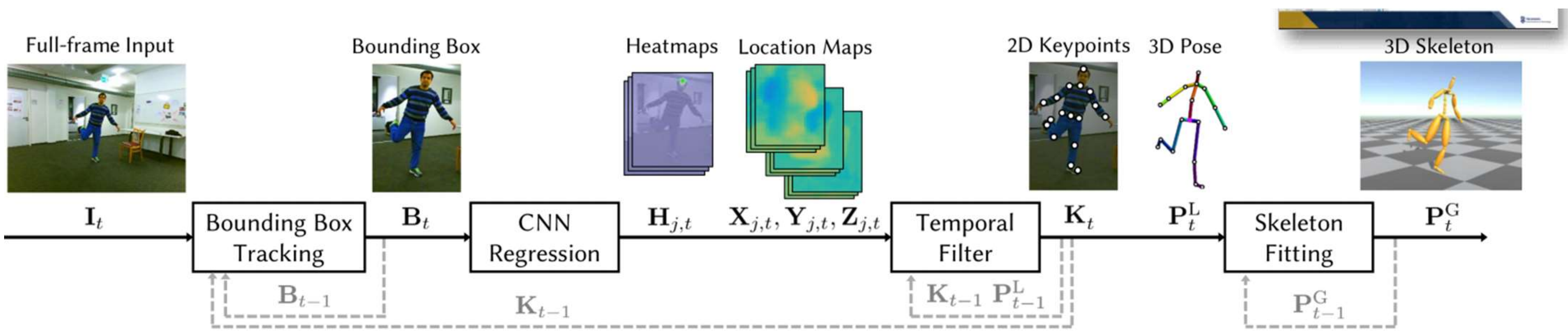
- Background subtraction
- Silhouette extraction using Laplacian operator
- Image Distance Transform



Input image | Foreground mask | silhouette | IDT (logscaled)

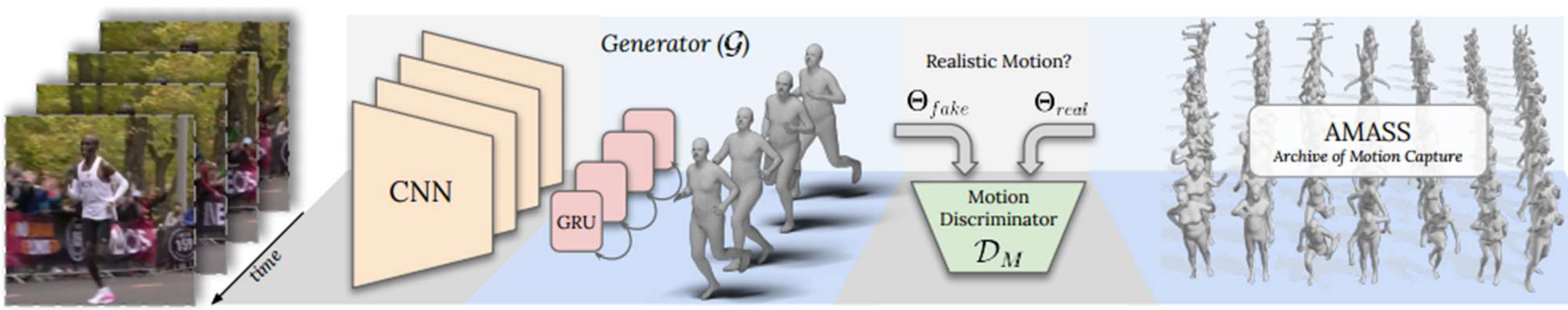
3D & 2D pose estimation

- VNECT
  - **Input**: sequence of monocular images
  - **Output**: 2d and 3d joint positions
  - Image is passed through a CNN, producing heatmaps
  - Heatmaps are passed through a temporal filter and skeleton fitting stage



3D & 2D pose estimation

- VIBE
  - **Input**: sequence of monocular images
  - **Output**: SMPL body shape and pose parameters (~90).
  - The input is passed through a CNN feature extractor
  - Features are combined over time using RNN
  - Training uses additional Discriminator to make sure that the output parameters are realistic.



6. Combining it all

Combining It All

- We formulate our problem as a **NLLS**.
- The parameters that we optimize are the **root rigid transformation** and the **joints angles**.
- We use our **Motion Model** and the outputs of the **Image Processing** to define our **cost function**, also called **Energy**.

Energy

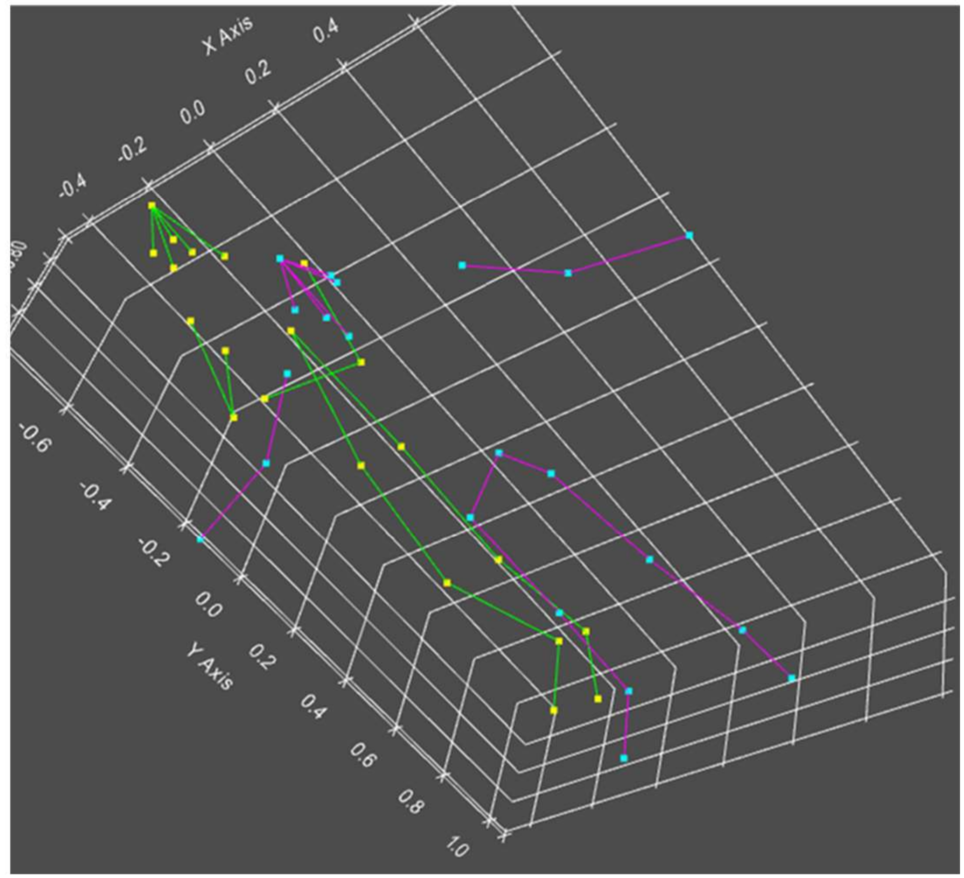
$$S^* = \underset{S}{\operatorname{argmin}}\, E_{\text{pose}}(S).$$

$$E_{\text{pose}}(\mathcal{S}) = E_{\text{2D}}(\mathcal{S}) + E_{\text{3D}}(\mathcal{S}) + E_{\text{silhouette}}(\mathcal{S}) + E_{\text{temporal}}(\mathcal{S}) + E_{\text{anatomic}}(\mathcal{S}) \ .$$

- The parameters to the energy is the **root translation + rotation + joint angles**.
- Is composed of different parts, those are derived either from matching our **model** to the current **frame**(silhouette, 3d, 2d), or based on prior knowledge on the movement of humans(temporal, anatomic).
- For each part we add weights to balance.

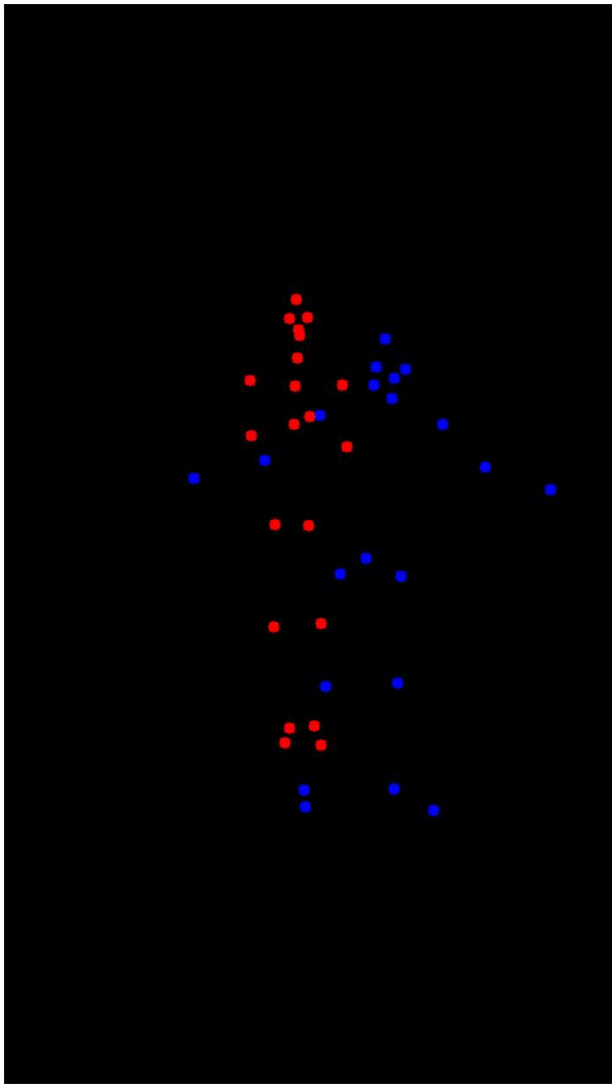
$$E_{3D}(\mathcal{S}) = \lambda_{3D} \sum_{i=1}^{J} \left\| p_{3D,i}(\theta, \mathbf{R}, \mathbf{t}) - \left(\mathbf{P}_{3D,i} + \mathbf{t}'\right) \right\|^2 \quad .$$

The difference between the predicted 3d joint positions and the models 3d joint positions



$$E_{2D}(\mathcal{S}) = \lambda_{2D} \sum_{i=1}^{J+4} \lambda_i \left\| \pi \left(p_{3D,i}(\theta, \mathbf{R}, \mathbf{t}) \right) - \mathbf{P}_{2D,i} \right\|^2 \quad .$$

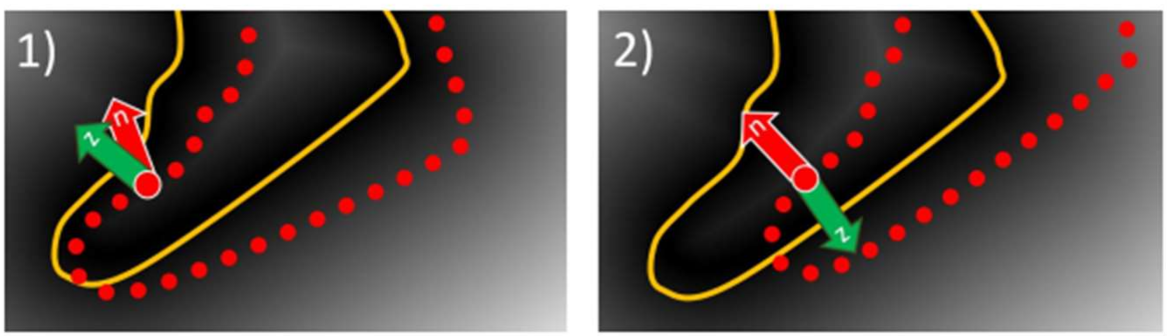
The difference between the predicted 2d joint positions and the projected model's joints

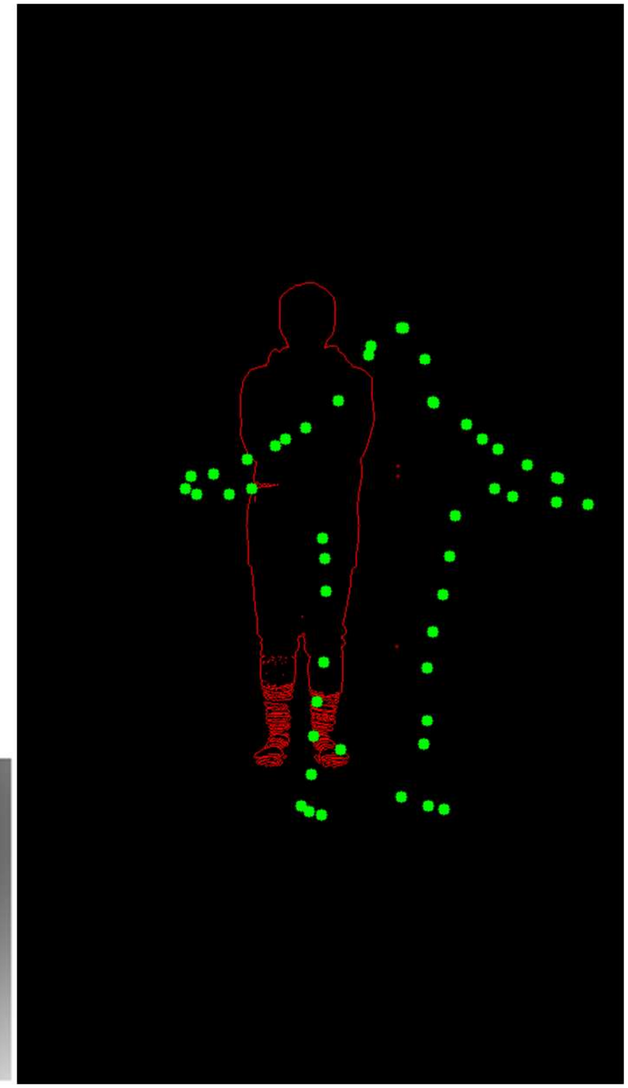


$$E_{\text{silhouette}}(\mathcal{S}) = \lambda_{\text{silhouette}} \sum_{i \in \mathcal{B}} b_i \cdot \left[I_{\text{DT}}\left(\pi(\mathbf{V}_i(\theta, \mathbf{R}, \mathbf{t}))\right) \right]^2 \ .$$

The distance transform of the silhouette at the pixels of the projected contour vertices.

The b in the formula is a special term to give us the correct sign when the point is inside the silhouette





$$E_{\text{temporal}}(\mathcal{S}) = \lambda_{\text{temporal}} \sum_{i=1}^{J} \lambda_i \left\| p_{\text{3D},\, i}(\theta, \mathbf{R}, \mathbf{t}) - p_{\text{3D},\, i}^{t-1}(\theta, \mathbf{R}, \mathbf{t}) \right\|^2 .$$

This is the difference in the model's joint position w.r.t the previous frame estimated pose.

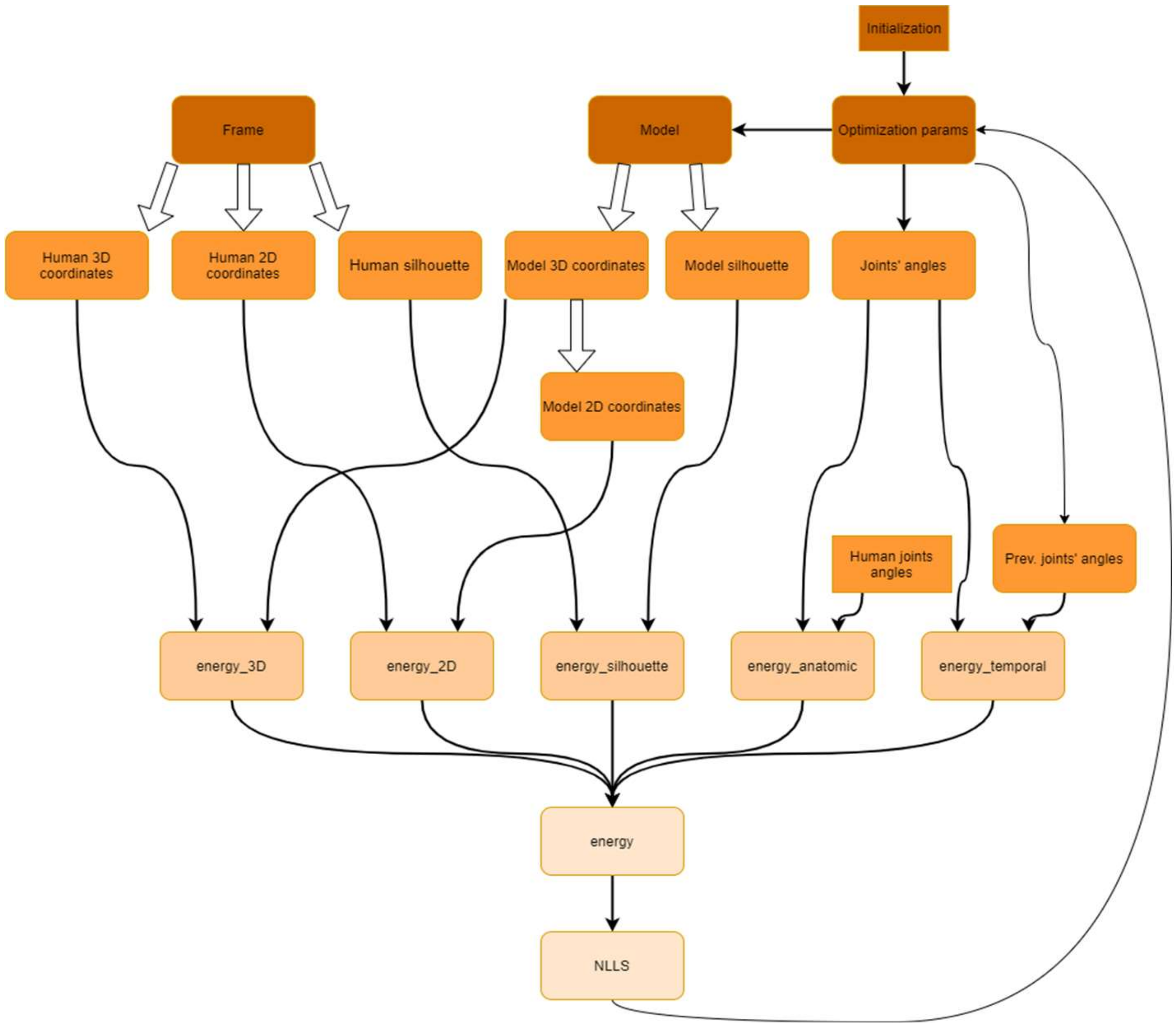
This can also be interpreted as the velocity of the joint.

$$E_{\text{anatomic}}(\mathcal{S}) = \lambda_{\text{anatomic}} \sum_{i=1}^{27} \Psi(\theta_i) \ .$$

$$\Psi(x) = \begin{cases} (x - \theta_{\text{max},\,i})^2, & \text{if } x > \theta_{\text{max},\,i} \\ (\theta_{\text{min},\,i} - x)^2, & \text{if } x < \theta_{\text{min},\,i} \\ 0 & \text{, otherwise} \ . \end{cases}$$

For each joint angle we have an upper and a lower limit.

If we pass one of those limits, then we add the difference to the energy of the pose.



Initialization
Frame
Model
Optimization params
Human 3D coordinates
Human 2D coordinates
Human silhouette
Model 3D coordinates
Model silhouette
Joints' angles
Model 2D coordinates
Human joints angles
Prev. joints' angles
energy_3D
energy_2D
energy_silhouette
energy_anatomic
energy_temporal
energy
NLLS

7.
Implementation

Implementation - code and packages

- **Anaconda** package manager
- **Github** for collaboration
- **Python 3.7** - surprisingly **fast** when used **correctly**.
- **Numpy 1.17** - fast array operations
- **Scipy 1.4** - NLLS solvers and rotations
- **OpenCV 4.1** - part of the image processing
- **Pytorch 1.4** - VIBE
- **Pyvista 0.25** - visualizations
- **Pycollada 0.4** - 3d I/O
- ...

8. Experiments

Exp. 1 - Dance + generic model

For development and testing we used a video downloaded from youtube + a generic human body rigged with blender

Exp. 1 - Conclusions

- Can see that the blending weights are not good, and create some unnatural movement.
- Large number of vertices slows down the optimization.
- Large number of DOFs (3 for each joint) slows down optimization, and leads to some of the unnatural movement.
- Still it kind of works.

Exp. 2 - Original video + model:

Here we used the resources shared by livecap authors, after we had troubles creating our own models.

Exp 2. Conclusions

- The results look much more satisfying.
- Reducing the number of vertices from ~19,000 to ~5,000 and reducing the number of DOFs from ~90 to ~30 greatly improved the runtime speeds, getting to almost real time speeds on a pc alone.
- Some mismatch exists in the mapping from predicted joint locations and model joint location, the results of this can be seen in the video.
- The predictions are a little bit jittery, this might be to due low weight of the temporal energy.

Exp 3. Balancing the energies

We tried changing the balances between the different energies and even dropping some of them.

Experiment / Weight	3d	2d	silhouette	temporal	anatomic
0. Baseline	1	1e-3	1e-3	0.1	0.5
1. No silhouette	1	1e-3	0	0.1	0.5
2. High anatomic + temporal	1	1e-3	1e-3	1	2
3. No Anatomic + temporal	1	1e-3	1e-3	0	0
4 No 3d	0	1e-3	1e-3	0.1	0.5
5. No 2d + silhouette	1	0	0	0.1	0.5

Exp 3. Sample results

1. No 3d loss
2. No 2d and silhouette loss
3. No regulation (anatomic + temporal)

1. No 3d Loss

2. No 2d and silhouette

3. No Regulation

Exp 3. Conclusions

1. Dropping 3d cost significantly reduces the quality of the results
2. Dropping both 2d and silhouette cost is not as significant
3. Dropping the regulatory terms in the cost results in unnatural movement

9. Differences From the Original Work

Differences

- The paper also goes through the model acquisition stage and the non-rigid optimization stage. We did not recreate that due to a lack of time.
- We choose to use Python. The programing language use by the authors is not clearly stated, but we estimate that it was a combination of C++ and MATLAB.
- We did not get real-time results, mainly because we run without gpu, and of the frameworks used.
- We used VIBE instead of VNECT, because we were not able to run it in our computers.

10. Conclusions & Future Work

Conclusions & Future Work

- The importance of making your implementation publicly available
- 3 separate parts - pose estimation, skeleton optimization and model rigging, maybe letting them benefit from each other might prove useful, for example:
  - using the pose estimation module to automatically rig the model
  - fine tuning the pose estimation model using the optimization results to the specific model, to reduce the number of iterations
- Implementing the non-rigid deformation might yield farther insight.