KicKinect - Torwandschiessen

Marcel Bruckner, Kevin Bein, Jonas Schulz, and Chandramohan Sudar

Technical University of Munich

KicKinect

KicKinect is the implementation of the game *Torwandschießen* where the player tries to kick a football through two openings in a goal wall. One hole is aligned at the bottom left and the other hole is at the top right. The player tries to kick the Ball through either of the two openings and collect points (1 point for the bottom and 3 points for the top hole). Both holes are just slightly bigger than the diameter of the ball which makes it difficult to hit. When either of the two holes is hit by the ball, the corresponding points are accredited.

Figure 1: Welcome screen of KicKinect

Linear Blend Skinning

Linear Blend Skinning (*LBS*) is used to deform a mesh by utilizing its skeletal structures. This is especially interesting to render human body or animal movements. *LBS* is a very efficient algorithm due to its linear character and as a result can be used for real-time renderings. To calculate the per vertex translation and rotation the kinematic chain of the bone structure is used.

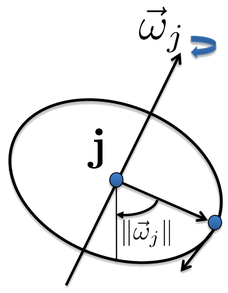

Figure 2: The bone orientation vector

For this project, *LBS* in its basic form is implemented. The translation vector of the bone is defined as the joint position j, $\vec{\omega}_j$ is the scaled axis of the rotation and $||\vec{\omega}_j||$ is the angle of rotation. Using the *Rodrigues* formula the rotation matrix R based on the bone orientation $\vec{\omega}_j$ is calculated as:

$$\mathrm{R} = e^{\hat{\vec{\omega}}} = \mathrm{I} + \hat{\vec{\omega}} \sin ||\vec{\omega}_j|| + \hat{\vec{\omega}}^2 (1 - \cos ||\vec{\omega}_j|) \tag{1}$$

The rotation matrix R forms together with the translation vector j the overall per vertex transformation matrix $G(\vec{\omega}_j, j)$ for the bone:

$$G(\vec{\omega}, j) = \begin{pmatrix} [e^{\vec{\omega}}]_{3\times3} & j_{3\times1} \\ 0_{1\times3} & 1] \end{pmatrix} \tag{2}$$

Due to the natural chaining of the bones along the spine and extremities the calculation of adjacent joints corresponds to a chaining of transformation matrices:

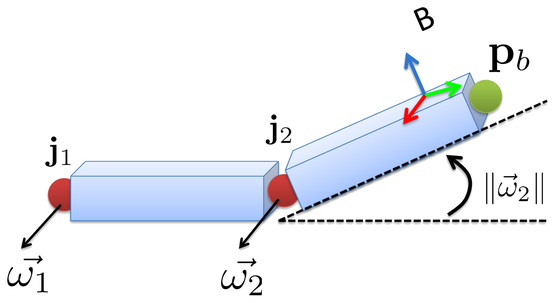

Figure 3: The kinematic chain along adjacent joints

The final transformation matrix $G(\vec{\omega}_j, j)$ of the bones along the kinematic chain is calculated as a concatenation of per bone matrices $G(\vec{\omega}_j, j)$:

$$G(\vec{\omega}_1, \vec{\omega}_2, \ldots, \vec{\omega_k}, j_1, j_2, \ldots, j_k) = G(\vec{\omega}_1, j_1) \cdot G(\vec{\omega}_1, j_1) \cdot \ldots \cdot \ldots G(\vec{\omega}_k, j_k) \tag{3}$$

To calculate the mesh deformation based on the kinematic chain transformations $G(\vec{\omega}_j, j)$ the rest pose vertices $\bar{t}_i$ and the per vertex assigned bone weights $w_{k,i}$ are used. The final transformed vertex position $\bar{t}'_i$ is calculated by using:

$$\bar{t}'_i = \sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta}, J) \bar{t}_i \tag{4}$$

Overview

A *Microsoft Kinect Sensor* is used to track the player and his movements. From the raw RGB-D input data a body skeleton is calculated and extracted.

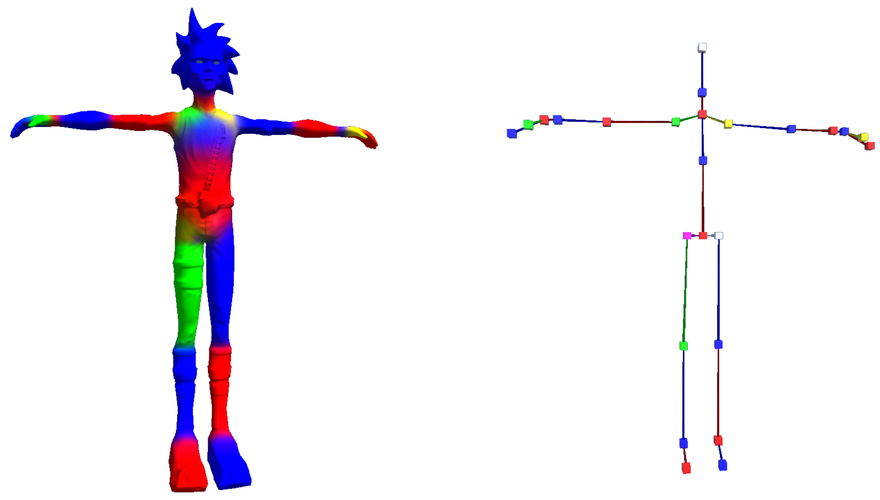

Figure 4: The rest pose mesh and colorful visualized bone weights (left) and the corresponding skeleton bones (right)

By using *Linear Blend Skinning* the rest pose mesh is deformed using the kinematic chain transformations $G(\vec{\omega}_j, j)$. The deformed mesh is assigned to the player game object in a *Unity3D* scene and represents the virtual avatar of the player.
The game content now consists of the player controlling the virtual avatar by moving in front of the *Kinect* and shooting the virtual ball into the holes of the goal wall to score as high as possible.

Figure 5: The player shooting onto the goal wall

Challenges

Ball velocity

Calculating the velocity with which the ball is accelerated towards the goal wall must be tuned manually. The following aspects account for a realistic value:

1. The distance of the player to the goal wall
2. The overall scaling of the scene
3. The size of the ball in relation to the player
4. The amount of hitboxes around the players feet as well as their shape
5. The angle of the hit and the rapidity of the body movement when swinging the foot
6. The mass of the ball and the foot

The first four aspects can be tuned directly in Unity's physics and model framework by modifying the parameters of the models and rigid bodies. For the fifth and sixth aspect, applying the formula of an elastic collision yielded very good results:

$$v_{ball;shot} = \frac{(2 \cdot m_{\text{ball}} \cdot v_{\text{ball}} + (m_{\text{foot}} - m_{\text{ball}}) \cdot v_{\text{foot}})}{(m_{\text{ball}} + m_{\text{foot}})} \tag{5}$$

Synchronization

The biggest challenge was to synchronize the *Kinect SDK* with *Unity3D*. The *Kinect* is recording an RGB-D input stream and through the *Kinect SDK* it provides the joints of the body (25 joints in total) and some orientations.
Even though, ultimately, these values are correct, the joints and rotations do not follow standard character rigging conventions. We had to implement a very tough mapping from the received *Kinect* joint data to our rigged character to be able to apply the *Linear Blend Skinning* algorithm presented earlier.
Unfortunately we were not able to integrate correct per bone rotations but could implement the game using some *Unity3D* packages.