Inverse Kinematics for Shape Grammars

Patrick Kühtreiber
Student

Martin Ilčík
Supervisor

Institute of Computer Graphics and Algorithms
Vienna University of Technology

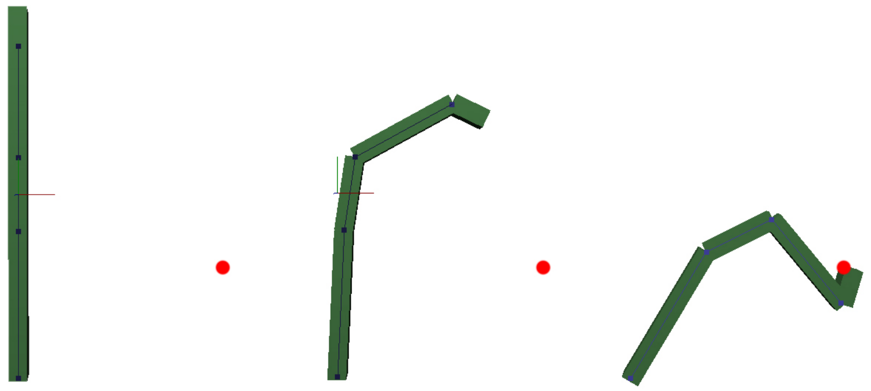

Figure 1: From left to right: the three-joint figure approaches the user specified target point (red dot).

Abstract

In this report I will describe the implementation of a solver of the Inverse Kinematic (IK) problem in an environment using shape grammar based on CGA Shape. There exists a number of different algorithms for this problem, however, this report describes only the cyclic coordinate descent (CCD) in more detail as this is the method we implemented.
The implementation enabels the positioning of a certain shape so that it touches a user-specified target point. The implemented algorithm rotates the shapes until the end effector is at the desired position.

1 Introduction

The IK problem is the opposite of forward kinematics where you can rotate all the shapes until the so called ”end effector” targets a certain point. With IK you can adress this problem from the other way. When using CCD the IK-algorithm takes the target point as input and iterates multiple times through all the shapes and operates on the different degrees of freedom of the object (e.g. rotation on a human arm with certain consraints) until the defined shape touches the point, which needs of course a more sophisticated approach.
Additionally this IK solver needs to be implemented as a rule in a shape grammar environment.
After describing the CCD-algorithm and our implementation in detail I will conclude this report with an outview and future work. The next section contains references to various IK solvers.

2 Related Work

There exists a number of different solvers for the IK problem, most coming from the field of robotics. Besides the already mentioned cyclic coordinate descent (CCD) [Wang and Chen 1991] there are also the pseudoinverse methods [Whitney 1969], Jacobian transpose methods [Wolovich and Elliott 1984], the Lavenberg-Marquardt damped least squares method [Wampler 1986], quasi Newton and conjugate gradient methods [Zhao and Badler 1994] and neural net and artificial intelligence methods [Grzeszczuk and Terzopoulos 1995]. I concentrated on the CCD because it is the method that is easiest to implement and also it provides very good looking results.

3 Cyclic Coordinate Descent

The cyclic coordinate descent approach, developed by [Wang and Chen 1991], has a lot of advantages over the previously mentioned IK solving approaches. The main advantage is, that it is easier to implement. The CCD algorithm for a model with an arbitrary number of joints works as follows. It starts at the endeffector which is usually the last shape of a model (e.g. the hand on a human arm). The algorithm is recursive. To leave the recursion, a certain thershold value needs to be chosen. This value specifies how near the endeffector has to be to the target point. In the following I present the the mathematical description of the CCD algorithm. See Figure 2 for a graphical explenation of the variables.

- Do this for all shapes in order from the last shape to the IK-root.
    - Calculate vector v_1, pointing from the local root of the shape to the point where the shape should actually touch the target point (the "end effector").
    - Calculate vector v_2, pointing from the local root of the shape to the target point.
    - Normalize v_1 and v_2.
    - Get the rotation angle $\alpha = \cos^{-1}(v_1 \bullet v_2)$, where $\bullet$ is the dot product.
    - Get the normalized rotation axis $r = \frac{v_1 \times v_2}{|v_1 \times v_2|}$, where $\times$ is the cross product.
    - Rotate the shape at the angle α over the rotation axis r, so that v_1 and v_2 are parallel now.
- When the IK-root is reached but the threshold is not, go back to the first processed shape and start the algorithm again.

This algorithm has to be stopped after a certain number of iterations (e.g. 20) so that it does not search endlessly for an unreachable goal, i.e. the threshold value will never shrink small enough. The IK-root is the shape to which the algorithm goes back to, e.g. the shoulder at a human model when you only want to move the arm.

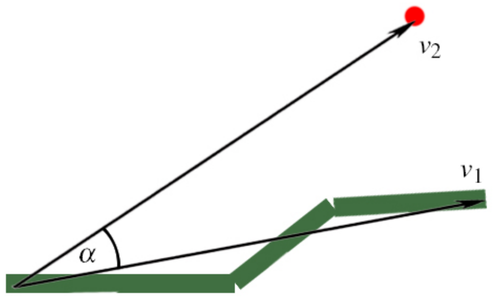

Figure 2: Drawing of the variables listed in Section 3. The processed shape in this figure is the root shape. The roots of v_1 and v_2 lie at the pivot point of this shape. The rotation axis r is orthogonal to v_1 and v_2.

4 Implementation

In this section I will describe all the different steps of my implementation of the CCD algorithm for the IK rule.
Firstly, joint limits have to be set. An ellbow for example can not be bent backwards. In our CGA Shape environment a SetJointLimits rule does that for all the required shapes.
Then the IK rule can be applied. My implementation follows the algorithm presented in section 4. The rule must be applied to the shape that shall touch the specified targetpoint. If no IK root is chosen, the algorithm goes back to the skeleton root. The maximal number of iterations and the threshold can be chosen as well. The values are 0.01 for the threshold and 50 for the maximal numbers of iterations. The user can also choose the end effector. It is defaulty at the top right corner of the shape which basically means the end effecotr equals $(1,1,1)^T$ in relative coordinates.
But this specified vector only counts for the last shape of the IK chain (i.e. the shape where the rule has been applied to) and stays not the same as we are recursively going through the shapes. So for all the shapes, except the first one, the corresponding "length vectors" have to be calculated and accumulated with the specified end effector to get v_1. All these vectors are stored in an array. If a shape gets transformed the vectors in the array need be transformed the same way as we need the exact vectors for the calculation of v_1.
The next step is to calculate v_2. The target point is specified but we need the coordinates of the targetpoint relative to the same point we calculated v_1 and the targetpoint is only given in world coordinates. To get this relative target point the position of the shape pivot point needs to be substracted from the global target point.
After obtaining v_1 and v_2 we can calculate the rotation angle α and the rotation axis r. We experienced troubles with numerical instability so it is a good idea to first make sure that $v_1 \bullet v_2 \in [-1,1]$ before applying the cos^{-1} function on it to get α.
In our CGA Shape environment rotation has been implemented using a quaternion. So in the IK rule this quaternion is now built with the given angle and axis and rotation can be applied.
Then, after a final check if the end effector is near enough to the targetpoint or if we reached the maximal number of iterations, we proceed to the next shape in the skeleton, calculate all variables anew and rotate the shape. If there is no such shape, which means we already reached the IK root, we go back to the initial shape and start the algorithm on the already deformed model again. This way the model approaches the final result step by step. A rudimentary pseudocode of our implementation

can be seen at Figure 3. It might be interesting to draw these appraoching steps but more about that in the final section about future work.

```
Var Shape; //The endeffector-shape

While(Distance>Threshold && counter < maximalIterations){
      Var depthCounter=0;
      While(Shape != null && Distance >= Threshold){
            If(counter > 0) //after first iteration take shapes from the list
                  Shape=shapesBuffer[childs];
            If(currently processed shape is the end effector shape)
                  Clear the childrenVectors list;
            Var realPos = pivot coordinates relative to kinematic root;
            Var v1 = vector from current pivot point to end effector;
            v1+= all previously stored v1-values from the childrenVectors list
            Var v2 = targetPoint - realPos;
            Normalize v1 and v2;
            Var alpha = $\cos^{-1}(v1 \bullet v2)$;
            Var r = v1 x v2;
            Normalize r;
            Rotate Shape around r with degree alpha;
            Rotate all vectors in childrenVectors list
            Add v1 in world coordinates to childrenVectors list
            Distance = v1 - v2;
            childs++;
            depthCounter++;
            if(depthCounter >= selected Depth by user) break;
            bufferShape=Shape;
            Shape=next Shape in skeleton or null if there is none;
      }
      Add bufferShape to shapesBuffer;
      counter++;
      childs=0;
}
```

Figure 3: Pseudocode of our implementation of the CCD algorithm.

4.1 IK depth

A speciality here is that the user can specify the depth of the IK chain. We decided to let the user control the depth via numbers and not via the for the user not visible unique ID of the shapes. In Figure 4 you see how the different values for the depth affect the model. These IK depth values are important if you want to move the arm of a human model. You then have to define the depth until the shoulder otherwise the algorithm would move the shapes until the skeleton root which is usually in the middle of the upper body.

5 Outview and future work

As said before the drawing of the iteration steps could be implemented in the future. Although it is not real animation (as there is no interpolation of the positions of the shapes) it will look rather well most of the time. According to Chris Hecker's lecture about inverse kinematics [Hecker 2002], the

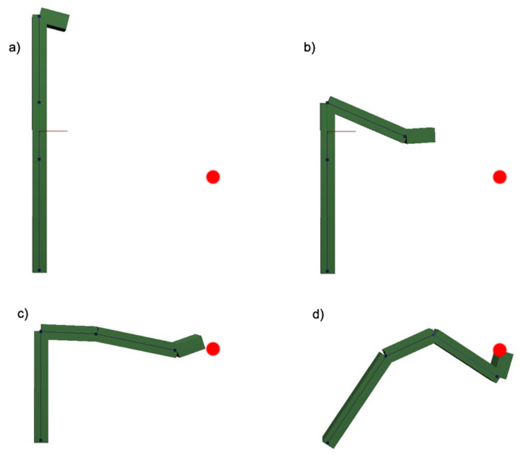

Figure 4: The different results for different IK depth values. a) 1, b) 2, c) 3, d) 0 = until the skeleton root

drawing of the outside iteration steps looks like animation because the end effector converges to the specified target point and tries to find a solution. However, to my knowledge there is no literature covering this topic.
The CCD algorithm is very stable. That means you get the same result every time you apply the IK rule. Sometimes the result might not look that well and you would want to get another angle of the shapes but still touch the target point. A solution for this problem might be to keep certain shapes in a fixed position (specified by the user) and rotate the other ones. This is already possible as we have the previously mentioned SetJointLimits rule, where you can specify that certain shapes can only be rotated to a certain degree, but it would be convinient to fixate the shapes dynamically within the IK rule.
The IK rule changes only the rotation at the moment. Where this is fine for human models it might be nice to have it change specified degrees of freedom. For example, if the size is a degree of freedom the IK rule could stretch the shape to reach an unreachable goal which seems impossible to do with CCD.

References

GRZESZCZUK, R., AND TERZOPOULOS, D. 1995. Automated learning of muscle-actuated locomotion through control abstraction. *Computer Graphics 29*, Annual Conference Series, 63–70.

HECKER, C. 2002. My adventure with inverse kinematics. *http://www.chrishecker.com*, (accessed Aug. 2010).

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3 (July), 614–623.

WAMPLER, C. 1986. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics 16*, 1 (January), 93–101.

WANG, L. T., AND CHEN, C. C. 1991. A combined optimization method for solving the inverse kinematics problem of mechanical manipulators. *IEEE Trans. Robotics and Automation 7*, 4 (Aug.), 489–499.

WHITNEY, D. E. 1969. Resolved motion rate control of manipulators and human prostheses. *Man Machine Systems, IEEE Transactions on 10*, 2, 47–53.

WOLOVICH, W., AND ELLIOTT, H. 1984. A computational technique for inverse kinematics. In *The 23rd IEEE Conference on Decision and Control*, IEEE, 1359–1363.

ZHAO, J., AND BADLER, N. I. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph. 13*, 4, 313–336.