Advanced Computer Graphics

Ikarus – Inverse Kinematics

John Bartholomew

jb5950

# Introduction

I chose to write an inverse kinematics solver/test-bed (called Ikarus). I have used the Cyclic Coordinate Descent algorithm, and have implemented root-changing and a simplistic joint constraint system. In the interests of full disclosure, I have had quite a lot of discussion with Mark Castell, who also implemented an inverse kinematics solver, however our implementations and reports are entirely separate.

# Algorithm

I implemented a Cyclic Coordinate Descent (CCD) based IK solver. CCD works by iterating over each joint along the chain from the end effector to the root of the hierarchy, and applying a rotation at that joint to minimize the distance between the end effector and the target. Visually, if you consider a vector from the joint being modified to the end effector, the rotation applied at the joint will make that vector point directly from the joint to the target. This process is repeated until the end effector is close enough to the target, or until the

# Results

# Conclusions/Analysis

# References

# Notes

- Algorithm:

--- Explain CCD

--- Explain root-changing (mention importance of data structures)

--- Explain conceptual idea of constraints

--- Explain constraint algorithm

- Results:

--- Converges on a solution very quickly in many circumstances

--- Doesn't automatically produce 'natural' poses (this would require modification of the algorithm to add extra

- Analysis/Conclusion:

--- Most IK algorithms (at least numerical algorithms) can be described in Hugo Elias's terms; N-dimensional space where N is the number of degrees of freedom; algorithms optimize some function that ranges over this space (usually distance-to-the-target-position, but it could be something else, e.g., like orientation of the end-effector). This gives a framework for comparison/analysis of the algorithms.

--- CCD works by iterating over the N dimensions, immediately moving it to its closest minimum.

--- This is actually quite a nice algorithm - deceptively fast in many situations. It's fastest

- Conclusions:

--- ???