Gesture-Based Navigation in Graph Databases
– The Kevin Bacon Game –

Felix Beier Stephan Baumann Heiko Betz Stefan Hagedorn
Ilmenau University of Technology, Germany
first.last@tu-ilmenau.de

Timo Wagner
Objectivity, Inc.
twagner@objectivity.com

Abstract: Motion sensing devices like Microsoft's Kinect offer an alternative to traditional computer input devices like keyboards and mouses. Graph databases can naturally make use of gesture control as traversing graphs can easily be described by swiping or pointing gestures. In our demo we traverse the Internet Movie Database (IMDB) using a Kinect interface with the control logic in our data stream engine AnduIN. The gesture detection is done based on AnduIN's complex event processing functionality.

1 Introduction

Research has shown that more than 60% of the interpersonal information exchange are passed non-verbal. New hardware achievements like gesture detecting devices as Microsoft's Kinect [SFC$^+$11] moved gestures as a new input alternative into the focus of research and industry. Also graph databases recently attracted interest again, not only driven by social networks, but also by applications in the fields of linked and big data. In contrary to SQL systems, they offer natural ways for user interaction, e.g., traversing nodes and edges can be intuitively expressed with hand movements. In our demonstration we show how gesture based interaction and graph databases can be combined. We address the following issues:

- Exploiting online techniques for complex event processing (CEP) implemented in our data stream engine AnduIN [KKH$^+$11] for gesture recognition in sensor data which is provided by a motion sensing input device as the Kinect camera.
- Using this solution to build a gesture controlled interface for a graph database which enables the user to navigate through the data with natural movements.

We have previously used gesture detection to navigate an OLAP cube [HSS$^+$12]. There we used gestures to execute for example rotate, drill down or role up operations. A link to a video of this demo is provided in the references.

The rest of this paper is structured according to our software architecture as shown in Figure 1. First, we provide a short overview of the Kinect and its drivers. Next, we introduce the InfiniteGraph graph database, followed by an introduction of the main gestures used for interaction. The paper closes with a description of the demonstration.

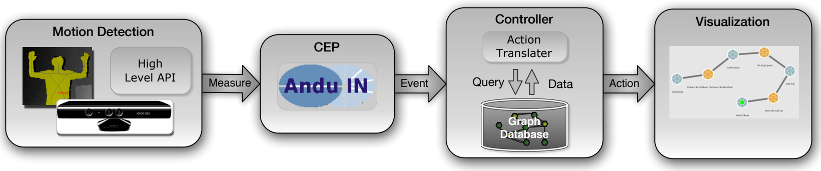


Figure 1: Software architecture. Motion sensors deliver a continuous stream of body positions. AnduIN's complex event processing (CEP) engine is used for detecting gestures in this sequence of poses. The gesture descriptions are passed to the graph DB wrapper which translates them into database queries for updating the controller's state and to visualize query results.

2 Gesture Detection using the Kinect System

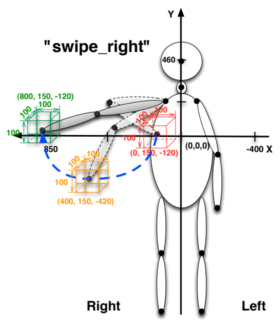
The Kinect camera comprises three types of sensors: a real-time infrared depth camera, a simple color camera, and multiple microphones. The cameras deliver 640x480 pixel images with a centimeter resolution at a rate of 30 frames per second. The sensors work at distances from 0.8m to 3.5m.

For communication with the Kinect, we used the OpenNI framework [Ope]. It implements several middleware components and provides interfaces for sensors as well as for applications. Components are registered in the form of *modules* and can be used for accessing the sensor values directly, or at a higher abstraction level as provided by NITE [NIT]. NITE implements modules for full body analysis and realizes the tracking of a human in form of a skeleton which consists of 15 characteristic joints like head, left/right hand, etc. The 3D coordinates of each skeleton joint are calculated every ten milliseconds using the 2D color image in conjunction with the depth image. For example, the coordinates of the left foot can be accessed with the following statements:

```
XnSkeletonJointPosition joint;
g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition(
     playerID, XN_SKEL_LEFT_FOOT, joint);
```

3 InfiniteGraph as a graph database

Our demonstration is based on InfiniteGraph [IG], developed by Objectivity Inc., as one representative of (commercially available) graph database systems. InfiniteGraph is a schema based graph database, that defines its schema using Java objects. Database management and access is provided by the class `GraphFactory`. Base classes for nodes and edges are provided, i.e. `BaseVertex` and `BaseEdge`, from which a user can derive application-specific subclasses. InfiniteGraph offers a transactional concept to insert, delete, or modify nodes or edges. A given graph can be traversed or searched. For this, methods to retrieve connected edges or neighboring nodes and such are provided in the base classes. Searching is based on a `Guide` that defines the order in which nodes are traversed. InfiniteGraph supports indexing according to class properties. In our demonstration we will focus on the traversal of a given graph. Although, we implemented our demonstration for InfiniteGraph, the approach is generally transferrable to similar systems like neo4j, where similar interfaces are available.



```
SELECT "swipe_right"
MATCHING (
  kinect(
    abs(right_hand_x - torso_x - 0) < 50 and
    abs(right_hand_y - torso_y - 150) < 50 and
    abs(right_hand_z - torso_z + 120) < 50
  ) ->
  kinect(
    abs(right_hand_x - torso_x - 400) < 50 and
    abs(right_hand_y - torso_y - 150) < 50 and
    abs(right_hand_z - torso_z + 420) < 50
  ) within 1 seconds select first consume all
) ->
kinect(
  abs(right_hand_x - torso_x - 800) < 50 and
  abs(right_hand_y - torso_y - 150) < 50 and
  abs(right_hand_z - torso_z + 120) < 50
) within 1 seconds select first consume all;
```

Figure 2: “Swipe_Right” gesture for expanding a selected edge

4 The Controller and Database Interaction

After identifying gestures based on events by the AnduIN system, it is the controller’s task to derive queries for the graph database based on the actions and the current state and trigger adequate queries for the graph database.

For this demonstration we have implemented a basic set of gestures/queries on the InfiniteGraph database focusing on graph navigation. Each detected gesture triggers a new query and results are displayed on the screen. The gestures are

- select edge/node - the user can select one of the currently visualized nodes or edges
- about edge/node - the user can request additional information about a node or edge, i.e. get the properties
- undo/redo - during a graph traversal, undo and redo of moves are enabled
- shortest path - the user can query the database for the shortest path between two present nodes
- solve - triggers a shortest path query between a start and end node

For example, a selected *Person* node of “Johnny Depp” is shown in Figure 3. The visualized graph can be expanded further by selecting an outgoing edge and expanding it with a “Swipe_Right” gesture. The gesture and its corresponding AnduIN CEP translation is illustrated in Figure 2. Once, the gesture is detected by AnduIN, the new node is requested from the InfiniteGraph database with *getNode(ID)*, where ID corresponds to the outgoing selected edge (comp. “The Ginger Man” in Figure 3), and is derived by the controller from the preceding select edge gesture.

5 Demonstration

For the demo session, we plan to show our prototype system in action. We will bring a notebook equipped with a Kinect camera running our InfiniteGraph interface for IMDB. Everyone can try to navigate the database by gestures playing the Kevin Bacon [Kev] game. The game is simple, one starts out with a node representing the actor Kevin Bacon and traverses the graph through gesture navigation to find the shortest path to another given actor (Figure 3). The length of the path is then the Kevin Bacon number of the other actor.

For the database we use a simple data model. There are two node types, `Person` and `Project`, and one edge type `workedOn`. `Person` contains name and gender of an actor, `Project` contains name and release year of a movie. `workedOn` contains the name of the role an actor played in a movie.

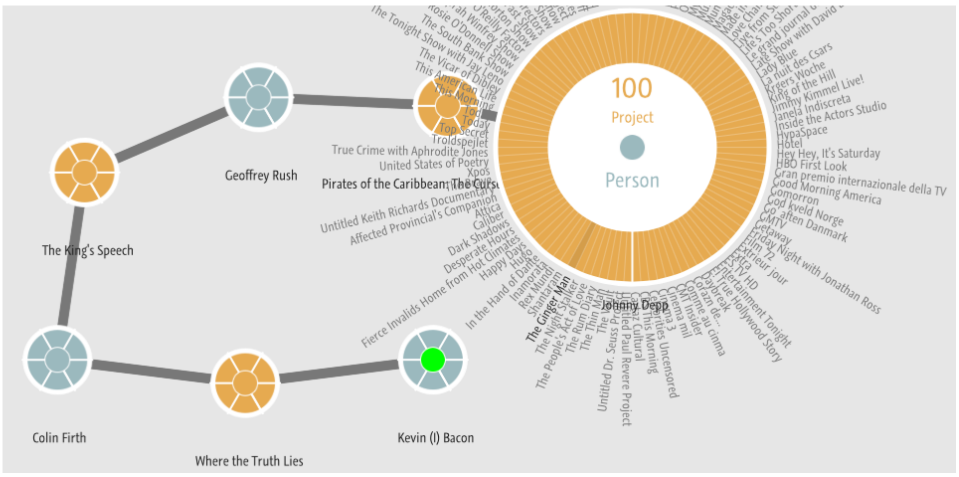


Figure 3: Screenshot of the demonstration showing a path between Kevin Bacon and Johnny Depp.

For a user playing the game, this means he or she has to select a node, query its properties, check for its outgoing edges and decide which next node, i.e. movie or actor, to choose. In addition, we allow a glance into the internals of the overall system. Raw events generated by the Kinect camera are passed to our stream engine AnduIN, which performs the gesture detection. Due to the usage of AnduIN's CEP functionality, necessary gestures can be exchanged on the fly. To show this effect, we prepare different gestures for a single interaction with the database. Additionally, the usage of the simple SQL like declarative user interface offers an easy way to define new gestures. After a short briefing, participants should be able to define there own gestures and apply them instead of the original ones. This way it will be possible for users to customize UI controls for traversing the graph.

References

[HSS+12] Steffen Hirte, Eugen Schubert, Andreas Seifert, Stephan Baumann, Daniel Klan, and Kai-Uwe Sattler. Data3 - A Kinect Interface for OLAP using Complex Event Processing. In *ICDE*, April 2012. Video: http://youtu.be/DROXI0_wDRM.

[IG] InfiniteGraph. http://objectivity.com/INFINITEGRAPH.

[Kev] The Kevin Bacon Game. `http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon`.

[KKH+11] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K. Sattler. Stream Engines Meet Wireless Sensor Networks: Cost-Based Planning and Processing of Complex Queries in AnduIN, Distributed and Parallel Databases. *Distributed and Parallel Databases*, 29(1):151–183, January 2011.

[NIT] NITE Middleware. `http://www.primesense.com/?p=515`.

[Ope] OpenNi. `http://www.openni.org/images/stories/pdf/OpenNI_UserGuide_v3.pdf`.

[SFC+11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. *Computer Vision and Pattern Recognition*, June 2011.