MagicVR - Gesture Recognition

Jerome D. Pönisch*
3D Models and Effects

Michael Maier†
Gesture Recognition

ZUSAMMENFASSUNG

This documentation is about MagicVR - a virtual reality application developed in OpenSG for the LRZ V2C in Garching supporting 3D-Gesture-Recognition and Lerp-Animations. This document covers the part 3D-Gesture-Recognition.

Index Terms: V2C—Virtual Reality—OpenSG—Treemaps

1 INTRODUCTION

In this documentation the concept and implementation of MagicVR gestures will be explained. MagicVR is a virtual reality application which was developed in C++ and OpenSG in order to run in the V2C mixed reality cave of the LRZ in Garching, Munich. It's main feature is a 3D-Gesture-Recognition opening a whole world of interaction possibilities using only the wand (or any tracker) in a 3D tracking system.

2 CONCEPT

MagicVR was developed in a cooperation between Jerome Pönisch and Michael Maier. This paper will be focused on the 3D-Gesture-Recognition done by Michael Maier while Jerome Pönisch's paper will be focused on the 3D Art and FractTime-Animation effects.

2.1 The idea behind MagicVR

The main idea of MagicVR is introducing the 3D-Gesture-Recognition module in a playful environment using important features of a virtual environment. A user/player should be animated to use 3D-Gestures as interesting interaction technique rather than pushing buttons. The virtual environment should help to gain the user/player's interest and make him feel comfortable while experiencing the mixed reality world. As User-Feedback it was decided to focus on animations and movements within the scene.

2.2 Movement Representation

A movement is represented by a trajectory. Fig. 3 shows examples of 2D trajectory definitions. The same can be done for 3D space. I use 2D trajectories in the trajectory examples of this paper because they are easier to visually perceive and compare.

Trajectory is a curve defined by a finite sequence of points with linear interpolation between consecutive points.

2.3 Recording Movements

In MagicVR the user plays a wizard that can do magic tricks by moving his magic wand. Therefore, the position of the wand is tracked over time to get a trajectory that represents the movement of the wand. Those movements are used to interact with the scene of the application. Only specific movements can trigger special events. The scene (see Fig. 1) contains four pedestals with a stone up on them. On each stone is a different symbol. If the user draws the symbol, an animaion is shown (see Fig. 2). To prevent the user from unintentionally triggering events by accidentally doing a specific movement, the user has to press the button on the backside of the wand. The movement is only recorded while the button is pressed. Thereby, only trajectories are recorded that are intended by the user.

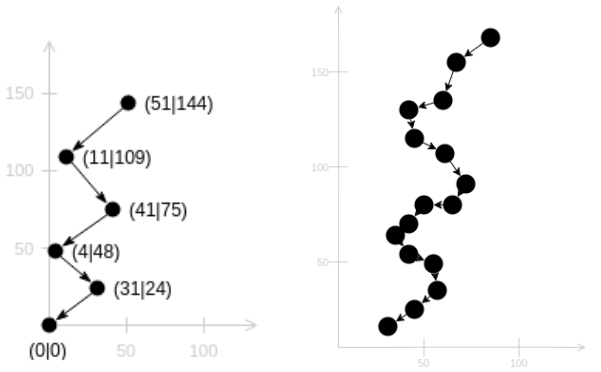
*e-mail: jerome.poenisch@campus.lmu.de
†e-mail: maier.michael@campus.lmu.de

Abbildung 1: The scene shows Stonehenge with a fireplace and four pedestals. On each pedestal is a stone with a symbol that represents an element (from right to left: Fire, thunder, water, wind).

Abbildung 2: The user has to draw the symbol with the wand to activate the kind of magic of the respective element. The photo shows the scene after various magic spells have been executed.

2.4 Definition Of Gestures

A gesture describes a movement. It is defined using a trajectory that describes the movement, preprocessing functions that transform trajectories and a comparison function that checks if two trajectories are similar.



(a) Pattern trajectory (b) Recorded trajectory

Abbildung 3: A pattern trajectory describes the ideal path of a movement. It is very unlikely that exactly the same trajectory is drawn and recorded.

2.5 Detection Of Gestures

Input trajectory (recorded movement) and pattern trajectory (description of a specific movement) are preprocessed and compared by distance to each other. Dependent on the result, a function is called to process the result (change the state of the application and trigger events (show effects)). Fig. 4 shows a diagram of this process.

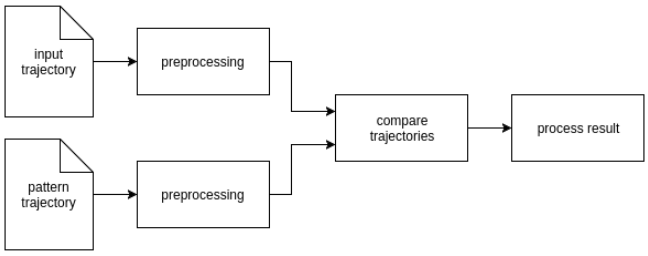


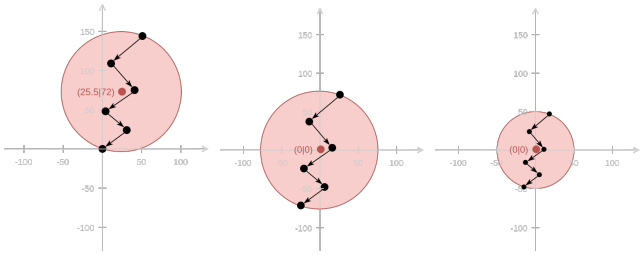
Abbildung 4: System diagram

2.6 Preprocessing

You can see in Fig. 3 that most commonly input and pattern trajectory differ in the number of points, length/size, position, etc. Distance functions may assume specific preconditions. For example, some require that both trajectories have equal number of points or length. The goal of the preprocessing (example shown in Fig. 5) is to prepare a trajectory for comparison with another trajectory (see Fig. 6). The appropriate preprocessing steps depend on the same factors as the limit that is used for Deciding Similarity including the preprocessing itself in that way that the preprocessing steps influence each other. It is important in the process shown in Fig. 5 to translate the trajectory to the origin before it is scaled.

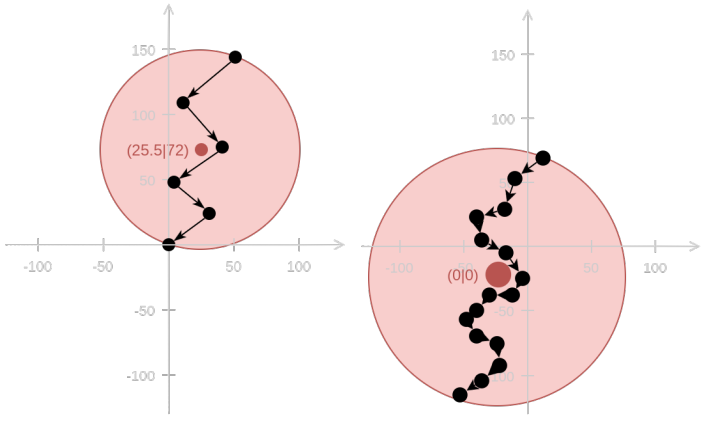
2.7 Comparing Trajectories

To check two trajectories for similarity we use a distance function to calculate the distance between them. The distance is a scalar that indicates how similar two trajectories are. We can decide similarity based on a limit. If the distance is lower than the limit we consider two trajectories to be similar.

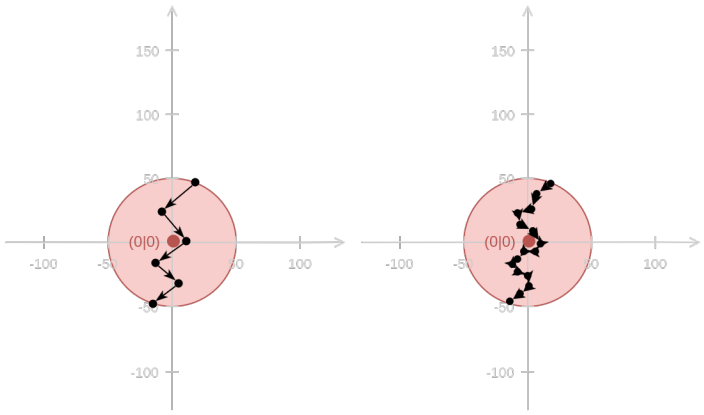


i Calculate Minimum Bounding Sphere ii Translate iii Scale

Abbildung 5: (i) The smallest circle (hyper sphere) that contains all points of the trajectory is calculated. (ii) Then the trajectory is moved so that the center of this circle is in the origin. (iii) After that the trajectory is scaled so that the diameter of the circle is 100.



i Trajectories before preprocessing (pattern on the left, input on the right)



ii Trajectories after preprocessing (pattern on the left, input on the right)

Abbildung 6: (i) The original trajectories differ in length/size and position. (ii) After the preprocessing they have similar length/size and the same position.

2.7.1 Distance Function

There is a wide range of distance functions to choose from. Each has its pros and cons. Good results are obtained using a modified Hausdorff distance function [1](see Fig. 7).

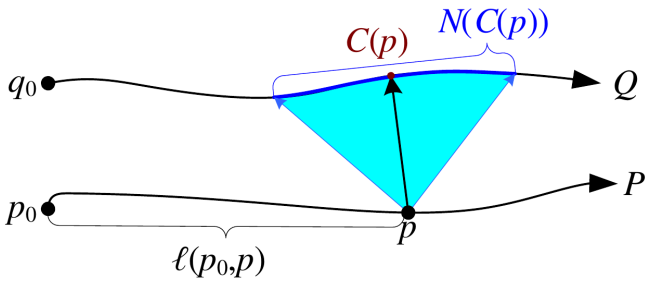


Abbildung 7: The modified Hausdorff distance function [1]is a variant of the Hausdorff distance that takes the order of the points into account. It runs over all discrete points of both trajectories. The unmodified version of the Hausdorff distance determines the distance between each point and the other trajectory. The modified variant, on the other hand, only takes a subtrajectory - here $N(C(p))$. For this purpose, the point $C(p)$ on the other trajectory is determined for each point p. $C(p)$ is the point that lies at the same position relative to the length of the trajectory. $N(C(p))$ defines the neighborhood of $C(p)$. These are all points which are only a certain distance away from $C(p)$ along the trajectory. The minimum distance to p is determined for this subtrajectory. I use the distance from point to distance, i. e. I consider the subtrajectory as a set of distances. So we now have a lot of space for each point of the discrete trajectory. The maximum of these distances is then the distance between the two trajectories. The order of the points is taken into account by $C(p)$.

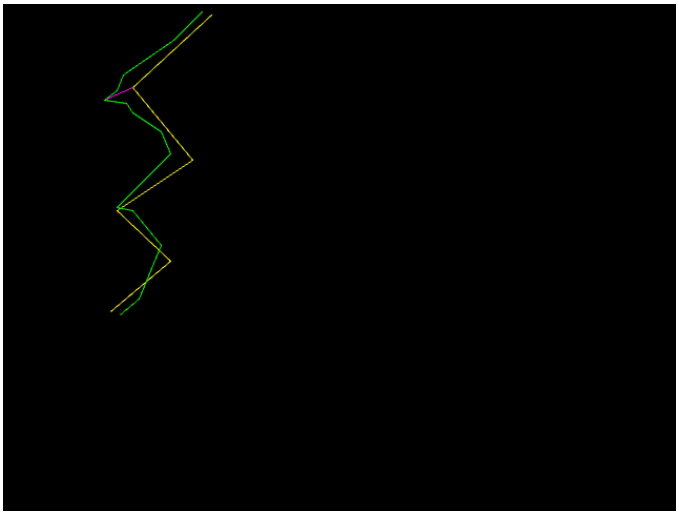
2.7.2 Deciding Similarity

We consider two trajectories to be similar if the distance is lower than the limit (see Fig. 8). Finding a good limit is crucial to avoid/reduce false positives and negatives. If the limit is not tolerant enough, measuring inaccuracy of the input device recording the object position and natural small deviations during each movement execution will make it hard or impossible to recognize a movement. If the limit is too tolerant we consider trajectories to be similar that should not.

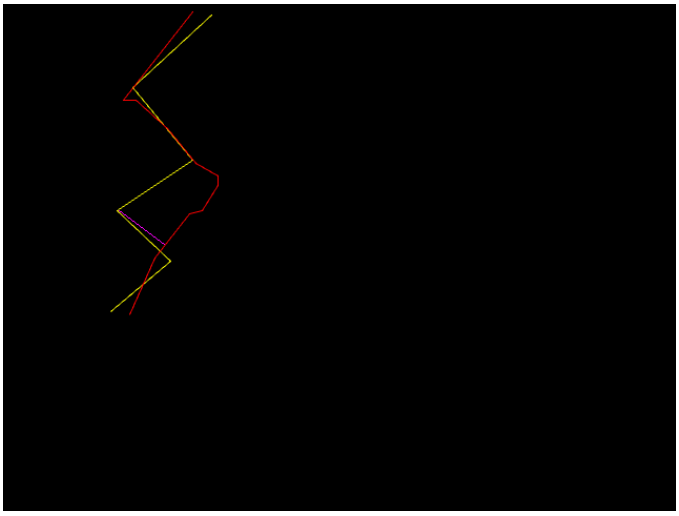
Another important factor is the preprocessing of the trajectories. Each preprocessing step might influence the distance (positive or negative). The influence depends on the distance function. Please also note that the distance might not grow linear. For example, some distance functions use squared values. Moreover, you should devote special attention to the scale of the trajectories, since the distance itself depends on it.

Taken together, the limit depends on

- the measuring (in-) accuracy of the input device
- the minimum movement execution accuracy we claim from the user
- the preprocessing of the trajectories
- the scale of the trajectories
- the distance function



i Similar trajectories



ii Dissimilar trajectories

Abbildung 8: Here you can see two screenshots of a 2D demo application of trajecmpthat visualizes the comparison of two standardized trajectories. The yellow trajectory represents the lightning pattern. (i) shows a green trajectory, which has been classified as similar. In (ii) you can see a red trajectory, which has been classified as dissimilar. The pink line represents the distance that has been calculated using the modified Hausdorff distance function [1].

3 Implementation

The project is implemented in C++ using the libraries Boost Geometry, RxCpp, range-v3and trajecmp. The library trajecmpis written by the author Michael Maier to detect gestures. The process shown in Fig. 4 is implemented like this:

```
const auto result_stream =
    compare(preprocess(input_stream),
            preprocess(pattern_stream));

result_stream.subscribe([](auto result) {
    // process result
});
```

`input_stream` and `pattern_stream` are streams of trajectories. Every time a input trajectory is emitted on `input_stream` it is preprocessed and compared with the (latest) preprocessed pattern trajectory of `pattern_stream`. Usually, `pattern_stream` contains only one (static pattern) trajectory. However, a pattern trajectory might be dynamically created. For example, you could use another input trajectory as a pattern.

You can compare the input trajectory with multiple patterns like this:

```
const auto preprocessed_input_stream =
    preprocess(input_stream);

const auto result_1_stream =
    compare(preprocessed_input_stream,
            preprocess(pattern_1_stream));

const auto result_2_stream =
    compare(preprocessed_input_stream,
            preprocess(pattern_2_stream));

result_1_stream.subscribe([](auto result) {
    // process result
});

result_2_stream.subscribe([](auto result) {
    // process result
});
```

All gestures of the project are implemented in the class `sources/magicvr/MagicTricks.cpp`. Apart from the preprocessing steps shown in Fig. 5 it contains a rotation transformation around the axis pointing upwards in the cave. The event subscription is done in `sources/magicvr/AppController.cpp` and `sources/magicvr/AppControllerWithWandSupport.cpp`.

4 Gestures

In this section the gestures are described. The effect of each gesture is described in Jerome's documentation.

4.1 Fire

The fire gesture (see Fig. 9) is a flame symbol drawn from left to right. It is defined by this trajectory:

```
Trajectory{
    {0,  0, 0},
    {-1, 3, 0},
    {1,  2, 0},
    {2,  5, 0},
    {2,  5, 0},
    {3,  2, 0},
    {5,  3, 0},
    {4,  0, 0},
}
```

Abbildung 9: Fire gesture

4.2 Water

The water gesture (see Fig. 10) is a wave symbol drawn from left to right. It is defined by this bezier curve:

```
const BezierCurve<> waterBezier{
    {0,    0,  0},
    {1.5f, 3,  0},
    {1.5f, -2, 0},
    {3,    1,  0},
};
```

Abbildung 10: Water gesture

4.3 Lightning

The lightning gesture (see Fig. 11) is a lightning symbol drawn from top to bottom. It is defined by this trajectory:

```
Trajectory{
    {1, 2, 0},
    {0, 1, 0},
    {1, 1, 0},
    {0, 0, 0},
}
```

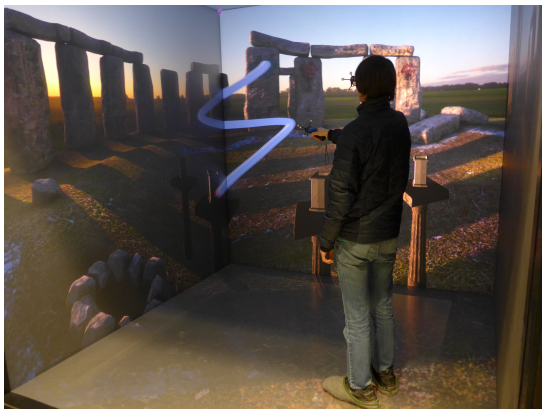


Abbildung 11: Lightning gesture

4.4 Wind

The wind gesture (see Fig. 12) is a spiral drawn from the inside to the outside. The trajectory is calculated in the method `sample` of the class `sources/magicvr/Spiral.cpp`.

Abbildung 12: Wind gesture

4.5 Circle

The circle gesture (see Fig. 13) is a single circle drawn from the left side of the center to the top. The trajectory is calculated in the method `sample` of the class `sources/magicvr/ranges/view/Circle.cpp`.

```
Circle(1).sample(0, 360, 10)
```

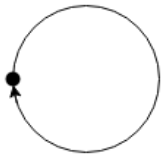


Abbildung 13: Circle gesture

4.6 Multiplication

The multiplication gesture is a Circle gesture with two circumnavigations.

```
Circle(1).sample(0, 720, 10)
```

4.7 Shoot

The shoot gesture (see Fig. 14) is a quarter segment of a circle drawn from above with a circumnavigation direction to the right (in the style of a whip movement).

```
const BezierCurve<> quaterCircleFromAbove{
        {0, 1, 0},
        {1, 1, 0},
        {1, 0, 0},
        {1, 0, 0},
};
```

Abbildung 14: Shoot gesture

5 Outlook

All gestures used in MagicVR are drawn in a plane. It is possible to define gestures that are not in a plane, but the movement would be harder to explain to a user. The symbols on the elemental stones shows the 2D trajectory that should be drawn in 3D space. A visualization of a 3D trajectory would require another user guidance. Our idea is a little fairy which gives hints about interaction possibilities and demonstrates according gestures by flying them in front of the user.

There are unlimited possibilities to define more gestures that interact with the scene. For example, a gesture could trigger different actions dependent on the position and orientation it is drawn. If you do a gesture in front of an object you could trigger an interaction on the object. One idea is to let an object (for example the lantern fly) if you do a wind gesture in front of it.

Literatur

[1] S. Atev, O. Masoud, and N. Papanikolopoulos. Learning traffic patterns at intersections by spectral clustering of motion trajectories. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4851–4856, Oct 2006. doi: 10.1109/IROS.2006.282362