Proceedings of Third International Symposium on Smart Graphics 2003

Dynamic Glyphs – Depicting Dynamics in Images of 3D Scenes

Marc Nienhaus and Jürgen Döllner

Hasso-Plattner-Institute
at the University of Potsdam,
Department for Computer Graphics Systems,
Prof.-Dr.-Helmert-Str. 2-3,
14482 Potsdam, Germany
{nienhaus, doellner}@hpi.uni-potsdam.de
http://www.hpi.uni-potsdam.de/deu/forschung/cgs

Abstract. Depicting dynamics offers manifold ways to visualize dynamics in static media, to understand dynamics in the whole, and to relate dynamics of the past and the future with the current state of a 3D scene. The depiction strategy we propose is based on visual elements, called dynamic glyphs, which are integrated in the 3D scene as additional 2D and 3D geometric objects. They are derived from a formal specification of dynamics based on acyclic, directed graphs, called behavior graphs. Different types of dynamics and corresponding mappings to dynamic glyphs can be identified, for instance, scene events at a discrete point in time, transformation processes of scene objects, and activities of scene actors. The designer or the application can control the visual mapping of dynamics to dynamic glyphs, and, thereby, create own styles of dynamic depiction. Applications of dynamic glyphs include the automated production of instruction manuals, illustrations, and storyboards.

1 Introduction

Depicting dynamics represents a challenging task for smart graphics: It is a powerful tool, deployed in arts and science yet for a long time as technique to illustrate dynamics of actors, objects, and processes in static media. As underlying principle, illustrations encode in images more than 3D scenery – abstract elements, for instance, arrows indicating a direction of movement, rays symbolizing an extraordinary event, or clouds containing descriptions of thoughts of an actor. This way, depictions of dynamics in images enable observers to understand dynamics of 3D scenery even in static images, to relate dynamics of the past and the future with the current state of a 3D scene, and to communicate all kinds of non-geometric information such as tension, danger, and feelings.

3D computer graphics provides a wealth of modeling and rendering techniques, which represent the technical basis upon which smart graphics technology can be built. In that direction, we propose a concept for augmenting images of 3D scenes by visual elements that abstract and symbolize dynamics (s. Fig 1). It is (1) based on the

formal hierarchical specification of dynamics by *behavior graphs*; (2) performs the visual mapping of behavior graphs to *dynamic glyphs* for a given point in time or time interval; and (3) uses *non-photorealistic rendering* as an appropriate rendering style to produce illustrations of 3D scenes and their dynamics.

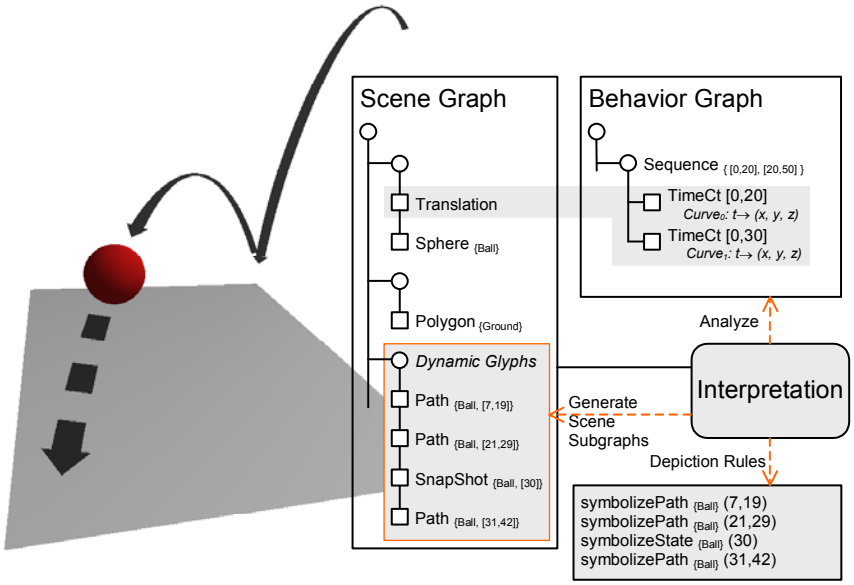

Figure 1: Screen aligned arrows as dynamic glyphs depict the animation of a ball bouncing on the ground (left). The conceptual structure to generate dynamic glyphs is given in a diagram (right). After analyzing the behavior graph, depiction rules build scene subgraphs that represent dynamic glyphs.

The concept of depicting dynamics in images of 3D scenes can be applied to many application areas. For instance, it can be used to implement a system for digital 3D storyboards generating a collection of representative images of 3D scenes with visually encoded dynamics summarizing part of a story. Another application area includes systems for producing illustrations of instruction manuals in an automated way. Furthermore, the concept can be applied to generate visual indices of linear media such as movies.

2 Specifying Dynamics

To specify dynamics of a 3D scene in a formal way, we construct behavior graphs. A behavior graph is a directed acyclic graph (DAG) that specifies time-dependent and event-dependent behavior of scene objects. A scene graph, in contrast, formally specifies geometry and appearance of scene objects [1].

Nodes of behavior graphs manage time layouts, calculate lifetimes, and control time assigned to child nodes. In addition, they maintain time-dependent constraints for elements of associated scene graphs.

Of course, both scene graphs and behavior graphs are tightly related: Scene graph elements define visual aspects of scene objects, whereas behavior graph elements define dynamic aspects of scene objects. In general, a single scene graph may have a number of behavior graphs associated with it.

2.1 Time Moments and Time Requirements

We define the following temporal abstract data types to manage the time flow in behavior graphs:

- **Moments.** A *moment* $M = (t_0, t_1, t)$ represents a point t in a time interval $[t_0, t_1]$. A moment assigned to a behavior node determines the node's lifetime interval and the current point in time within this interval. Moments are essential for behavior nodes that specify processes. Based on the knowledge about their lifetime, behavior nodes can plan their activity. The point in time t contained in a moment M communicates the current time to behavior nodes.
- **Time Requirements.** A *time requirement* $R = (T_{natural}, T_{max}, T_{min}, A)$ describes the time demand of a behavior node. It consists of the natural (i.e. desired, optimal) duration $T_{natural}$, the minimal duration T_{min}, and the maximal duration T_{min}. A time requirement can also specify that the natural duration is infinite. Furthermore, it defines a time alignment A, which determines how to position a shorter moment within a longer moment. With $A = 0.0$, the shorter moment starts at the same time as the longer moment; $A = 1.0$ causes both moments to end at the same time; and $A = 0.5$ centers the shorter moment within the longer moment.

Behavior nodes do not include time requirements by default. We add these requirements by a special kind of behavior nodes in the behavior graph, called *time setters*, or calculate them implicitly through *behavior groups*.

2.2 Managing Lifetimes

One of its fundamental tasks of a behavior graph is to define lifetimes of activities and point in times of events. To organize the overall structure of the time flow, we set up time groups, whereas the local time flow can be modified by time modifiers. We are going to explain both types of behavior nodes in the following.

2.2.1 Time Groups

Time groups represent a major category of nodes that constitute a behavior graph. A time group node calculates the individual lifetimes of its child nodes based on the children's time requirements and its own time-layout strategy. If a time group node receives a time event, it checks which child nodes to activate or deactivate. It synchronizes all active child nodes to the new time, and assigns new moments to them.

A number of time group nodes implement specific time layouting strategies. Examples for those classes include:

- **Time Sequence.** Defines the total time requirements as sum of the time requirements of its child nodes. It distributes a received moment proportionally to the child nodes. The moments assigned to the child nodes are sequential and disjoint. Only one child node is alive at any given time during the lifetime of the sequence.
- **Time Simultaneity.** Defines the total time requirement as the maximum of the time requirements of the child nodes. It distributes a received moment to the child nodes if their natural duration is equal to the duration of the moment. If not, the simultaneity layout tries to shrink or stretch the time requirements of the child nodes to fit the duration. If they still do not match it aligns the lifetime of the child nodes within the moment.
- **Time Table.** Defines for each child node an explicit time requirement. It manages activation and deactivation of child nodes according to it own lifetime.

Since time groups automate distribution and alignment of time intervals, designers are relieved from calculating absolute times in specifications of dynamics. Time groups also take care of the discrete nature of points in time. If a new point in time is reached, they take care not to forget discrete events being scheduled for the past time interval. As main feature, time groups facilitate hierarchical specifications of activities and events at a high level of abstraction similar to specifications in storybooks.

Fig. 2 shows how time groups can compose activities, and how time requirements get evaluated. The behavior nodes A_1, A_2, and S_i are processed sequentially. Behavior node S_i consists of two simultaneous behavior nodes, A_{3a} and A_{3b}. D_1, D_2, and D_3 define infinitely stretchable time requirements of 1, 2, and 1 seconds; they are not shrinkable. The sequence S_e is prefixed with a duration D of 100 seconds. If S_e actually gets from its parent 100 seconds, it distributes this moment proportionally to its child nodes, i.e. A_1 and S_i get 25 seconds each, and A_2 50 seconds. Since A_{3a} can last at most (1+14) = 15 seconds, S_i centers the lifetime of A_{3a} within the 25 seconds. S_e activates in turn A_1, A_2 and S_i, A_{3a} and A_{3b} are activated by S_i.

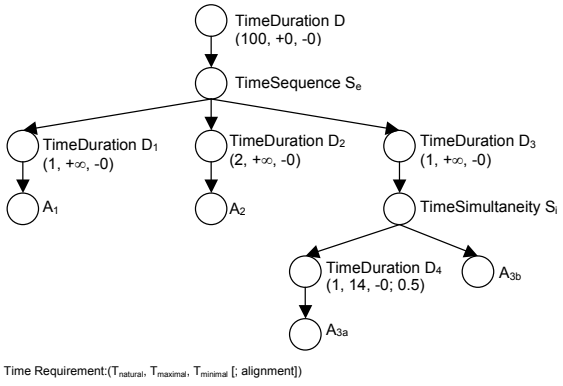

Figure 2. Example of a behavior graph.

2.3 Activities and Events

Constraint nodes represent activity and events as core elements of dynamic specifications. A generic class takes care of most variants of constraints:

- **TimeCt.** Associates a time-to-value mapping with a property of a scene object. For example, a time-constraint can constrain the position of an object associating a time-to-vector mapping with the object's midpoint.

Technically, constraint nodes control time-varying parameters of objects contained in scene graphs, for instance, position, direction, color, or size of a scene object. For each of the time-varying parameters, constraint nodes require a time-to-parameter mapping. Whenever a constraint node receives a new moment, it calculates new parameter values, and assigns these values to its constrained scene objects. A number of classes implement specific time-to-parameter mappings. Examples for those classes include:

- **Constant Map.** Assigns a constant value to constrained objects.
- **Linear Map.** Assigns a value that results from linear interpolation of specified values.
- **Curve Map.** Assigns a value that results from calculating a point of a parameterized curve by interpreting time as curve parameter.
- **Method Map.** Assigns a value that results from a method call.
- **Function Map.** Assigns a value that results from a function call.

2.2.2 Modifying Local Time Flows

As additional building blocks to specify dynamics, we define a number of time modifier nodes, which transform local time received by child nodes. Mathematically, these nodes define a time-to-time mapping that is applied to all moments passed through the node. Among the kinds of transformations (s. Fig. 3) are:

- **Constant Transformation.** It assigns a constant time to its child nodes regardless of the time progress communicated to the time modifier node.
- **Discrete Transformation.** It defines a conceptually discontinuous time progress. The available lifetime interval is decomposed in piecewise constant time intervals. Using this kind of time modifier, for instance, jerky movements can be modeled.
- **Repeat Transformation.** It maps a moment modulo a time interval, and passes the resulting moment to its child nodes. For example, to model an activity that lasts 5 seconds and that should be repeated permanently, we specify an infinite duration followed by a time repeater with the modulo moment [0, 5sec].
- **Reverse Transformation.** It inverts the direction of the time progress for the child nodes. Time reversal nodes are useful to model invert activities (provided that the underlying time constraints are invertible).
- **Creep Transformation.** It defines a creeping time progress, i.e., the time progress is slow in the beginning and speeds up at the end of a time interval. Alternatively, the time progress can be speed up in the beginning and slow down at the end.

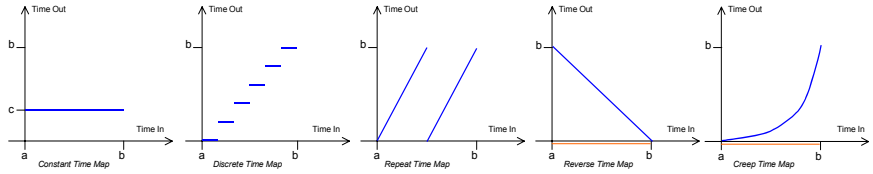

Figure 3. Examples of time-modifier functions.

3 Dynamic Glyphs

Dynamic glyphs are visual elements that symbolize dynamics in images of 3D scenes. They do not represent original scene elements, and hence are not modeled as part of the scene by the scene designer. Instead, dynamic glyphs result from analyzing and interpreting behavior graphs for a specified time interval, and the subsequent visual mapping of the results to graphical elements. These elements are inserted as part of the scene graph automatically in order to augment the image by the abstract representation of dynamics found in the behavior graph.

For analysis, we traverse the behavior graph searching for behavior nodes that are active in the specified time interval. Behavior nodes that do not become active in the specified time interval are ignored in the following. The remaining nodes can be interpreted at different levels of abstraction.

Low-Level Depiction Rules. At the lowest level of abstraction, we can interpret each single node for itself, and we can at least determine for each node the set of geometries in the scene graph that are modified by the node's activities.

For example, if an active time-constraint node is encountered that specifies a time-dependent function applied to a translation object in the scene graph, we can conclude that all geometries affected by the translation will be moved. For them, we can depict the path they will follow by creating an additional visual element, called path. A path has the shape of a flexible 3D arrow, represented graphically as quad-strip, and oriented towards the viewer to ensure full visibility (s. Fig. 1). If the time-to-vector function of the constraint node is based on a parameterized curve, for example, the path will resemble the shape of the curve (while the curve will never be visualized because it is used only to calculate object positions). The path type represents a prototypic dynamic glyph: It is a 3D shape whose configuration, appearance, and position is set up automatically based on the kind of constraint and its time-to-value mapping. It also becomes part of the scene graph without being a scene object.

The way encountered constraint nodes are interpreted can be graphically designed arbitrarily. The aforementioned time constraint associated with a time-dependent function and a translation object could also be visualized by indicating the followed path using 3D points. Technically, a dynamic glyph is encoded by a scene subgraph, which can take advantage of all defined shapes, graphics attributes, and containers like the main scene graph can do.

A formal definition of the general visual mapping of time constraints at the lowest level of interpretation can be given at this point. It maps the triple consisting of time constraint, its time map, and its constrained scene objects to a dynamic glyph.

Higher-Level Depiction Rules. We can also detect patterns in encountered behavior nodes that indicate dynamics at a higher level of abstraction. Mappings based on detecting patterns are applied prior to lower-level mappings. They aim at symbolizing complex dynamics.

For example, if a scene object is animated by a tailspin, a simultaneity group having two child nodes, one for constraining its position and one for constraining its rotation angle, encodes this kind of dynamics in a behavior graph. It is symbolized by a single dynamic glyph, a twisted path.

Integrating Dynamic Glyphs and Scene Graphs. After analyzing behavior graphs and applying higher-level and lower-level interpretations, we can add the resulting scene subgraphs to the main scene graph. For different time intervals being considered for analysis, dynamic glyphs are added to or removed from the scene graph, respectively for image rendering. We assume that interpretation of behavior nodes as well as scene subgraphs representing dynamic glyphs are subject to a user-controlled design process in order to fine-tune the visual appearance. Of course, these processes can also be automated as well if interpretation schema and dynamic glyph designs are fixed for a given category of scene objects and related dynamics, for example, instructions for assembling furniture parts.

4 Rendering Scenes with Dynamic Glyphs

In general the images we are interested in outline and sketch complex scenarios and their related dynamics. Therefore, to achieve abstraction is essential for images of high perceptual and cognitive quality. We found as most appropriate the styles of hand drawn illustrations [8], storyboards [3], and comic strips [5]. The concept allows us to adopt the creation of dynamic glyphs from guidelines used for classic media, which have been successfully approved over decades. With respect to arrows, for example, we generate a 3-dimensional visual element that appears to be mostly parallel to the view plane.

To render dynamic glyphs in a non-photorealistic fashion we particularly deploy artistic strokes [7]. Artistic strokes are well suited to place lines and curves into 3D scenery, for instance, to visualize a motion path (see Fig. 4). Using variations in style leads to individual strokes and thus emphasize hand drawn creation.

Speedlines depict dynamics of scene objects [4]; as dynamic glyphs they symbolize direction and velocity of animated scene objects for a given point in time. Additionally, speedlines relate the past since they picture where the scene object has been before. Furthermore, they also relate the future since the position where the scene object will be soon can be estimated by deriving the future direction of flight from speedlines. To render speedlines, we extract extreme or artistically chosen points of polygonal geometry and, then, process their positions in 3-dimensional space with

respect to a given time interval. Based on these, we form an appropriate artistic stroke that symbolizes motion (see Fig. 5).

We also apply non-photorealistic rendering to the main 3D scene to achieve a consistent graphics style. For example, 3D scene objects can be rendered with enhanced edges [6] and NPR illumination models [2]. Real-time NPR techniques allow us to integrate dynamic glyphs in interactive graphics applications. The snapshots shown in this paper are actually derived from our interactive prototypic implementation.

5 Examples

Flight of a Paper Plane. We consider the following scenario: A paper plane flies beside a wall. It collides with the wall twice. Each collision alters the direction of flight. Finally, the paper plane descends smoothly nearly the camera.

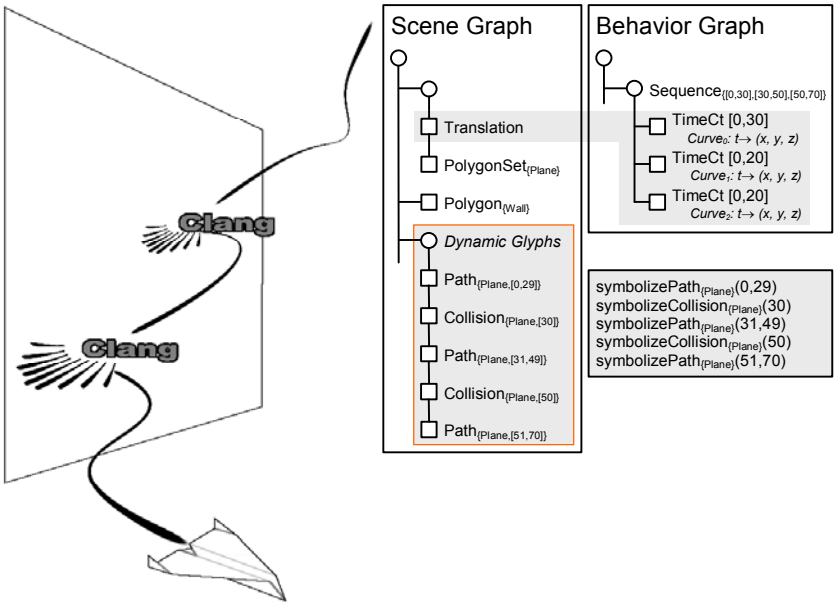

Figure 4: Dynamic glyphs depicting the flight and collisions of a paper plane (left). The diagram shows scene and behavior graph and depiction rules needed to construct dynamics glyphs (right).

The initial scene graph for the animation consists of a translation node that specifies the position of the paper plane, a set of polygons representing the geometry of the paper plane, and a single polygon representing the wall. Since the trajectory of the flight gets interrupted twice, the whole animation of the flight specified in the behavior graph consists of three curves that are processed in sequential order. The three curve-map based constraint nodes, shown in Fig. 4, map a point in time to a 3-

dimensional vector derived from the given curve. Then, the constraint nodes manipulate the translation in the scene graph and, thus, position the paper plane in 3D scenery. The time-sequence behavior node activates and deactivates, respectively, the constraint nodes in sequential order due to their time requirements.

To visually communicate the animation of the flight, we symbolize each curve in a chosen time interval and both collisions at that points in time, when the events take place, by the use of dynamic glyphs.

Therefore, we specify what to depict for a certain scene object in either a time interval or at a point in time in depiction rules. The collection of depiction rules specifies the symbolization of dynamics; they can be given in any order since they do not influence each other.

If both, the behavior graph and a collection of depiction rules, are provided we analyze the behavior graph to extract information related to scene objects used for interpreting the rules to, finally, generate dynamic glyphs. To symbolize the path of the paper plane's flight, we choose long artistic strokes placed along each curve according to the specified time interval. As dynamic glyphs depicting the collision at a discrete point in time, we place short artistic strokes starting around the point of collision and following the direction of flight to visualize the impulse. Furthermore, we add a notation at or nearby the point of collision to symbolize the noise that arises. These dynamic glyphs will then be inserted into the scene graph. Fig. 4 illustrates the whole flight of the paper plane.

Bouncing Ball and Flying Paper Plane. We consider the following scenario: A ball

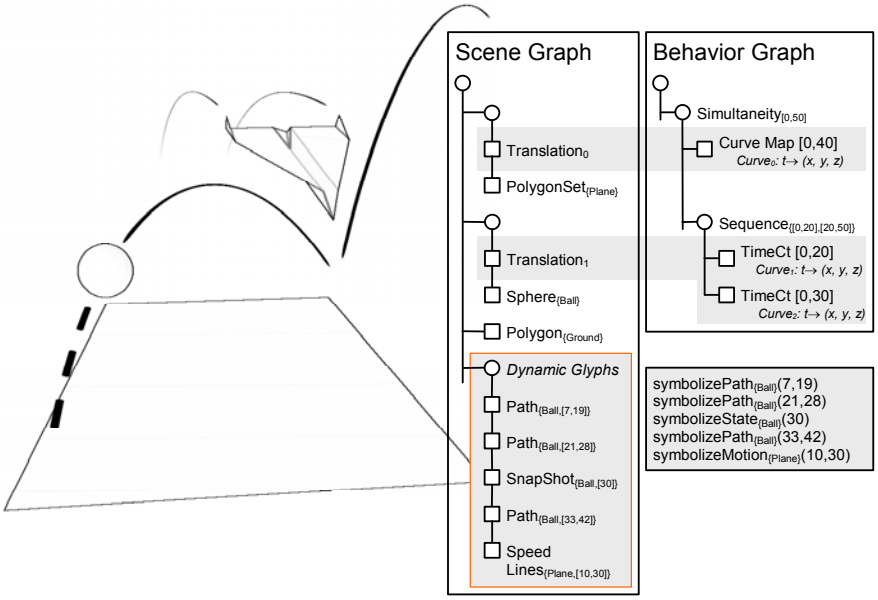

Figure 5: Dynamic glyphs depicting the simultaneous animation of a bouncing ball and a flying paper plane (left). The diagram shows scene and behavior graph and depiction rules needed to construct dynamics glyphs (right).

bounces on the ground. It has hit the ground once, has lifted up, and falls down following a parabolic curve. Simultaneously, a paper plane is flying around.

As before, the scene graph contains a translation node that controls the position of the paper plane. Additionally, it contains a translation node and a spherical shape to position and visualize the ball. Since the bouncing ball and the flying plane are animated simultaneously the behavior graph contains a simultaneity behavior node that starts the animations of both scene objects at the same time. One curve-map based constraint node describes the trajectory of the plane. Again, the path of the ball is divided into two curves. The corresponding constraint nodes are assembled in a sequence behavior node to be processed one after another.

For depicting this scenario, speedlines gradually fade off in the past like condensation trails disappear behind a plane. In analogy, we can conceive speedlines like turbulences behind a moving object invoked by its velocity and its striking surface. Fig. 5 shows the resulting image of the 3D scene with integrated dynamic glyphs.

5 Conclusions

The presented concept formally specifies dynamics and derives dynamic glyphs that visually encode events and activities in images of 3D scenes; it extends the well-known scene graph by a complementary behavior graph. Dynamic glyphs offer a rich vocabulary to designers for expressing different kinds of dynamics; depiction rules can be designed on different levels of abstraction.

We see a large potential of application for the automated production of technical and instructive illustrations as well as for visual summaries of linear media contents. Our future research concentrates on identifying and categorizing dynamic glyphs, depiction rules, and appropriate rendering techniques. We also would like to apply the presented technique to concrete application domains such as authoring tools.

References

1. Döllner, J., Hinrichs, K.: Object-oriented 3D Modeling, Animation and Interaction. In: The Journal of Visualization and Computer Animation (JVCA), 8(1) (1997) 33-64
2. Gooch, B., Gooch, A.: Non-Photorealistic Rendering. A.K. Peters (2001)
3. Katz, S.D.: Film Directing Shot by Shot: Visualizing from Concept to Screen. Focal Press (1991)
4. Masuch, M., Schlechtweg, S., Schulz, R.: Speedlines – Depicting Motion in Motionless Pictures. In: ACM SIGGRAPH'99 Sketch and Applications (1999) 277
5. McCloud, S.: Understanding Comics – The Invisible Art. HarperPerennial, New York (1994)
6. Nienhaus, M., Döllner, J.: Edge-Enhancement – An Algorithm For Real-Time Non-Photorealistic Rendering. In: Journal of WSCG'03, 11(2) (2003) 346-353
7. Northrup, J.D., Markosian, L.: Artistic Silhouettes: A Hybrid Approach. In: Proceedings of NPAR 2000 (2000) 31-38
8. Strothotte, T., Schlechtweg, S.: Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation. Morgan Kaufman, San Francisco, CA (2002)