Depicting Dynamics Using Principles of Visual Art and Narrations

Marc Nienhaus and Jürgen Döllner
University of Potsdam, Germany

In visual art, a single static image frequently represents much more than projected 3D scenery. Artists include subtle visual elements outlining movements, indicating past or future events, sketching ongoing activities, or guiding the observer's attention. Artists have found ways to visualize physical as well as nonphysical dynamics of scenes using graphics techniques. In a sense, we can consider these *smart depictions*—a form of expressive, visual content adopting the styles of visual art and abstraction techniques. These depictions can serve, for example, as pictograms and signs that advise and assist people. They're also present in comic books and storyboards that effectively present dynamics and narrate sequential processes (see Figure 1).

Our smart depiction system automatically generates compelling images of dynamics following traditional visual art and graphic design principles (such as in comic books and storyboards).

Taking smart depictions a step further, we created a system that automatically generates smart, compelling images of 3D scenes that depict dynamics following traditional design principles of visual art, visual narrations, and graphic design, such as those found in comic books[1] and storyboards.[2] (For an introduction to the terminology, see the sidebar "Key Terms and Principles.") These media offer a rich vocabulary of visual art deployed as techniques to facilitate visual communication of a wealth of activities and events in static images. In particular, we can symbolize in a single, static image past, ongoing, and future activities as well as events taking place in 3D scenes. Additionally, we can take into account nonvisual information. For example, in the scope of narratives, we can integrate information such as tension, danger, and feelings into the symbolization process.

Designers and artists traditionally depict dynamics by hand or use imaging or sketch tools. The challenge for us was to find a solution for automating the process of specifying, interpreting, and mapping dynamics of and visual narrations in 3D scenes. Our general method takes the following approach:

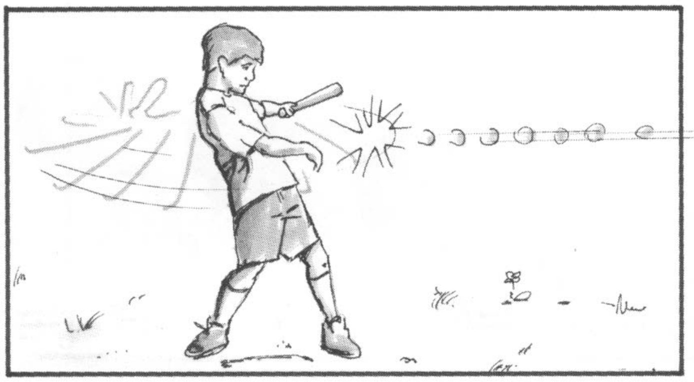
- Depiction techniques analyze the scene and behavior descriptions (for example, encoded in scene graphs) and map found and relevant dynamics to dynamics glyphs.
- Dynamics glyphs represent additional graphics elements that augment the resulting image of the 3D scene.

Designers can configure the way depiction techniques operate, and they can edit the visual and textual appearance of dynamics glyphs. Consequently, the system lets us model "from word to image."[3]

As an enabling technology, the system intensely uses nonphotorealistic rendering.[4] In fact, digitizing the process of generating visual art is increasingly more feasible because of expressive and artistic rendering algorithms, most of them now operating in real time.

Compared to imaging and sketch tools, the dynamics-depiction system offers several benefits:

- It explicitly models graphical representations of dynamics.



1 Storyboard depiction illustrating the batting sequence of a boy playing baseball. The depicted dynamics include the swing of the bat, the bat hitting the ball, and the accelerated ball.

Image courtesy of Michael Wiese Productions; http://www.mwp.com

Key Terms and Principles

The following are key terms and principles that we use throughout this article.

Visual and sequential art

Symbolization simplifies the perception of activities and events by an iconic language that abstracts from reality. The vocabulary of symbols includes arrows, strokes, bubbles, and signs.

Motion lines indicate moving objects by well-placed strokes. Streaking the background additionally indicates a moving camera.

Ghost images mark past, present, and future positions of objects by drawing multiple images of the original objects.

Visual metaphors indicate nonvisual phenomena like sound, speech, smell, tension, and feelings using symbols that are associated to a scene or story context.

Panels frame single depictions that form entities of a narration. Each panel can depict a single activity or event that contributes to the comprehension of a story.

Closure represents the ability to reconstruct and conceive sequential processes and narrations based on depictions that omit transitional steps and show only discrete moments in different perspectives. (For example, arranging a sequence as a collection of keyframe panels.)

Storyboarding shooting directions

Shots frame part of a staging; typically they indicate camera placement and narration instructions for later production.

Shot flow represents the visual connection of a sequence of shots, whereby each shot can vary in size, aiming at a consistent spatial–temporal order. Shot sizes include medium and close-up shots.

Medium-shots frame only half the part of a scene object—for example, to capture an actor's gestures and body language.

Close-ups frame a small part of a scene object in detail to position the viewer closer to it—for example, to take a position for a dialogue sequence.

Moving cameras

Crane shots define a mostly vertical uninterrupted movement of a camera. In the beginning, they establish the environment towering above the scene (*establishing shot*), and then enter into details to direct the attention from the general to the specific.

Tracking shots define a flowing movement of the camera, tracking an object in a single shot or in a sequence of shots to visualize the varying composition of multiple story elements.

Related Work

In many domain-specific areas, researchers have investigated smart depictions that aim to reduce design, time, and cost efforts. In the domain of assembly instructions, Agrawala et al.[1] present a system that plans assembly operations and produces compelling step-by-step illustrations for assembling everyday objects. The algorithmic techniques are based on design principles derived from cognitive psychology research. In contrast, we focus on producing images for communicating dynamics and narrating visually based on concepts found in comic books and storyboards.

In the scope of visualizing dynamics, Cutting[2] surveys traditional techniques for depicting motion in static images from a perceptual point of view. He states that representations of motion—compared to reproductions of static scenes—haven't been met adequately and encourages focusing on techniques that have the ability to convey motion in static images. He further introduces criteria such as clarity of object, direction of motion, and precision of motion to judge the efficacy of representations of motion in art, science, and culture.

Comic books represent visual media that can communicate complex narratives in a way that even children can understand easily without reading words. McCloud[3] explains a wide variety of visualization and abstraction techniques used to generate sequential art in comic books. These include techniques to depict the motion of single objects and to illustrate noises and speeches bound to time.

Storyboard artists deploy similar techniques to visualize

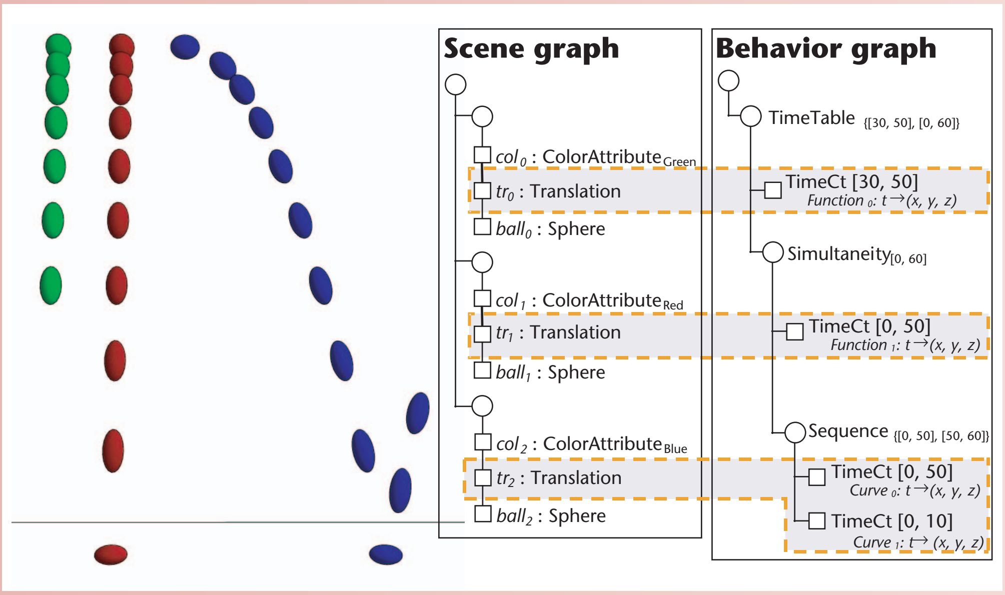
continued on p. 42

- We can apply it to any standard 3D scene description, and it integrates smoothly into any scene graph-based graphics system.
- It supports design alternatives by selecting specific types of dynamics and by configuring the symbolization process. This lets the user experiment and choose the depictions that best communicate his or her ideas.
- A user can easily modify a given depiction and accept and adopt changes after a reconsideration phase to start the next iteration step.

We can apply smart depictions in several scenarios:

- Nonartists can model and generate smart depictions of dynamics and visual narration out of common scene descriptions based on the system's built-in analysis and symbolization capabilities.
- Graphics artists can customize and automate the production of smart depictions.
- In planning and discussion processes, smart depictions provide content-rich static imagery that's well-suited as a basis for manual and cooperative sketching and illustrating.
- In the preproduction phase of motion picture productions, smart depictions serve as storyboard-like depictions derived from a previsualization of the scene, including its narration and intended dynamics.

To see other ways researchers have attempted to apply smart depictions, see the "Related Work" sidebar.


Scene graph
col_0 : ColorAttribute$_{Green}$
tr_0 : Translation
$ball_0$: Sphere
col_1 : ColorAttribute$_{Red}$
tr_1 : Translation
$ball_1$: Sphere
col_2 : ColorAttribute$_{Blue}$
tr_2 : Translation
$ball_2$: Sphere
Behavior graph
TimeTable {[30, 50], [0, 60]}
TimeCt [30, 50]
Function $_0$: $t \rightarrow (x, y, z)$
Simultaneity$_{[0, 60]}$
TimeCt [0, 50]
Function $_1$: $t \rightarrow (x, y, z)$
Sequence {[0, 50], [50, 60]}
TimeCt [0, 50]
Curve $_0$: $t \rightarrow (x, y, z)$
TimeCt [0, 10]
Curve $_1$: $t \rightarrow (x, y, z)$

A Squash and stretch applied as a traditional animation principle to bouncing balls in a 3D scene. Their animations and different time layout strategies have been modeled using scene and behavior graphs.

continued from p. 41

and illustrate the storyline of a movie as storyboards; Katz[4] and Begleiter[5] present the underlying design principles. Usually, storyboards are produced and reconsidered in collaboration with the director and the screenwriter of the motion-picture production; they provide a basis for discussions about the screenplay. Storyboards deliver a skeletal structure, which documents the set design and depicts the shooting directions of the story used in preparation for the production. As effective diagrams for documenting, communicating, and discussing ideas, they let outside participants understand the layout of the story and set design.

Traditional 2D hand-drawn animations comprise well-established techniques for conveying animations. Lasseter[6] identifies and terms basic principles of traditional animations, such as the squash-and-stretch technique. He further postulates their importance for 3D computer animations. Chenney et al.[7] apply the squash-and-stretch technique to generate comprehensible cartoon-style animations by squashing and stretching objects through motion.

In a time-lapsed animation, the deformations of objects in each frame vividly depict the objects' dynamics: their velocity and acceleration. We deploy squash and stretch as a technique to generate multiple images of an object in motion using forward lean (see Figure A) to maintain the clarity of the moving object and to indicate its direction of motion in a static image.[2]

To visualize motion in still images, Hsu and Lee[8] implement skeletal strokes for generating *speed lines*. Speed lines (also known as motion lines) streak away from the object in the opposite moving direction. Thereby, the lines convey the object's locomotion and velocity similar to motion blur but in a static image using expressive rendering.

Masuch et al.[9] present one of the first approaches in computer graphics that exclusively focuses on presenting the motion of objects in still images using nonphotorealistic rendering. They complement speed lines with repeatedly drawn contours of moving objects to depict their motion in a single image. Further stylization—such as different line styles—provide a hand-drawn impression.

Nienhaus and Döllner[10] introduce a concept for deriving and interpreting dynamics provided by scene and animation specifications. The framework maps dynamics to dynamics glyphs that represent, for instance, speed lines for depicting motion.

We extend our initial approach with a formal description of the process for assembling dynamics in scene and behavior graphs. We also note other improvements in this article.

References

1. M. Agrawala et al., "Designing Effective Step-by-Step Assembly Instructions," *Proc. ACM Siggraph*, ACM Press, 2003, pp. 828-837.
2. J.E. Cutting, "Representing Motion in a Static Image: Constraints and Parallels in Art, Science, and Popular Culture," *Perception*, vol. 31, no. 10, 2002, pp. 1165-1193.
3. S. McCloud, *Understanding Comics—The Invisible Art*, Harper Perennial, 1994.
4. S.D. Katz, *Film Directing Shot by Shot: Visualizing from Concept to Screen*, Michael Wiese Productions, 1991.
5. M. Begleiter, *From Word to Image—Storyboarding and the Film-making Process*, Michael Wiese Productions, 2001.
6. J. Lasseter, "Principles of Traditional Animation Applied to 3D Computer Animation," *Proc. ACM Siggraph*, vol. 21, no. 4, ACM Press, 1987, pp. 35-43.
7. S. Chenney et al., "Simulating Cartoon Style Animation," *Proc. 2nd Int'l Symp. Non-Photorealistic Animation and Rendering*, ACM Press, 2002, pp. 133-138.
8. S.C. Hsu and I.H.H. Lee, "Drawing and Animation Using Skeletal Strokes," *Proc. ACM Siggraph*, ACM Press, 1994, pp. 109-118.
9. M. Masuch, S. Schlechtweg, and R. Schulz, "Speedlines—Depicting Motion in Motionless Pictures," *Siggraph Conf. Abstracts and Applications*, ACM Press, 1999, p. 277.
10. M. Nienhaus and J. Döllner, "Dynamic Glyphs—Depicting Dynamics in Images of 3D Scenes," *Proc. Smart Graphics*, Springer, 2003, pp. 102-111.

Scene Graphs

The *scene graph*[1] specifies 3D scenes in a hierarchical way using *scene nodes* as building blocks. In general, a scene graph library provides a collection of scene nodes, which model structural and graphical aspects of 3D scenes. The following are descriptions of the most important categories of scene nodes:

- *Groups*. Build up the hierarchical structure. Groups are also used as inner nodes of a scene graph. Examples are the *Branch Node*, which collect a number of subgraphs, and the *Switch Node*, which select one out of many subgraphs as an active child.
- *Shapes*. Specify geometric objects and are arranged typically as leaf nodes. Examples include Box, Sphere, Cone, and PolygonMesh.
- *Transformations*. Specify geometric transformations. The collection of transformation nodes along a path through the scene graph defines the transformation from the local to the world coordinate system. Examples include Translation, Scaling, Rotation, and DirectionOfFlight.
- *Appearance attributes*. Specify properties and techniques that define the visual appearance of scenes and scene objects. Examples include Color, Material, and Texture as well as attributes used in the scope of nonphotorealistic rendering, such as EdgeEnhancement, CartoonStyle, and SketchyDrawing.
- *Environmental attributes*. Specify properties of the scene's environment. Examples include LightSource, PhongLightingModel, GoochLightingModel, ShadowCaster, and ShadowReceiver.
- *Nongraphics attributes*. Provide application-specific and semantics information within the hierarchical scene description. Examples include Identifier as a textual description of a subgraph, and FilterTag.

As a general mode of operation, only the nodes along the path from the root node to a specific node have an impact on that node and its components.

Scene graph rendering

For image synthesis, the scene graph is traversed in preorder (a specific order in which hierarchical graphs can be traversed, such as top–down or left–right). During the traversal, a graphics context manages hierarchically defined attributes. According to the rendering technique, scene graph rendering can imply multiple traversals of the scene graph. For example, nonphotorealistic edge enhancement produces several intermediate frame buffer results. Scene graph rendering is a critical real-time process. The scene graph library requires an efficient functionality to create and modify scene graphs (such as adding new subgraphs) and scene nodes (time-dependent properties).

Scene graph inspection

The scene graph inspection represents a generic traversal function to report the structure and content of a scene graph. The inspection allows for retrieving the hierarchy layout (such as "Collect all nodes that represent a specific character having a specific Identifier"), browsing through the scene graph (reporting each node, its components, and types), and detecting dependencies between attributes and shapes (such as "Which attributes apply to a given shape?" or "Which shapes are affected by a given attribute?"). Consequently, inspection represents the tool to interpret scene graphs. The user can invoke the inspection function at any time, independently from scene graph rendering.

Reference

1. J. Döllner and K. Hinrichs, "A Generic Rendering System," *IEEE Trans. Visualization and Computer Graphics*, vol. 8, no. 2, 2002, pp. 99-118.

Specifying scenes and their dynamics

We'd like to show how we specify scenes and their dynamics in our smart depiction system. These specifications represent the basis for all further functionality.

Specification requirements

In computer graphics, hierarchical scene descriptions have a long tradition, and various scene graph libraries and scene description languages support them. In a typical scene specification, we arrange 3D shapes, appearance attributes, geometric transformations, and environmental objects into a hierarchical structure. With these components, or scene nodes, developers construct scenes composed of individual scene objects.

With respect to the specification of scenes, the following functional requirements are essential for the scene graph library:

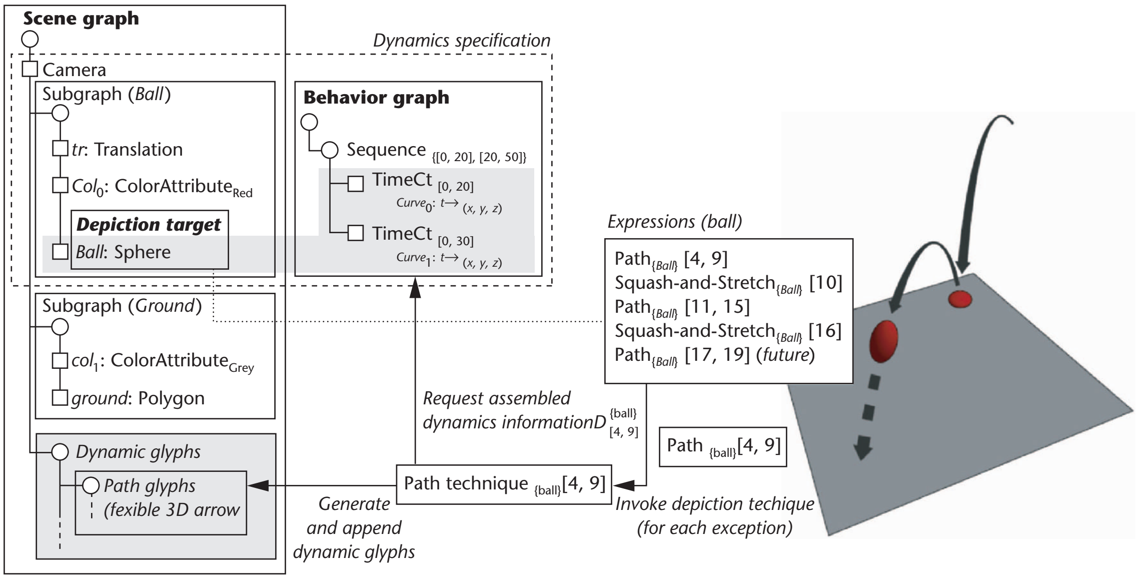
- The scene specification must support a generic traversal operation—for example, to access each individual scene object, its components, and its scene graph context.
- The scene specification must allow for assigning (semantics-based) identifiers to scene objects and for defining complex scene objects.

With respect to the specification of dynamics, we assume that it must allow for the following requirements:

- identifying which time-dependent changes are applied to a specific scene object,
- determining the lifetime of each animation, and
- evaluating the scene specifications for any given point in time.

In our approach, we apply the scene graphs (see the "Scene Graphs" sidebar) and behavior graphs as described by Döllner and Hinrichs.[5] Both types of graphs are represented as directed acyclic graphs (DAG).

Figure 2 (next page) illustrates a scene graph that specifies a simple scene consisting of a sphere representing a ball and a polygon representing the ground.



2 **System overview (left) illustrates the general workflow for generating dynamics depictions. The scene and behavior graph specify the ball's dynamics, which is the depiction target. Associating expression to the ball allows for modeling the representation of motion. A combination of squash and stretch and arrows as dynamics glyphs relate the dynamics of the bouncing ball in the past, present, and future (right).**

The graph branches into two subgraphs for specifying both the position (*tr*) and the color (col_0) of the ball and the color (col_1) of the ground.

Specifying dynamics using behavior graphs

The behavior graph specifies time-dependent and event-dependent aspects of scenes and scene objects. For a given scene graph, one or more behavior graphs may exist. Nodes of behavior graphs generally manipulate one or more nodes contained in the associated scene graph. The nodes used to construct behavior graphs are different from those of scene graphs.

The fundamental task of behavior graphs includes the definition of lifetimes of activities and points in time of events. Activities and events specify time-dependent changes of scene node properties. Activities take place during a defined, nonzero time interval, whereas events have no measurable duration because they take place instantaneously. Each node of the behavior graph provides its own time requirement, which represents the time demand to process the activity and event.

Layout of time flows. *Time-group nodes*, a major category of behavior-graph nodes, hierarchically organize the time flow at a high level of abstraction similar to specifications in storybooks. A time-group node calculates the lifetimes of its child nodes based on their time requirements and its own time-layout strategy. When a time-group node receives a time event, it checks which child nodes to activate or deactivate and then delegates the time event to its active child nodes. Specialized time-group nodes include the following:

- *Sequence*. Defines the total time requirement as the sum of the time requirements of its child nodes. It delegates the time flow to its child nodes in sequential order. Only one child node is alive at any given time during the sequence's lifetime.
- *Simultaneity*. Defines the total time requirement as the maximum of the child nodes' time requirements. It delegates the time flow simultaneously to its child nodes. The simultaneity layout shrinks or stretches the time requirements of the child nodes or applies alignment strategies to the lifetime of the child nodes to fit the duration.[5]
- *Time table*. Defines for each child node an explicit time requirement. It manages activation and deactivation of child nodes according to the child nodes' lifetime. For example, a time table can specify different starting times for individual objects in an animation.

Figure A illustrates a time-lapsed animation of three bouncing balls. Time layouts specify the lifetimes of each dynamic—for instance, a time table specifies different starting times for the green, red, and blue balls.

Activities and events. Having organized the overall time flow, constraint nodes let us specify activities and events. Essentially they associate a time-to-value mapping with the property of a scene node. For example, constraint nodes can set up the position of an object by associating a time-to-vector mapping with the object's midpoint. Time-to-value mappings of the form

$$f\text{: Time} \rightarrow \text{Type}$$

can implement a variety of mappings, such as mapping time to a constant value (*constant map*), to a value that results from linear interpolation of specified values (*linear map*), and to a value that results from calculating a point of a parameterized curve by interpreting time as a curve parameter (*curve map*).

A time constraint defined as

$$tct: (f(\text{Time}), \text{SceneNodes}) \rightarrow \text{SceneNodes}$$

controls time-varying parameters of a scene node contained in the scene graph. Whenever a constraint node receives a time event during its lifetime, it calculates new parameter values and assigns them to its constrained scene node. The generic class *TimeCt* takes care of most constraint variants.

In Figure A, time-constraint nodes (see the behavior graph) constrain translation nodes (see the scene graph) to specify the balls' movement. A function map, which maps time to a value that results from a function call, controls the fall of the green and red balls taking into account gravity. In Figure 2, two adjoining curves control the midpoint and, thus, the movement of the bouncing ball. They process in sequential order to form a single continuous trajectory.

Modifying local time flows. *Time-modifier nodes* define time-to-time mappings, which we use to alter the local time flow in behavior graphs. For instance, a reverse modifier inverts the direction of the time progress for its child nodes. Consider the bouncing ball in Figure 2. Here, we could use a reversal node to invert the ball's direction. Similar modifiers exist, such as repeating a time interval multiple times (*repeat modifier*) or defining a creeping time progress (*creep modifier*)—that is, slowing down progress in the beginning and speeding it up at the end.

Behavior graph inspection. In analogy to scene graphs, an inspection operation exists for behavior graphs, which we use to examine the time flow and mappings. For a given time interval, we can reproduce activation and deactivation of behavior nodes, reproduce the results of a mapping, and identify linkages of constraint nodes to scene nodes of the scene graph. Thus, we can analyze the state of the 3D scene for a given point in time.

Assembling dynamics information

In the next step for generating smart depictions, we have to assemble dynamics information—that is, we must detect information about which nodes of the scene graph are affected by nodes in the behavior graph at any point in time.

We first apply an inspection operation to the scene graph to trace the path from its root node to a given scene node *node*. As result, we get the path set *P*(*node*) containing a sorted list of scene nodes with respect to their scene graph depth:

$$P(node) = \{obj_k : obj_k \in \text{path}, k = \text{depth}\}$$

In particular, *P*(*node*) records all attributes and transformations that could potentially impact a node *node*.

Then, we invoke an inspection of the behavior graph to analyze its time layouts. As a result, we determine the global lifetime of time-constraint nodes. Let *tct* be a time-constraint node, then the analysis returns

$$tct_{[t_0, t_1]}(f, obj) : \text{active } \forall t \in [t_0, t_1] \wedge \text{inactive } \forall t \notin [t_0, t_1]$$

Now, we relate both results to each other to determine the set of time constraints that influence properties of scene nodes of *P*(*node*):

$$C(node) := \{tct(f, obj) : obj \in P(node)\}$$

We can further derive a subset of *C*(*node*) containing time constraints that are active at a certain point in time *t*:

$$C_t(node) := \left\{ \begin{array}{l} tct_{[t_0, t_1]}(f, obj) : t_0 \leq t \leq t_1, \\ obj \in P(node) \end{array} \right\}$$

Similarly, we can derive a subset of C(*node*) containing time constraints that are active (anywhere) in a given time interval [T_0, T_1]:

$$C_{[T_0, T_1]}(node) := \left\{ \begin{array}{l} tct_{[t_0, t_1]}(f, obj) : obj \in P(node) \\ \wedge [T_0, T_1] \cap [t_0, t_1] \neq \emptyset \end{array} \right\}$$

Taking also into account the set *P*(*node*) of scene nodes, we can evaluate the state or condition of a scene node at any point in time or time interval and identify the types (or translation) of scene nodes that contribute to a state change.

Because *P*(*node*) contains the transformation hierarchy, we can easily determine the trajectory of an object in 3D space by additionally sampling the set $C_{[T_0, T_1]}$(*node*) at discrete points in time during its time interval. We can then use position, velocity, and acceleration to depict an animation.

In conclusion, the tuple

$$D^{node}_{[T_0, T_1]} := \left(P(node), C_{[T_0, T_1]}(node) \right)$$

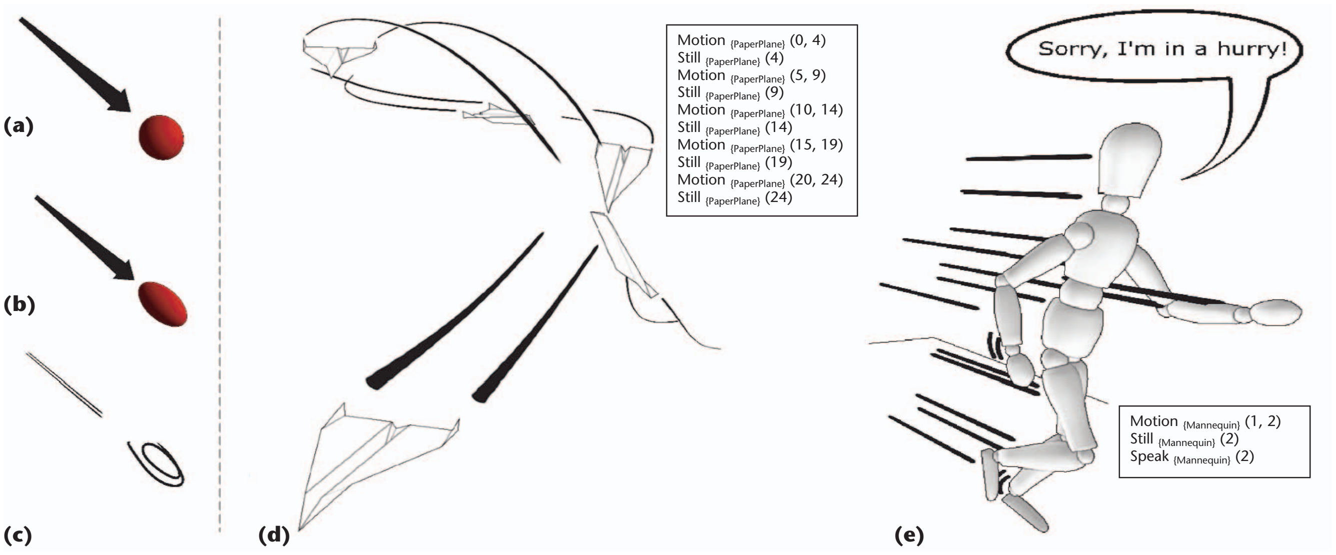
represents the assembled dynamics information of a scene node during a time interval.

Characterizing dynamics

Once we've collected dynamics information for a scene object, we have to select one of many possible characteristics of the dynamics, which we want to visualize in the final smart depiction. The later symbolization process uses this information, which the user typically provides.

A *characteristic of dynamics* represents a token that classifies the kind of activity performed or event triggered by a scene object—the *depiction target*—in an informal way. In the case of a sphere as depiction target moving between two positions, we can declare a *path characteristic* to refer to the depiction target's trajectory.

Basic characteristics of dynamics, which are elements



3 (a–c) Different symbols for a moving ball. (d) Flight of a paper plane using motion lines starting from the endings of the wings. (e) Running, talking character.

of an extensible set of tokens, include the following:

- *Path*. Indicates a schematic description of a movement of a depiction target.
- *Motion*. Indicates a more natural and informal description of a movement in contrast to the path characteristic.
- *Still*. Indicates past, present, and future positions and orientations of a depiction target.
- *Collision*. Indicates a collision with other scene objects as an event related to a depiction target.

Symbolization and depiction techniques

Having assembled dynamics information and chosen its characteristics, we can now symbolize the dynamics. This process is encapsulated with depiction techniques, which implement specific characteristics of dynamics by mapping assembled dynamics information to visual elements. These elements represent dynamics glyphs—that is, visual elements symbolizing activities and events in static images of 3D scenes. Technically, scene graphs specify dynamics glyphs, and these scene graphs link to the main scene graph as subgraphs for rendering.

For example, we implemented depiction techniques for symbolizing path and motion characteristics (see Figure 3). The *path technique* visualizes the trajectory of a movement; it constructs a flexible 3D arrow aligned to it that's oriented toward the viewer. For depicting motion, the *motion technique* generates motion lines (see the "Key Terms and Principles" sidebar), and includes additional strokes to provide a jittered appearance to make the motion easier to perceive (see Figure 3).

In Figure 3a, the path and the nondeformed ball visualize its motion in a motionless way. The ball seems to rest at that point of travel. Figure 3b shows deformation of the ball using a squash-and-stretch technique depicting believable motion. For Figure 3c, the sketchy depiction corresponds to an efficient drawing style and implies a fast-moving ball. In Figure 3e, we used motion lines for those parts of the character that move in the main direction of the motion and additional strokes for those that swing in opposite directions.

A depiction technique requests assembled dynamics information for a given time interval and for a given depiction target as a main information source. Formally, we can define a depiction technique as a mapping of the depiction targets' dynamics to a set of dynamics glyphs for the time interval $[T_0, T_1]$:

$$\text{DepictionTechnique}^{\text{Characteristic}} : D^{\text{depiction target}}_{[T_0, T_1]} \rightarrow \text{DynamicsGlyphs}$$

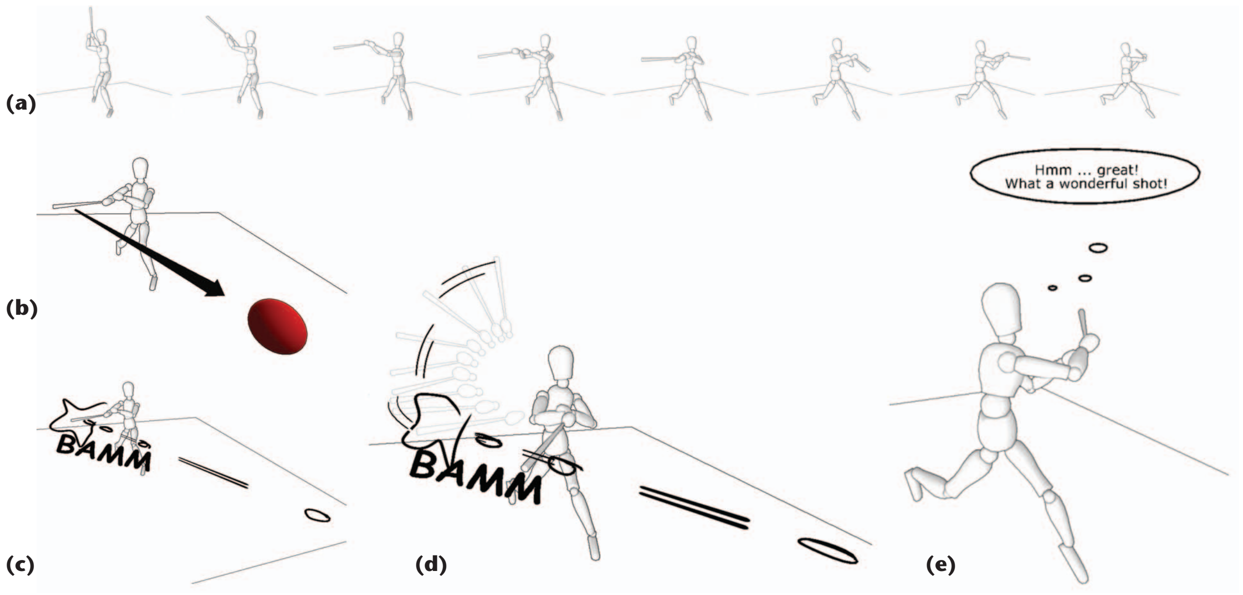
Figure 2 illustrates the path of a bouncing ball. The path technique generates arrows for the ball's trajectory. For this, it determines the position of the depiction target at different points in time to reconstruct that path. The *still technique* constructs ghost images (see the "Key Terms and Principles" sidebar) of the ball in a squash-and-stretch style at discrete points in time. The depiction technique maps position, velocity, and acceleration to symbolize the ball in that traditional form. (We use the acceleration of the ball to determine the transition between squash and stretch modes.)

Thus, with depiction techniques, the system maps triples consisting of characteristics of dynamics, a time interval, and assembled dynamics information to sets of dynamics glyphs.

Interactive composition of depictions

In practice, depicting dynamics represents a creative process and depends largely on the intentions and skills of the graphics designer. To give designers and artists as much control as possible, our system lets these depictions be interactively composed and customized. For this, we provide an expression-like language, which lets users invoke and set up depiction techniques.

Expressions let users create, store, apply, and configure



4 **Animation sequence showing a batter hitting a ball. (a) Illustration of the time-lapsed animation. (b) Batter when he hits the ball, and the path of the ball after being hit. (c) Depicting the hit by symbolizing the collision and the noise. The depiction shows the same action but in a sketchy style. (d) Narrating the batter's motion sequence, resulting in a more vibrant depiction. (e) Showing the batter realizing his excellent hit.**

dynamics depictions. In particular, users can directly set up parameters of depiction techniques to do the following:

- *Control the visual appearance of dynamics glyphs*. As an example, we consider the still characteristic that indicates positions and orientations of depiction targets as ghost images. The depiction technique can simply render the depiction target, render the depiction target in a squash-and-stretch style to additionally visualize its velocity by deformations (see Figures A and 2), or render a sketchy representation to mimic a hand-drawn illustration (see Figure 3).
- *Control the composition of dynamics glyphs*. For example, the collision technique symbolizes potential collisions between two given scene objects. The user can define the set of dynamics glyphs for visualizing collisions by rendering associated sounds as texts (see Figure 4).
- *Control the composition of time*. Defining what is past, present, and future is crucial for dynamics representations in images. For instance, the dashed arrow in Figure 2 effectively illustrates the path of the bouncing ball in the future. The user can provide temporal hints with expressions as an optional parameter.

In general, each expression requires the dynamics' characteristics that identify the depiction technique, the time interval to specify the period for depicting dynamics, and optional parameters to configure the depiction technique.

```
procedure evaluate(SetOfExpression exprs, DepictionTarget obj) begin
    ∀ expr ∈ exprs begin
        DepictionTechnique dt ← findTechnique(expr.Characteristic)

        SceneNode dynGlyphs ← dt.depict ($D^{\text{depiction target}}_{\text{expr.TimeInterval}}$ , expr.Parameter)

        SceneGraph.append(dynGlyphs)
    end
end
```

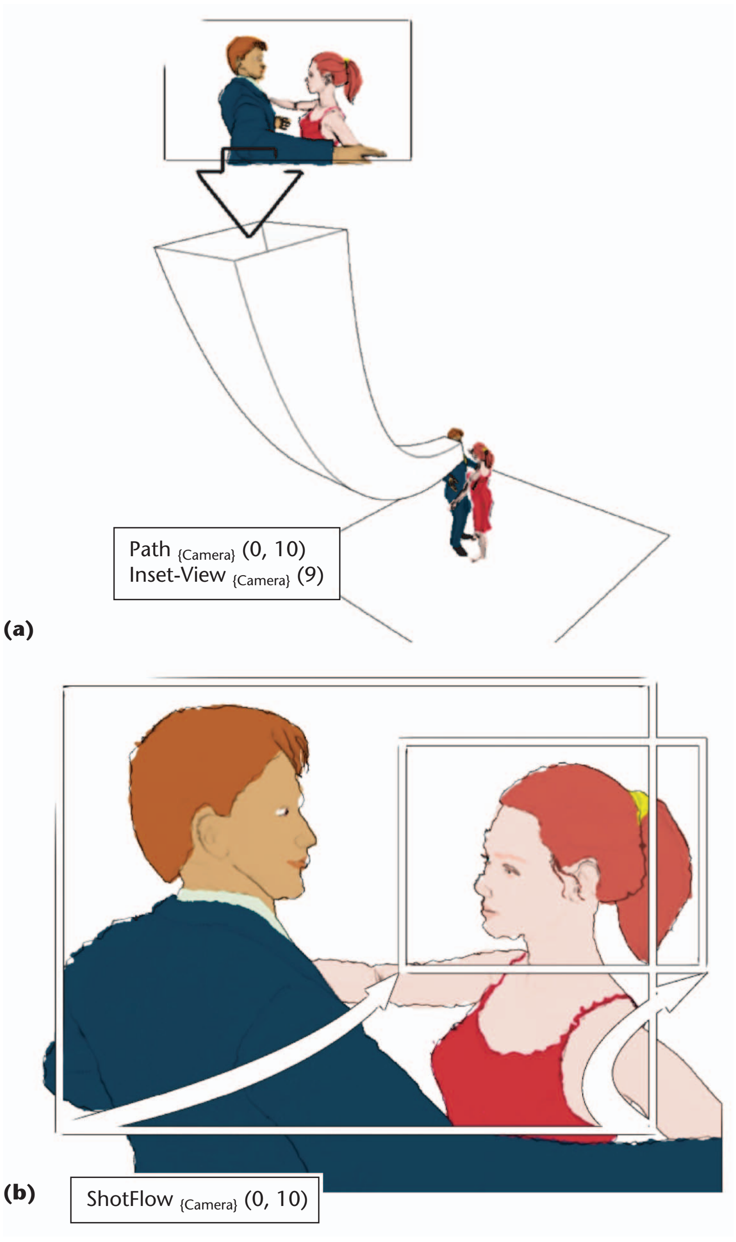
5 **Mapping dynamics to dynamics glyphs using expressions and depiction techniques.**

System overview and user involvement

Figure 2 summarizes the general workflow of our system so far. To model the intended depiction of an object's dynamics, the user selects a scene node in the scene graph as a depiction target and defines a set of expressions. Once associated with the depiction target, the system evaluates the set of expressions as follows:

1. For each expression, the system invokes the corresponding depiction technique, whereby the technique choice is based on the characteristics of dynamics given by the expression.
2. The depiction technique requests the dynamics specification to retrieve assembled dynamics information for the specified depiction target and time interval.
3. The depiction technique interprets retrieved data and constructs dynamics glyphs.
4. Dynamics glyphs, eventually, are linked to the main scene graph.

The pseudocode in Figure 5 illustrates the evaluation of



6 Camera-related depiction techniques can visualize (a) crane shots or (b) close ups in storyboard-like depictions. An additional *inset view characteristic* in (a) illustrates a medium shot of the scene; it was inspired by the long crane shot from the movie *Notorious*. For generating the sketchy depictions we deploy *sketchy drawing*, a real-time, nonphotorealistic rendering technique.

a depiction target and its associated set of expressions.

The system renders the 3D scene together with the dynamics glyphs. It doesn't render the depiction target itself because its picture is inessential and, in particular, would interfere with the depictions of its dynamics.

In our implementation, selecting a depiction target triggers the inspection of both the scene and behavior graphs. Then, we can invoke and process depiction techniques in real time, so that the user can interactively experiment with techniques using expressions and navigate the 3D scene. The sets of expressions (except for optional parameters) in the insets of Figures 2, 3, 6, and 7 define the corresponding depictions.

Using semantics for depictions

Until now, we have merely considered scene nodes as depiction targets. However, a scene graph representation is sometimes not sufficient to unambiguously define a depiction target. That is, semantics information about scene objects must be available. In particular, depictions of activities and events depend on semantics information because generally no obvious depiction techniques exist like in the case of depicting motion and path. For instance, we can't generate and position meaningful bubbles symbolizing speeches and thoughts until defining the character and its head explicitly (see Figures 3 and 4).

Assigning semantics to scene objects

Specifying scene objects with semantics information is subject to 3D scene modeling. We can assign semantics information to scene objects with a specialized attribute class called *identifier*. Identifier attributes permit hierarchical semantic descriptions for complex scene objects.

If a scene node contains an identifier, techniques looking for that kind of information will search in that node and its subgraphs. Otherwise, they will prune that node in the traversal. In this manner, our system can assemble a collection of scene nodes for one depiction target with specific semantics.

We define S as the set of scene nodes that contribute to a semantics description:

$$S = \{\text{node: node contributes to semantics}\}$$

The system assembles dynamics information for each scene node. So,

$$D^{S}_{[T_0,T_1]} := \left\{ D^{node}_{[T_0,T_1]} : node \in S \right\}$$

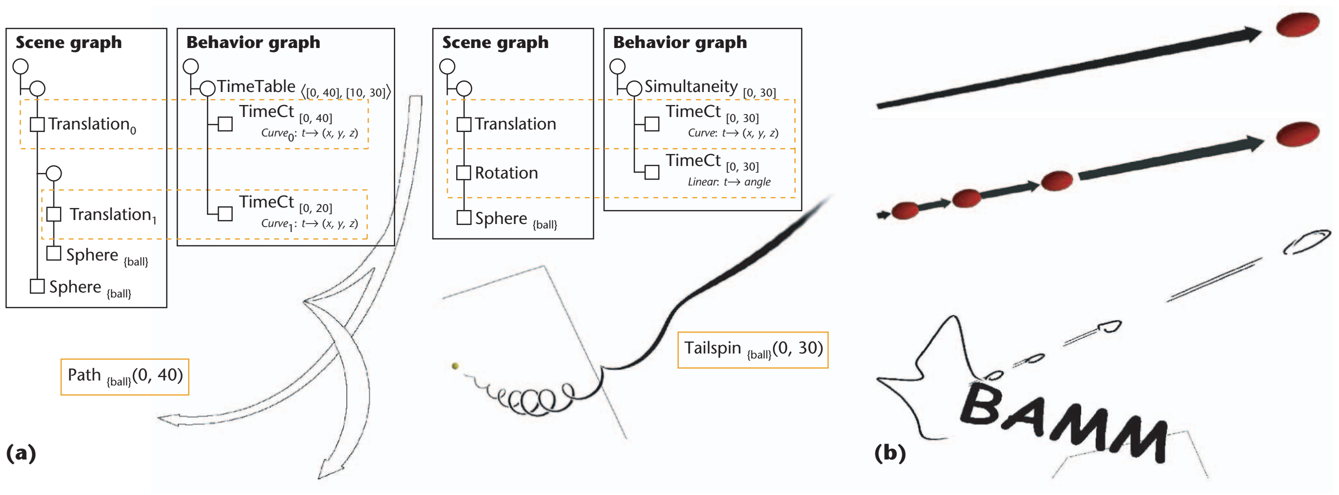
provides all assembled dynamics information that influence the depiction target with the specified semantics.

An independent subgraph represents each of the characters in Figures 3 and 4 . The subgraph contains further subgraphs, each of which represent parts of the body such as a head, arm, or hand. In this case, S consists of those shapes that form the visible corpus of the character. Once we assign animation data, such as motion capture data, $D^{\text{Character}}_{[t_0,t_1]}$ provides all dynamics information of the character. Consequently, depiction techniques can locate single parts of the body, identify their relations to each other, or consider the character as a whole at any point in time for generating dynamics glyphs. So, by defining the character through a hierarchical composition of identifiers, we can narrate the batting sequence in Figure 4.

Semantics-related depictions

For scene graphs enhanced by semantics information, we can refine depiction techniques and the characteristics of dynamics.

Semantics-related depiction techniques. For a specific characteristic of dynamics, we can implement



7 **(a) Pattern-based techniques analyze scene and behavior graphs leading to advanced depiction techniques such as the split path or the tailspin. Techniques also exist for identifying patterns in the set of expressions. (b) Applying the split operation to a path depicts the ball's movement and its acceleration. The sketchy depiction also includes a causing event. The ball is catapulted and, thus, accelerated by an external force.**

depiction techniques that convey dynamics more precisely for objects with specific semantics than a depiction technique implemented for a general scene object. For instance, a depiction technique that's specialized for camera semantics can symbolize the trajectory of a camera (path characteristic) as an extruded rectangular frame (see Figure 6). Moviemakers often use this sort of depiction in storyboards to visualize a long crane shot (see the "Key Terms and Principles" sidebar).[2] In addition to the camera's position, an extruded frame encodes its viewing direction and alignment.

Another example considers the depiction technique for the still characteristic of the paper plane semantics: It deforms the wings and endings of the paper plane under cross-acceleration similar to a real paper plane. This leads to a more dramatic appearance of the paper plane in a visual narration of its flight.

The following pseudocode illustrates the modified selection procedure for semantics-related depiction techniques:

```
DepictionTechnique dt ← findTechnique
    (obj.Semantics,
  expr.Characteristic)
```

Semantics-related characteristics of dynamics. Semantics information leads to a broader vocabulary of characteristics of dynamics—that is, for specific semantics we can add new characteristics and the appropriate depiction techniques for them.

For example, in cinematography a shot flow is clearly a characteristic of camera semantics. So, we can add that characteristic and implement its depiction technique regarding design principles for cameras inspired by storyboard depictions. Figure 6 illustrates a shooting direction from a medium shot to a close up. Here, both frames indicate which part of the scene is visible when taking the shot at certain points in time. The arrows indicate the movement of the camera for taking the close up. To cope with the manifold ways of camera movements and illustrations of shooting directions deployed by storyboard artists, we believe that much potential exists for exploring semantics-related depiction techniques.

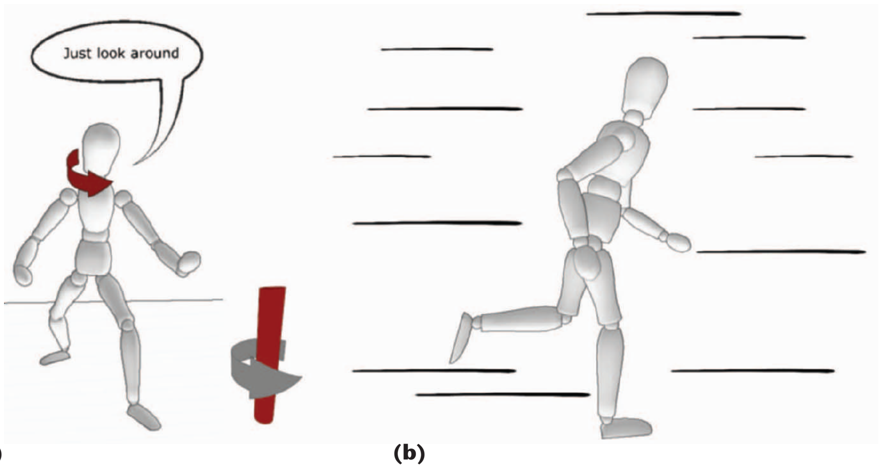
Information retrieval functions

Depiction techniques request $D^{S}_{[t_0, t_1]}$ to retrieve encoded time-dependent data (see Figure 2), such as the reconstruction of the transformation hierarchy. To facilitate the actual implementation of depiction techniques, we define *information retrieval functions* (such as the hierarchy retrieval function) that search for and analyze semantics-related data of a depiction target for a specific point in time.

A *center retrieval function*, for instance, determines the center of a depiction target, which we need to depict the object's trajectory. For a ball (or sphere) the center is likely to be the origin of its coordinate system in model space, whereas a character's center isn't well defined. The center might be located in the character's geometrical bounding box. This isn't appropriate because the box adjusts to its animated geometry. So, we opt for the hip as a character's center. In conclusion, the path technique applies to both the ball and the character for constructing path glyphs. Thus, retrieval functions allow for invoking depiction techniques for a broader set of semantics.

Core retrieval functions implement computational geometry algorithms. For instance, depiction techniques frequently require extreme points of 3D geometries for constructing motion lines. An *extreme points retrieval function* determines these points by evaluating the spatial arrangement of the vertices in strips, which are aligned to the object's moving direction.[4] In contrast, the extreme points retrieval function for the paper plane semantics provides the endings of its wings that typically produce turbulences (depicted by motion lines in Figure 3).

For a character, retrieval functions considering



8 (a) Bent arrow indicating a single rotation used to illustrate a character turning his head to look around. (b) Motion lines producing a streaked background indicate the motion of both the camera and the character.

human measurements[6] can determine information that isn't available at first glance. For instance, *facial measurement retrieval functions* can, based on the position and orientation of a character's head, provide the position of the eyes, nose, and mouth even though they aren't modeled explicitly. In this way, the *speaking technique* can align the cone of the bubble toward the character's mouth in Figures 3 and 8. Thus, retrieval functions in combination with semantics information can provide beneficial information.

Pattern-based symbolization

Besides modeling depictions interactively using expressions, further analysis of dynamics by identifying patterns assists the process of symbolization. Pattern-based techniques allow for determining relations

- in the composition of scene and behavior graphs,
- in the set of expressions, and
- among different dynamics that influence one another.

Thus, pattern-based techniques can give clues for producing depictions automatically and can enhance their comprehensibility.

Composition of scene and behavior graphs. At a higher level of abstraction, the assembly of both scene and behavior graphs and their relations can reveal patterns. In particular, animating those transformations that influence scene objects can lead to advanced characteristics of dynamics, including the following:

- *Turnaround*. If a scene object rotates about an axis in model space, a constraint node in the behavior graph animates the rotation angle of a rotation transformation located directly before the object in its scene graph path ($P(node)$). Whenever we detect this composition, we can create a turnaround characteristic for its motion yielding in a bent arrow aligned around the axis with a certain distance (see Figure 8).
- *Tailspin*. If a tailspin animates a scene object, then a simultaneity group having two child nodes (one for constraining its position and one for constraining its rotation angle) encodes these dynamics in a behavior graph. The system then maps a path characteristic to a tailspin characteristic to symbolize the dynamics with specific path glyphs (see Figure 7).
- *Split path*. If at least two scene objects that build up one depiction target follow the same trajectory in the beginning of an animation and then follow individual trajectories, their paths split smoothly into two. If we encounter a configuration of diverging paths in scene and behavior graphs, we can apply a split-path characteristic (see Figure 7).
- *Explode*. The pattern indicating an explosion characteristic is similar to that of the split path, but this time many scene objects of a single depiction target might abruptly change their direction of motion arbitrarily. Then, semantics-related explosion techniques either symbolize each launching part separately or produce a cloud of dust. They can also intensify the perception of the explosion by semantics-related sound using text.
- *Expand/collapse*. We interpret an animated scale transformation by enlarging or scaling down a single scene object with an expansion or collapse characteristic. If the scaling is located directly before the scene object, then the object pulses. Otherwise, if further transformations—such as translations—are located in between, the object additionally moves in 3D space. The expand/collapse technique handles expansions and collapses differently. The technique symbolizes the expansion through multiple arrows starting at the scene object's center heading in different directions while increasing their width. They end at the estimated boundary of the enlarged object. In the collapse mode, the technique inverts the direction of the arrows, so that they point to the object's center. In case of an assembly of diverging objects, we again apply the explode characteristic.

Set of expressions. A set of expressions that the user specifies can be subject to automatic enhancements. Our system provides a join operation and a split operation to assist dynamics interpretations.

- *Join operation*. We can sometimes depict expressions that temporarily overlap through specialized characteristics of dynamics. The join operation scans the set of expressions and merges applicable expressions into single expressions to invoke advanced depiction techniques. For instance, a collision might occur during an object's motion. Visualizing both separately can produce dynamics glyphs that overlap in the depiction producing disturbing effects. Combining both enhances glyph constructions because a specialized depiction technique smoothly incorporates them for rendering.

- *Split operation*. A single expression of a long time interval can be split into several expressions because a more fine-grained schema might depict the dynamics more appropriately. To facilitate a split operation, our system queries associated aspects of dynamics to derive indicative information such as velocity or acceleration. For instance, the motion characteristic of an accelerating object can be split into several expressions for motion to depict the dynamics in several time intervals and thus dramatize acceleration (see Figure 7).

Interacting dynamics. The dynamics of depiction targets can influence depictions of other objects' dynamics. For instance, well-placed motion lines distinguish fast from very fast movements. Generally this is the case with a fixed camera. But if the camera is moving with the object, we apply a traveling shot characteristic. Here, the object remains focused while motion lines are then used for the background to depict the motion of both the camera and the object[1] (see Figure 8). So, the relations of different dynamics have influence over depiction techniques in the whole. The pattern-based approach for symbolization helps us resolve cases in which dynamics influence one another.

Future work

We presented an automated depiction system for analyzing and symbolizing dynamics. Based on common scene and behavior specifications, the system produces smart depictions in a cost- and time-efficient way, and users can extend it with application-specific analysis and symbolization techniques. In our experience, nonphotorealistic 3D rendering techniques fit best to achieve results that come close to traditional and artistic works.

We noticed that new designs of dynamics glyphs could be systematically implemented on top of the presented framework. Future work might investigate which visual design of dynamics glyphs to use—for example, in the scope of virtual and augmentedreality applications. For the placement of dynamics glyphs, we could further automate the layout (such as for frames and bubbles).

The semantics-based analysis and symbolization must be analyzed in more detail. In particular, all kinds of camera-related depictions, which are relevant in the preproduction of a movie, need to be optimized further.

In addition, a pattern catalogue should be investigated. Although we identified pattern-based techniques and can cope with the mapping of patterns to glyphs, pattern-based techniques—in particular, interacting dynamics—are subject to future research. More techniques, patterns, and glyphs should be investigated for speech and sound as an interesting class of dynamics and an important category of multimedia content.

We believe that our techniques for depicting dynamics enhance image quality even for standard interactive and animated computer graphics applications, since they let us outline certain activities, visually indicate events, or enhance certain actors or objects. Depicting dynamics as a mostly automated process shows much potential for rendering more than just 3D scenery into single images. ■

Acknowledgments

We thank the Academy of Film and Television "Konrad Wolf," Babelsberg (HFF), and Ulrich Weinberg for inspiring discussions. We also thank the Virtual Rendering System team (http://www.vrs3d.org) at the Hasso-Plattner-Institute for their technical support.

References

1. S. McCloud, *Understanding Comics—The Invisible Art*, Harper Perennial, 1994.
2. S.D. Katz, *Film Directing Shot by Shot: Visualizing from Concept to Screen*, Michael Wiese Productions, 1991.
3. M. Begleiter, *From Word to Image—Storyboarding and the Filmmaking Process*, Michael Wiese Productions, 2001.
4. T. Strothotte and S. Schlechtweg, *Non-Photorealistic Computer Graphics—Modeling, Rendering, and Animation*, Morgan Kaufmann, 2002.
5. J. Döllner and K. Hinrichs, "Object-Oriented 3D Modeling, Animation and Interaction," *J. Visualization and Computer Animation*, vol. 8, no. 1, 1997, pp. 33-64.
6. A.R. Tilley and Henry Dreyfuss Associates, *The Measure of Man and Woman: Human Factors in Design*, John Wiley & Sons, 2001.

Marc Nienhaus *is a PhD candidate and research assistant at the Hasso-Plattner-Institute at the University of Potsdam, Germany. Nienhaus also studied mathematics and computer science at the University of Münster. His research interests include real-time rendering, nonphotorealistic rendering, and depiction strategies for symbolizing dynamics. He's a student member of the ACM and IEEE. Contact him at marc.nienhaus@hpi.uni-potsdam.de.*

Jürgen Döllner *is a professor in the Hasso-Plattner-Institute at the University of Potsdam, Germany, where he directs the computer graphics and visualization division. His research interests include real-time 3D rendering, nonphotorealistic rendering, spatial visualization, and software architectures of graphics systems. Döllner studied mathematics and computer science, and received a PhD in computer science from the University of Münster. Contact him at doellner@hpi.uni-potsdam.de.*