Non-Photorealistic Rendering Techniques for Motion in Computer Games

MICHAEL HALLER, CHRISTIAN HANL, and JEREMIAH DIEPHUIS, Upper Austria University of Applied Sciences, Media Technology and Design, Hagenberg

Still images can portray motion by using a stylized technique. Most of these techniques are commonly used for dynamic images as well (e.g., for cartoons). Typically, an artist abstracts the motion of a specific scene or animation to illustrate movement. Depicting motion in real-time environments is no less essential, and therefore a similar approach is desirable. Our approach is focused on three methods to stylize motion: squash-and-stretch, multiple images, and motion lines. A discussion of these methods for depicting motion in dynamic images and an implementation are presented in this paper. Finally, we discuss the results and conclude with the outlook for further development.

Categories and Subject Descriptors: I.3.2 **[Computer Graphics]:** Graphics Systems; I.3.6 **[Computer Graphics]:** Methodologies and Techniques; I.3.8 **[Computer Graphics]:** Applications
General Terms: Algorithms
Additional Key Words and Phrases: Non-photorealistic rendering, squash-and-stretch, motion lines, real-time, comics, motion, computer games.

1 INTRODUCTION

Modern computer graphics algorithms and current graphics hardware allow for a high degree of photorealism for game development. The programmable hardware in today's consumer graphics cards makes highly realistic techniques available in real-time. In the era of increasingly photorealistic games, buzzwords like bump-mapping, real-time shadows, and per-pixel shading/lighting are attracting a great deal of interest [Ishaya 2003]. Computer graphics has long been defined as the quest to achieve photorealism. As this goal gets closer, the realization grows that there is more to images than realism alone –particularly for game development. The same programmable hardware makes rendering photorealistically not only possible, but allows us to produce stylized renderings (e.g., cartoon shaders, etc.). As a result, a wide range of non-photorealistic (NPR) real-time algorithms are available now that previously were available only in the pre-rendered domain, where calculating an image took more than a second. Users expect realistic behavior in worlds that are rendered photorealistically. In games this requirement can result in a tremendous increase in the level of detail and in the accompanying workload. It is interesting that even in the era of beautifully realistic, rendered environments, people still occasionally unpack their old dusty computers and replay their old games like TETRIS, Space Invaders, Indiana Jones III, PAC-MAN, and Pong. Gamers do not care about the flat-shaded triangles and do not even miss the HDR textures or the bump-mapped polygons in video-quality – they simply want to play. Mark Bolas discovered in

Authors' address: Upper Austria University of Applied Sciences, Media Technology and Design, A-4232 Hagenberg, Austria; email: [haller|christian.hanl|diephuis] haller@fh-hagenberg.at

his research at Stanford University that while realistic environments help to engage the user and create a sense of being "there," greater abstraction engages the user's senses and imagination to create a greater sense of being "elsewhere," of being "in" the world being created. Another question is, of course, whether we should limit ourselves to the realistic environment approach only or take a more abstract, open approach? Ferwerda [2003] distinguished varieties of realism:

- *physical* realism, in which the virtual objects provide the same visual simulation as the real scene;
- *photorealism*, in which the image produces the same visual response as the scene; and
- *functional* realism, in which the image provides the same visual information as the scene.

Non-photorealistic pictures can be more effective at conveying information, be more expressive and more beautiful; virtual objects should be more convincing than realistic ones. Durand [2002] states that the border between photorealism and non-photorealism can be fuzzy and that the notion of realism itself is very complex. He also draws the interesting conclusion that the virtual world has to be interpreted convincingly rather than realistically. In other words, it is enough that the virtual, non-existing world *looks* real; i.e., it should be a *believable* world. The virtual objects should be expressive, clear, and look aesthetically perfect. McCloud [1993] states a similar conclusion via two examples: in the first example, he shows an image of a person as a stylized cartoon; in the second, he visualizes the same person in a naturalistically rendered style. McCloud's conclusion is that the stylized image convinces more people than the photo-realistic, naturalistically-rendered one. In his book, *Understanding Comics*, McCloud's examples, for the most part, address still images. But what's about motion? Is there a way to express motion? Are there any "rules," especially for games, that enhance the users' perceptions? We don't usually depict the motion of objects in computer-generated dynamic images – especially in real-time rendered dynamic images. Although motion is essential, it is not usually visualized (the visualization of motion was surveyed by Marcel Duchamp, and in 1912 by the Italian futurists, e.g., Giacomo Balla, Umberto Boccioni). In most technical descriptions, motion is portrayed in a number of abstract ways (e.g., by arrows, motion lines).

Even instruction manuals often use arrows and motion lines to visualize motion and time. Motion is also a very important element for real-time graphics, especially in computer games. It would be interesting to combine the techniques we know from comics in a real-time environment. Ubisoft's XIII game was one of the first commercial games with a graphic-novel presentation (a comic-book style) and with graphics elements from the comic's world [Ubisoft 2004].

As the title implies, this article first of all presents a novel approach for rendering the motion of objects in a real-time environment, not for still images. After a short discussion of related work, described in Section 2, we demonstrate our approach for rendering motion in Section 3. In particular, we show that motion can be presented in three different ways: via motion lines, multiple images, and the squash-and-stretch technique. Our technique makes use of the latest generation of hardware: a vertex shader and a fragment shader for the presentation of the concepts. The implementation is discussed in Section 4. Finally in Section 5, we show the results and present possibilities for further work.

2 RELATED WORK

In the early 1930s, animators at the Walt Disney Studio presented the 12 principles of animation (for a more detailed description and discussion of all principles, see Thomas [1981] – a short summary of principles was discussed in Lasseter [1987]). These principles were mainly used to guide production and creative discussions as well to train young animators better and faster. The guidelines for animators were the motivation for our approach, hence the integration of some "principles" in a real-time graphics environment. The original principles are still relevant today because they help us to create more believable characters and situations: a bouncing ball seems to be more realistic (or more believable) if its shape is deformed during the bounce. McCloud [1993] has also analyzed the abstract illustration of motion in comics. He describes the following possibilities as the most important for still images [McCloud 1993]:

- motion lines and
- multiple images

The first method, the motion lines technique, is the most common one for visualizing motion in comics. It is interesting to see the differences between the comics and the motion lines techniques used in the US, Europe, and Japan. However, this technique can express a high degree of movement, and emphasizes the quality of the illustration's dynamics. Motion lines are mostly used for still images (cf., Schulz [1999]), but we believe that its usage in a dynamic environment would enhance the users' perception as well. Therefore, in addition to the fun factor, the attention of the user becomes more focused on motion; clever use of this technique improves the user's perception a great deal. Using multiple images, and thus creating a "blurred" object on the screen, often seems to be more realistic than the normal movement of the object without the blur effect. When objects move very fast, our eyes expect to "see" something (still images, contours), even if the object is off the screen. Very complex motions can be emphasized with this technique because users have more time to analyze movement.

A combination of the two techniques, together with Disney's squash-and-stretch technique, will achieve the best results for visualizing motion in real-time. The deformation of an object due to its speed (during acceleration and slow-down) is a very important technique in enhancing visual perception. The squash-effect exaggerates the object and the stretch-effect anticipates the collision [Chenney 2002]. An important rule for the squash-and-stretch technique is that the volume of the object should remain constant, even if the object gets squashed or stretched. It goes without saying that in the real world we do not "see" objects being deformed. The stretching technique does not create a realistic effect; it makes a bouncing ball appear to move faster right before and after it hits the ground. A very good overview of how to implement the squash-and-stretch technique is presented by Chenney [2002].

Collomosse et al. [2003] postulate, that an abstract illustration of motion even makes sense for real movie sequences. In their work, they enhance the motion in movies by adding motion lines or deforming real objects so that they seem to be moving quickly. Their movies are impressive and convincing. Unfortunately their techniques require extensive pre-processing.

3 OUR APPROACH

We believe that the motion of objects can be classified into the following three groups: squash-and-stretch, multiple images, and motion lines. These methods can be used for static as well as dynamic images. The three methods can be combined to enhance the

graphical behavior of motion. However, we have to guarantee the following requirements and limitations:

- detailed geometric data for all objects we want to apply the technique to;
- the object's geometric data has to be modifiable;
- the position of the object has to be available at all times; and
- only past motions can be considered, future motions cannot .

3.1 Squash-and-Stretch

The most important principle in animation is the squash-and-stretch technique [LASSETER87][CHENNEY02]. When an object is moved, the movement emphasizes any rigidity in the object. The following parameters are used to adjust the style of motion (it includes direction, speed or acceleration of movement as well as the rigidity of the object):

- *Maximum speed*, $v_{\max}$ **:** the maximum speed an object can achieve.
- *Maximum acceleration*, $a_{\max}$ **:** the maximum acceleration an object can achieve.
- *Maximum speed scale*, $k_{v_{\max}}$ **:** the maximum amount an object is scaled based on the speed of the object.
- *Maximum acceleration scale,* $k_{a_{\max}}$ **:** the maximum amount an object is scaled based on the acceleration of the object.
- *Minimum acceleration scale,* $k_{a_{\min}}$ **:** the amount an object is scaled when slowed down with the negative maximum acceleration.

The units for maximum speed and maximum acceleration are insignificant because they are always seen in relation to the current speed or acceleration. All parameters are user-defined and assigned to a specific object. The desired results can be achieved easily by varying the parameters. The speed-dependent scaling parameter k_v can be calculated by setting the current speed in relation to the maximum speed. The speed-dependent rigidity of the object is represented by $k_{v_{\max}}$ and influences the amount of the scaling parameter k_v :

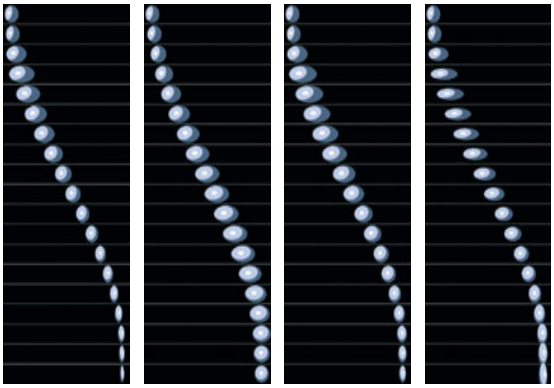
$$k_v = 1 + \frac{v}{v_{\max}} \cdot \left(k_{v_{\max}} - 1\right)$$

If the object is moving at constant speed, the scaling parameter also remains constant. If the object is slowing down, it is still stretched, since the scaling parameter k_v cannot fall below the value of 1. If the acceleration of the object is considered, the scaling factor is calculated corresponding to the following equations:

$$k_a = 1 + \frac{a}{a_{\max}} \cdot \left(k_{a_{\max}} - 1\right) \mid a \geq 0$$

$$k_a = 1 + \frac{a}{a_{\max}} \cdot \left(1 - k_{a_{\min}}\right) \mid a < 0$$

With a positive acceleration a the speed is increasing and the object is stretched. The closer a gets to $a_{\max}$ the more closely k_a gets to $k_{a_{\max}}$. If the acceleration a is negative, the object is slowed and the object is squashed. The closer the absolute value of a gets to



(a) (b) (c) (d)

Fig. 1. Time slices of an animation of a ball speeding up and slowing down horizontally. The squash-and-stretch is based on: (a) speed, (b) acceleration, (c) and (d) speed and acceleration. In (d) the volume of the ball is kept constant.

$a_{\min}$, the more closely k_a gets to $k_{a_{\min}}$. The resulting scaling parameter k_{result} is applied to the object in the direction of the motion:

$$k_{result} = k_v \cdot k_a$$

If the volume of the object is considered, the scaling parameter k_{norm} has to be normal to the direction of the motion:

$$k_{norm} = \sqrt{\frac{1}{k_{result}}}$$

By rotating the object in the direction of the motion, scaling can be controlled by a single parameter s [Chenney 2002]. Consequently, a single transformation matrix can be found:

$$\begin{bmatrix} s & 0 & 0 \\ 0 & \sqrt{1/s} & 0 \\ 0 & 0 & \sqrt{1/s} \end{bmatrix}$$

shows the different components of the squash-and-stretch parameter applied to a horizontally moving ball. In the pictures (a) to (c), the volume of the ball varies – in picture (d) the volume of the ball is kept constant, thus the behavior of the ball seems to be more dynamic. The figures (b) and (c) show the positive influence of the acceleration parameter.

Multiple Images. Snapshots of the moving object are taken for every constant time interval that can be configured. McCloud [1993] describes this phenomenon as multiple images. The goal is to more deeply involve the viewer in the action. The easiest form of multiple images is to draw the whole object several times, but there are a number of different styles for multiple images. How often a contour replication is generated has to be definable, thus we have one parameter that can be set by the user:

- *Replication rate*, n**:** The amount of contour replications generated per second.

Assuming that a scene is displayed with 60 fps and the replication rate n is 20, a contour replication is generated every third frame. One way to generate multiple images is to draw the object again for every contour replication on the basis of its position. This

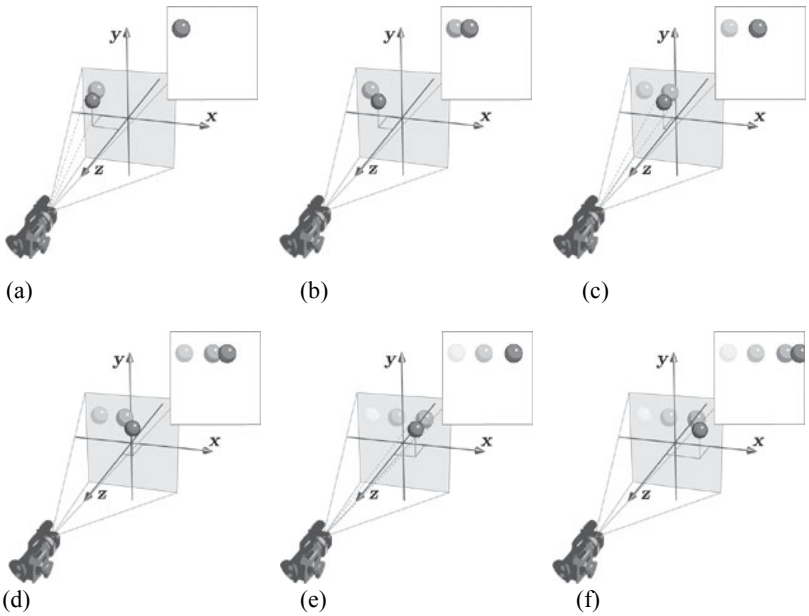


Fig. 2. Generating multiple images: every second frame (a) and (c) and (e), the moving object, is projected and rendered to the texture. To display the moving object (i.e. the dark-gray ball), including its contour replications, the texture is drawn in the background before the object is drawn.

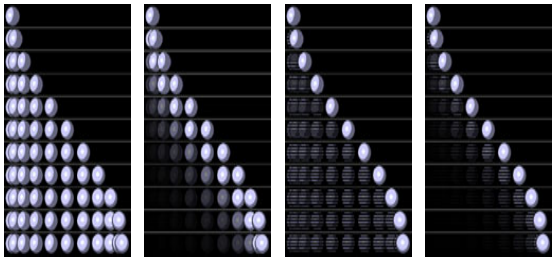


Fig. 3. Multiple images used for a horizontally moving object. In (a) the contour replications are drawn with full opacity; (b) shows continuously decreased opacity. In (c) and (d) the contour replications are stylized.

requires a great deal of computing power. In order to use multiple images in real-time, we used a more efficient method. A texture containing all contour replications is drawn in the background. If a new contour replication is generated, the object is rendered to the texture. A common style of multiple images is the continuous decrease in contour replications. Figure 2 shows the process of generating and displaying multiple images. Another very common method is to stylize multiple images by drawing the contour replications only partly or streaked. Figure 3 depicts the different styles of multiple images.

Motion Lines. Motion lines have been used for quite a long time to convey the sense of motion. Based on the medium, the culture, and of course on the personal style of the

artists, a number of different types of motion lines emerged. So several different parameters are necessary to influence the appearance of the motion lines when generated automatically. Schulz [1999] calls for a couple of requirements for the motion lines in still images:

1. The upper and the lower motion lines relative to the direction of motion have to be at the same level as the maximum extent of the object (A1).
2. Motion lines are to be placed on significant regions of the object (A2).
3. The number of motion lines has to be configurable (A3).
4. The space between motion lines must not be too constant (A4).
5. Accumulations of motion lines on certain regions of the object are not allowed (A5).
6. Motion lines are to begin after the object and are not allowed to cut the object (A6).
7. The length of motion lines has to be configurable (A7).

The object's geometrical data is used to calculate the starting points of motion lines. Every object consists of triangles defined by vertices. So the starting points for the limiting upper and lower motion lines have to be positioned on vertices (A1). To fulfill (A2), the starting points of the motion lines between the limiting ones have to be positioned on vertices as well. To determine the starting points of the motion lines, the positions of the vertices are transformed into screen coordinates (cf.,Figure 4). To get the starting points more easily, the object is rotated to the motion direction as shown in Figure 5. Now the limiting vertices can be determined by comparing their coordinates of the y-axis. To get the starting points for the motion lines in between, the object is split into equally large strokes (cf., Figure 6). In every stripe, the vertex with the lowest x-coordinate is used as a starting point.

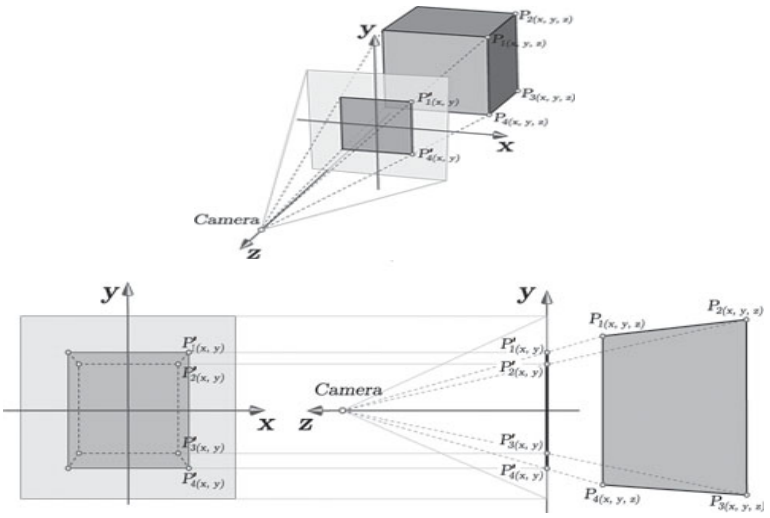
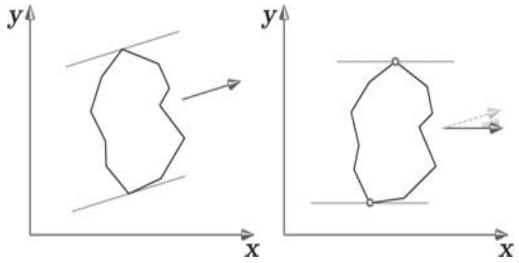
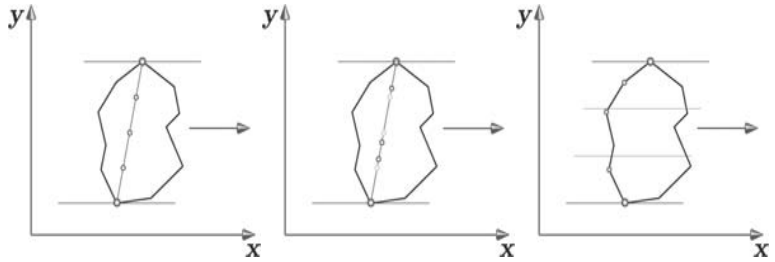


Fig. 4. The limiting vertices in world coordinates $P_{(x,y,z)}$ do not have to be the limiting vertices in screen coordinates $P'_{(x,y)}$. Therefore the coordinates of the object have to be transformed into screen coordinates before the starting points of the motion lines can be determined.



(a) (b)

Fig. 5. Determining the limiting starting points of the motion lines. (a) is the original situation. In (b) the object is rotated into motion direction so the limiting vertices relative to the motion direction can be determined by comparing their y-coordinates.



(a) (b) (c)

Fig. 6**.** Determining the starting points of the motion lines between the limiting ones. Interpolating the limiting vertices (a) results in constant spacing. In (b) the starting points are shifted by a random value. In (c) the vertices at the rear of the object relative to the motion direction are used as starting points.

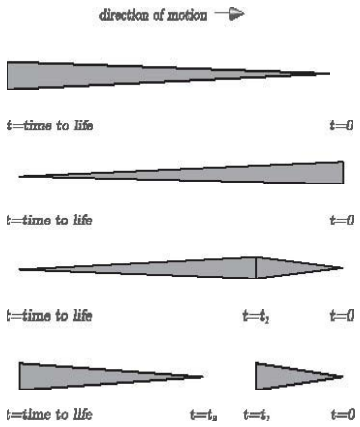


Fig. 7**.** The appearance of the motion lines can be influenced by changing the line width during the life cycle of the particles.

After the starting points have been determined, the motions lines have to be drawn. A particle system is used to draw the motion lines, consisting of a defined number of particle emitters corresponding to the number of motion lines. Each of these emitters releases a defined number of particles. The life cycle of a particle starts with the initialization of defined properties and lasts for a predefined period. During this period

the properties of the particle are changing. When the period is over, the life cycle ends and the particle dies.

Since the vertices describing the starting points are known, a particle emitter can be placed at their position. In every program cycle the emitter releases a particle at the position of the emitter. The position of the emitter is updated every program cycle. The position of the emitted particle stays the same throughout its whole life cycle. All particles belonging to the same emitter are connected with lines.

Figure 7 shows how the appearance of the motion lines can be influenced by changing just one parameter of the particles.

4 IMPLEMENTATION

In order to test our proposed methods, described in Section 3, we implemented them using OpenGL and Cg, the high-level languages for graphics programming developed by nVIDIA. In addition, we used the Halflife model as file format, since it contains geometric as well as animation data. Each of these techniques is implemented as follows:

- *Squash-and-Stretch*: The scaling parameters are calculated based on the speed, acceleration, and direction of the motion of each object. The parameters are given to the vertex program, which handles the scaling of the object. The vertex shader is implemented in Cg, with the advantage that the transformation of the vertices occurs on the graphics card and computing power is saved on the CPU.[1] Figure 8 shows a code fragment of the vertex program responsible for transforming the vertices corresponding to the scaling parameters.
- *Multiple images*: To generate and display multiple images, it is necessary to switch between the two projection modes. The orthogonal projection mode and perspective projection mode indicate how the transformation for the projection looks, and are defined via the projection matrix. In the orthogonal projection mode it is possible to use the two-dimensional screen coordinates. This mode is used to update and display the texture containing the contour replications. The perspective projection is used in perspective mode to render the 3D objects. Figure 9 shows the schematic application flow for generating and displaying multiple images.

```
// 1. Set the position to the model origin
float3 modelOrigin = float3(0, 0, 0);
modelOrigin = VectorTransform ( modelOrigin, boneMatrix );
position.xyz -= modelOrigin;

// 2. Rotate the object based on the direction of the motion
position.xyz = VectorRotate ( position.xyz, motionMatrix );

// 3. Apply the scaling
position.x *= motionScaleX;  // k_result
position.y *= motionScaleYZ; // k_norm
position.z *= motionScaleYZ; // k_norm

// 4. Bring the object to its original orientation
position.xyz = VectorIRotate ( position.xyz, motionMatrix );

// 5. Move the object back to the original position
position.xyz += modelOrigin;
```

Fig. 8: Rudimentary Cg vertex program code for applying scaling parameters.

[1] The graphics card has to support Cg.

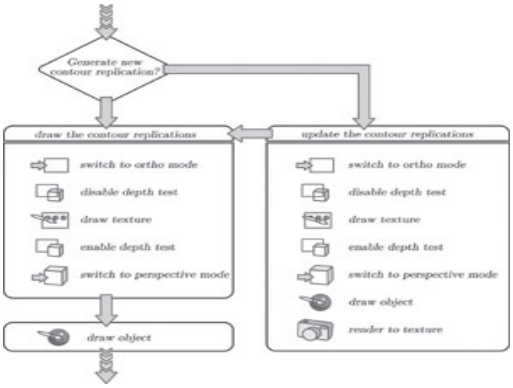


Fig. 9: Schematic flow diagram for generating multiple images and displaying them.

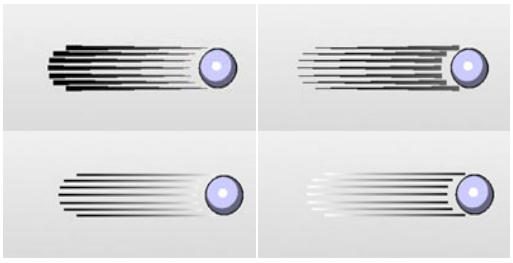


Fig. 10. The style of the lines can be influenced easily by changing the start and end values of the parameters.

- *Motion lines:* Every emitter has different properties that describe the behavior of the emitter and its particles. The most important are the current position of the emitter and the duration of the life cycles of the emitted particles. The length of the motion lines is defined through the duration of the life cycle. Every property of the particle (e.g., line width or color) has a start value and an end value. The different line styles can be achieved with these parameters.

Cg was not used for the implementation of the particle system because there were no apparent benefits from using a vertex or fragment shader to do the rendering. The fact that there is not an exorbitant number of emitters made the decision easier, otherwise using the GPU would make more sense. Indeed, a quick implementation of a particle system is possible on the GPU (a small particle system implementation in Cg is also given in Fernando [2003]), but a more complex particle system with more properties that can be modified by the user would be more difficult to implement using Cg.

5 RESULTS AND DISCUSSION

Figures 11 and 12 depict some results of our implemented prototype. Even at the early stages of development, the techniques looked quite promising; although some of the features were not rendered correctly in the beginning, the results were still quite convincing. This was especially true for the motion lines – the important thing is simply that they appear – it doesn't matter if they are placed correctly; the visual impression improves tremendously. The motion lines technique was the favorite for most people,

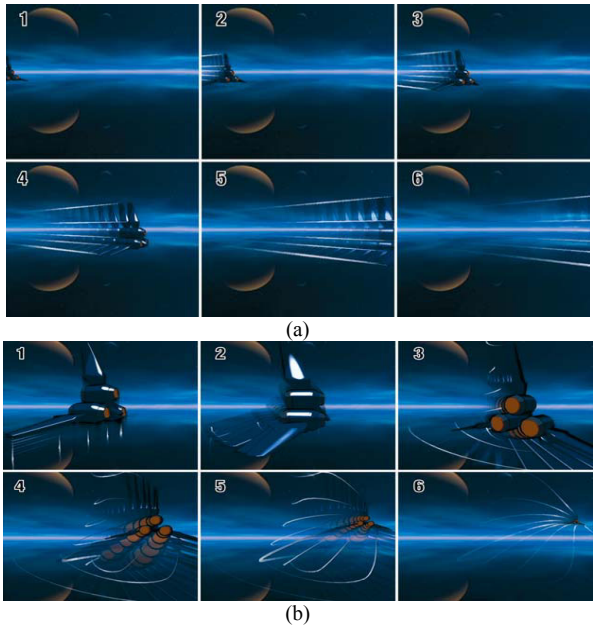


Fig. 11. A combination of the motion line technique and the use of multiple images produces a convincing experience of motion.

first because it offers good visual feedback to the user and second, it emphasizes the complex movements of objects.

Figure 11 shows a simple motion (a) in which the starship is moving forward in a straight line. In this case, even if the ship is out of the screen (cf., last picture), the user gets the information and the impression that the ship does not change its course. Figure 11(b) shows a more complex motion. In this case, the starship does not change its position at the beginning, but is already in action. The user would not immediately recognize that something is happening. Finally, the pictures 4-6 of Figure 11 (b) show that the ship is accelerating and then moving away at a high speed.

The squash-and-stretch technique is important mostly for representing the dynamic behavior of an object and increasing the physical performance of an object (e.g., the elasticity of a rubber ball). Moreover, we get more information about the properties of an object (we assume more about the mass, the inertia, and the surface appearance). In fact, the squash-and-stretch technique extends the duration of motion and collisions. We agree with the Chenny et al. hypothesis that this ensures that people can see the contact and the collision because short-lived events are extended over several frames [Chenny 2002]. On the other hand, this technique has to be chosen carefully: it was interesting to see during our tests that this technique mostly made sense for spaceships or for objects that are really elastic. On the other side it seemed to produce very unrealistic (and unbelievable) results for objects that are well known (e.g. an airplane that was stretched during acceleration).

The advantage of using Cg is that the implementation using the vertex shader is very intuitive and easy to use because there is a direct access to all object vertices, and thus the implementation is accomplished very quickly.

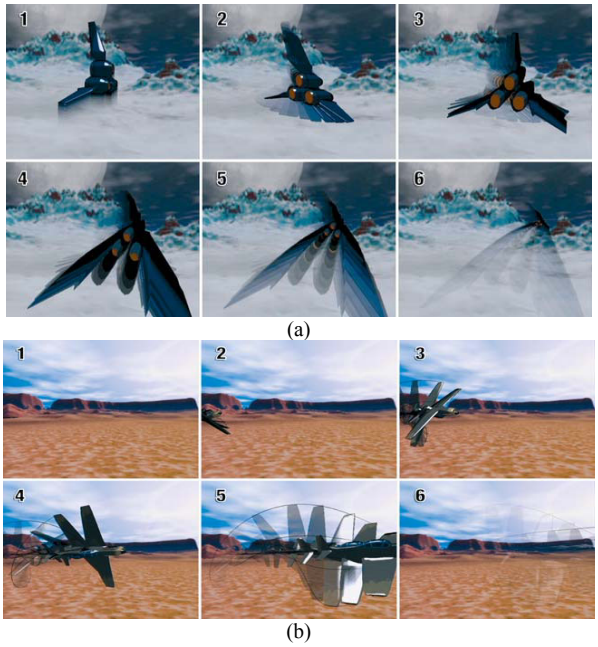


Fig. 12. In the first sections of figure (a), the squash-and-stretch technique is mainly used to render a starship that starts moving at the speed of light. In the second section of figure (b) the complex motion of the airplane is visualized via a combination of motion lines and multiple images.

Finally, the use of multiple blurred images of the object helps to emphasize the motion of a scene**.** We distinguish between the object that is blurred and the background – provided that the camera is tracking and following an object.

Even though most of the metaphors and techniques presented here are mainly used for still images, their use in dynamic images is very interesting, and attractive enough for further research. At the moment, our framework is primarily designed to test the various techniques. But it is also conceivable to use the techniques in an augmented reality environment in which the superimposed (virtual) moving objects can be perceived by the users more easily if they are rendered non-photorealistically using our proposed techniques.

REFERENCES

LASSETER, J. 1987. Principles of traditional animation applied to 3D computer animation. *Computer Graphics 21,* 4 (July 1987), 35-44. (SIGGRAPH 87)

COLLOMOSSE, J. P., ROWNTREE, D. AND HALL, P. M. 2003. Cartoon-style rendering of motion from video. *Vision, Video and Graphics* (July 2003), 117–124.

CHENNEY, S., PINGEL, M., IVERSON, R., AND SZYMANSKI, M. 2002. Simulating cartoon style animation. In *Proceedings of the Second International Symposium on Non-Photorealistic Animation and Rendering* (NPAR-02, June 3–5 2002). ACM Press, New York, 133–138.

FERNANDO, R. AND KILGARD, M. J. 2003. *The Cg Tutorial, The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, Reading, MA, 2003.

FERWERDA, J. 2003. Three varieties of realism in computer graphics. In *Proceedings of the SPIE Conference on Human Vision and Electronic Imaging'03*.

HANL, CH. 2004. Echtzeit-Rendering von Bewegung im Comic-Stil. Diploma thesis, Univ. of Applied Sciences at Hagenberg, Hagenberg, Austria, July 2004.

ISHAYA, V. 2003. Real-time cartoon rendering with direct-X 8.0 hardware. GameDec.net, June 2003. http://www.gamedev.net/reference/articles/article2021.asp.

MASUCH, M., SCHLECHTWEG, S., AND SCHULZ, R. 1999. Speedlines, depicting motion in motionless pictures. In: *SIGGRAPH'99 Conference Abstracts and Applications*. S. 277. ACM, New York, 1999.

MCLOUD, S. 1993. *Understanding Comics-The Invisible Art*. Kitchen Sink Press, 1993.

SCHULZ, R. 1999. Visualisierung von Bewegungen in Liniengrafiken. Diploma thesis, Otto-von-Guericke-Univ. Magdeburg, Fakultät für Informatik, Magdeburg, March 1999.

THOMAS, F. AND JOHNSTON, O.1981. *The Illusion of Life*. Disney Animation, Disney Edition, 1981.

UBISOFT. 2004. http://www.xiii-thegame.com/.

Received October 2004; accepted November 2004