

# Non-photorealistic rendering techniques for motion in computer games

Michael Haller, Christian Hanl, and Jeremiah Diephuis  
[haller|christian.hanl|diephuis]@fh-hagenberg.at  
Upper Austria University of Applied Sciences, Media Technology and Design  
A-4232 Hagenberg

## Abstract

Still images often take advantage of stylized techniques to portray motion. 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 would be desirable. Our approach is focused on three methods to stylize motion: squash-and-stretch, multiple images, and motion lines. These methods for depicting motion in dynamic images are discussed in the course of this paper, and an implementation is presented. Finally, we discuss the results and conclude with an outlook of further development.

**Keywords:** 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 these highly-realistic techniques available in real-time, and 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 [ISHAYA03]. Computer graphics has long been defined as a quest to achieve photorealism. As it gets closer to this goal, the field realizes that there is more to images than realism alone – in particular for game development. And the same programmable hardware makes it possible to render not only photorealistically, but it also allows one 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 the calculation of an image took more than a second. Users expect realistic behavior from worlds that are rendered photorealistically. In games, this requirement can result in a tremendous increase in the level of detail and, with that, necessary workload. It is interesting to see 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 they do not even miss the HDR textures or the bump-mapped polygons in video-quality – they simply want to play. Mark Bolas discovered in 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, “in” the world created. Another aspect is, of course, if we should limit our approach only to the visual point or if we should see it in a more open way. Ferwerda distinguished in [FERWERDA03] three different 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.

When talking of NPR applications, we think of more artistic environments. Non-photorealistic pictures can be more effective at conveying information, more expressive or more beautiful. Virtual objects should be more convincing than realistic. Durand describes in [DURAND02] that the border between photorealism and non-photorealism can be fuzzy and the notion of realism itself is very complex. Another interesting conclusion of his paper is that the virtual world has to be interpreted more convincingly rather than realistically. In other words: it is enough if the virtual, the not existing world, superficially looks real. It should be a believable world. The virtual objects should be expressive, clear, and look aesthetically perfect. A similar example is given by McCloud in [MCCLOUD93] in which he analyzes this statement with two examples: In the first example, he shows an image of a person as a stylized cartoon; in the second image he visualizes the same person in a naturalistically rendered style. McCloud's conclusion is that the stylized image convinces more people than using a photo-realistic, naturalistic rendering style. In his book “Understanding comics”, McCloud's examples mostly address still images. What about motion? Is there a way to express motion? Are there any “rules”, especially for games, that would enhance users' perceptions? Normally, we don't depict and present the motion of objects in computer-generated dynamic images – especially in real-time rendered, dynamic images it is not common to visualize motion. The visualization of motion was previously examined by Marcel Duchamp and in 1912 by the Italian futurists (e.g. Giacomo Balla, Umberto Boccioni etc.). In comics, motion is a very essential thing and McCloud presents different ways to depict motion in still images. In most technical descriptions, motion is portrayed in a number of abstract ways (e.g. by arrows, motion lines, etc.).

Even instruction manuals often use this kind of description to visualize motion and time. Motion is also a very important element for real-time graphics, especially in computer games, and it would be interesting to combine the techniques we know from comics in a real-time environment. The game XIII from Ubisoft was one of the first commercial games with a graphic-

novel presentation (a comic-book visual style is used) and graphics including elements known from the comic-world [UBISOFT04].

As the title implies, this paper shows first of all a novel approach of rendering motion of objects in a real-time environment and 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: using motion lines, using multiple images, and by using the squash-and-stretch technique. Our technique uses the latest generation of hardware, thus we employ a vertex shader and a fragment shader for the presentation of the concepts. The implementation is discussed in section 4. In section 5, we show the results and finally we discuss these results in more detail in section 6, which presents possibilities for further work..

## 2 Related Work

In the early 1930s, animators at the Walt Disney Studio presented the 12 principles of animation. A more detailed description and discussion of all principles can be found in [THOMAS81] – a short summary of the principles has been discussed in [LASSETER87]. These principles were mainly used to guide production and creative discussions as well to train young animators better and faster. The presented 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 has also analyzed the abstract illustration of motion in comics and he describes the following possibilities as the most important for still images [MCCLOUD93]:

- Motion lines, and
- Multiple images

The first method, the motion lines technique, is the most common technique 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 it emphasizes the quality of the dynamics of the illustration. Motion lines are mostly used for still images (cf. [SCHULZ99]), but we believe that their 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 the motion; clever usage of this technique improves the perception of the users a great deal. Using multiple images and thus creating a “blurred” object on the screen often seems to be more realistic than just a normal movement of the object without the blur effect. When objects are moving very fast, our eyes expect to “see” something (some still images, contours, etc.), even if the object is off the screen. Very complex motions can be emphasized by this technique, because users have more time to analyze the movement.

A combination of the two techniques together with the squash-and-stretch technique of Disney’s principles will achieve best results for visualizing motion in real-time. The deformation of an object based on its speed (during the acceleration and the slow-down) is a very important technique to enhance the visual perception. The squash-effect exaggerates the object and the stretch-effect anticipates the collision [CHENNEY02]. 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” a deformation of objects; thus the stretching technique is not realistic, but it makes a bouncing ball appear to be moving 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 et al. in [CHENNEY02].

Collomosse et al. present in [COLLOMOSSE03] that an abstract illustration of motion even makes sense for real movie sequences. In their work, they enhance the motion of movies by adding motion lines or by deforming real objects so that they seem to be moving quickly. The results of their movies are impressive and convincing – unfortunately, their techniques require extensive pre-processing and thus the movies are not rendered in real-time.

## 3 Our approach

We believe that 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 images as well as for dynamic images. All of the three methods can be combined with each other to enhance the graphical behavior of motion. However, we have to guarantee the following requirements and limitations:

- we need detailed geometry data for all objects which we want to apply the technique to,
- the geometry data of the object has to be modifiable,
- the position of the object has to be available at any time, and
- only past motion can be considered, thus future motion can not be considered.

### 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 of the maximum speed and the 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 easily achieved 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  $k_v$  :

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

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:

$$k_a = 1 + \frac{a}{a_{\max}} \cdot (k_{a_{\max}} - 1) \quad | \quad a \geq 0$$

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

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 down and the object is squashed. The closer the absolute value of  $a$  gets to  $a_{\min}$ , 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:

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

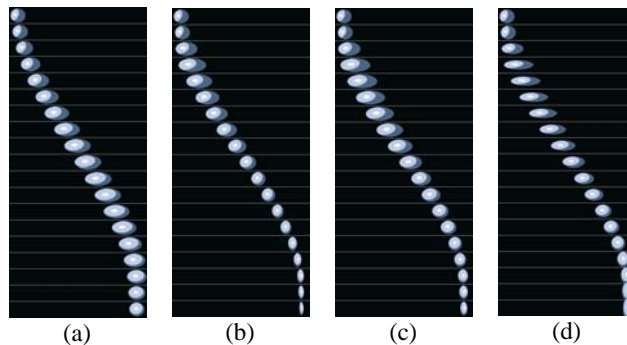
If the volume of the object is considered, the scaling parameter  $k_{norm}$  has to be normal to the direction of the motion:

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

By rotating the object in the direction of the motion, the scaling can be controlled by a single parameter  $s$  [CHENNEY02]. Consequently, a single transformation matrix can be found:

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

Figure 1 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.



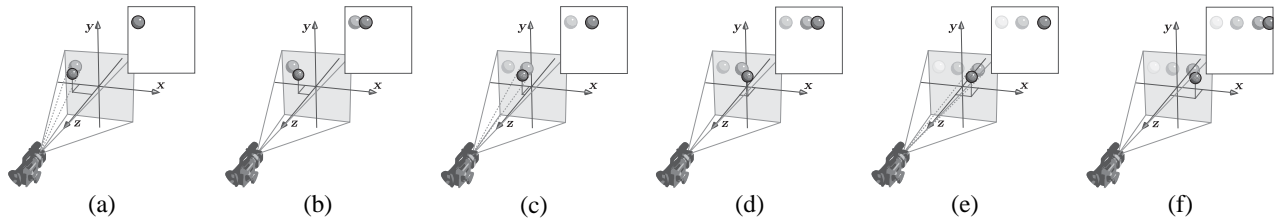
**Figure 1:** Time slices of an animation of a ball speeding up and slowing down horizontally. The squash and stretch based on: (a) speed, (b) acceleration, (c) and (d) speed and acceleration. In (d) the volume of the ball is kept constant.

## Multiple images

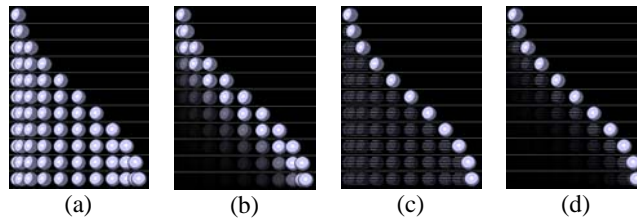
For every constant time interval, which can be configured, snapshots of the moving object are taken. Those contour replications of the object are still shown if the animation progresses and the object move on. In [MCLOUD93] McCloud describes this phenomenon as multiple images. The goal is to more deeply 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 a scene is displayed with 60 fps and the replication rate  $n$  is 20, then a contour replication is generated every third frame. One way to generate multiple images would be to draw the object again for every contour replication based on its position. A great deal of computing power would be required to do so. 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 of the 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 different styles of multiple images.



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



**Figure 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.

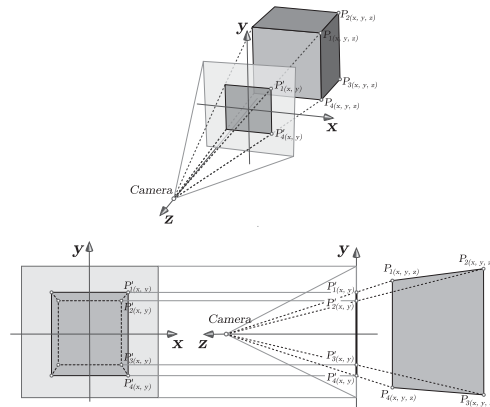
## Motion lines

Motion lines have been used for quite a long time to convey the sense of motion. Based on the medium, culture and of course on the personal style of the artists, a number of different types emerged. Therefore, several different parameters are necessary to influence the appearance of the motion lines when generated automatically. Schulz specified in [SCHULZ99] a couple of requirements for the motion lines used for still images:

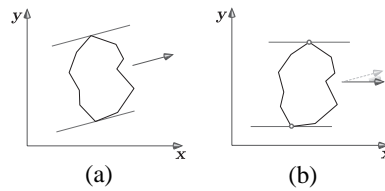
1. The upper and the lower motion lines relative to the motion direction have to be at the same level as the maximum extent of the object (A1).
2. Motion lines are to be placed on significant regions of the object (A2).
3. The number of motion lines has to be configurable (A3).
4. The space between the 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 the motion lines has to be configurable (A7).

To calculate the starting points of the motion lines, the geometry data of the object is used. Every object consists of triangles defined by vertices. Therefore, 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

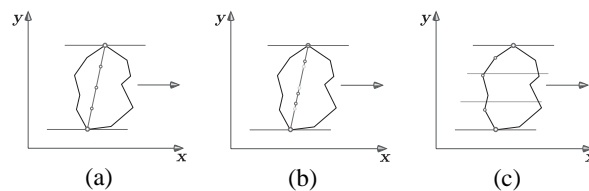
well. To determine the starting points of the motion lines, the positions of the vertices are transformed into screen coordinates (cf. Figure 4). To get the starting points more easily, the object is rotated to the motion direction as shown in Figure 5. Now the limiting vertices can be determined by comparing their coordinates of the y-axis. To get the starting points for the motion lines 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 starting point.



**Figure 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.

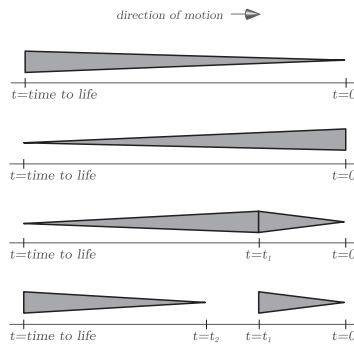


**Figure 5:** Determining the limiting starting points of the motion lines. (a) is the original situation. In (b) the object is rotated into motion direction so the limiting vertices relative to the motion direction can be determined by comparing their y-coordinates.



**Figure 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.

After the starting points have been determined, the motions lines have to be drawn. A particle system is used to draw the motion lines. It consists 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 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.



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

Since the vertices describing the starting points are known, a particle emitter can be placed at their position. 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, which were described in section 3, we implemented them using OpenGL and Cg, the high-level language for graphics programming developed by nVIDIA. In addition, we used the Halflife model as file format, because it contains geometry 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. Those 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<sup>1</sup>. Figure 8 shows a code fragment of the vertex program responsible for transforming the vertices corresponding to the scaling parameters.

---

```

// 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;

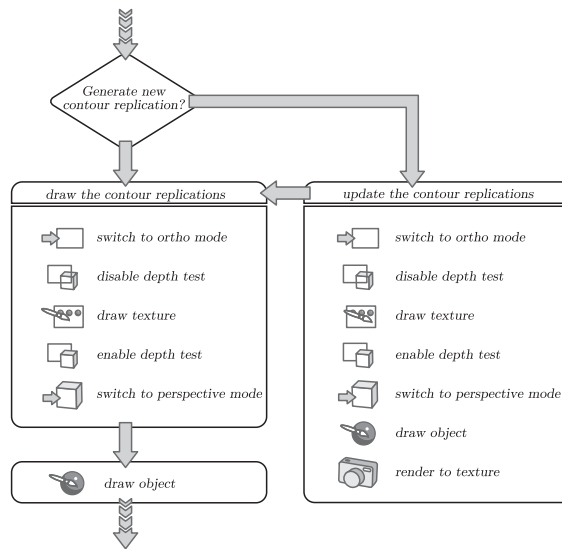
```

---

**Figure 8:** Rudimentary Cg vertex program code for applying the scaling parameters.

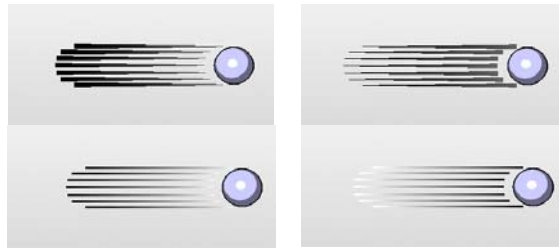
- **Multiple images:** To generate and to display multiple images, it is necessary to switch between the two projection modes. The orthogonal projection mode and perspective projection mode indicate how the transformation for the projection looks and are defined through the projection matrix. In the 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.

<sup>1</sup> The graphics card has to support Cg.



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

- Motion lines:** Every emitter has different properties which 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 cycle of the particles emitted. 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. With these parameters; the different line styles can be achieved.

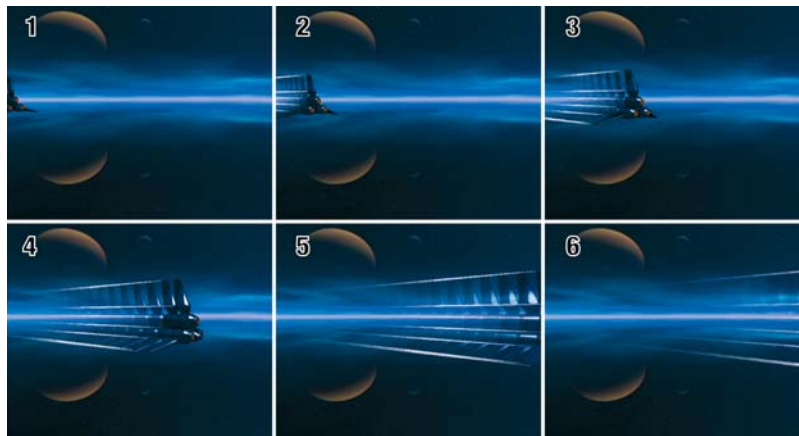


**Figure 10:** By changing the start and the end value of the parameters, the style of the lines can be easily influenced.

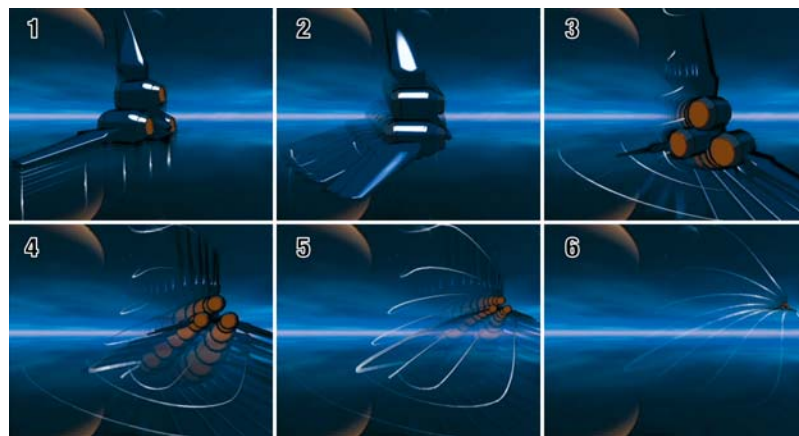
Cg was not used for the implementation of the particle system because there were no apparent benefits from using a vertex or fragment shader to do the rendering. The fact that there is not an exorbitant number of emitters made the decision easier – otherwise the usage of the GPU would make more sense. Indeed, a quick implementation of a particle system would be possible on the GPU (a small particle system implementation in Cg is also given in [FERNANDO03]), 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

Figure 11 and Figure 12 depict some results of our implemented prototype. Even at the early stages of development, the techniques looked quite promising. Some of the features were not rendered correctly in the beginning, but 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 will be improved tremendously. The motion lines technique was the favorite for most people. The reason is firstly because it offers good visual feedback to the user and, secondly, it emphasizes complex movements of objects.



(a)



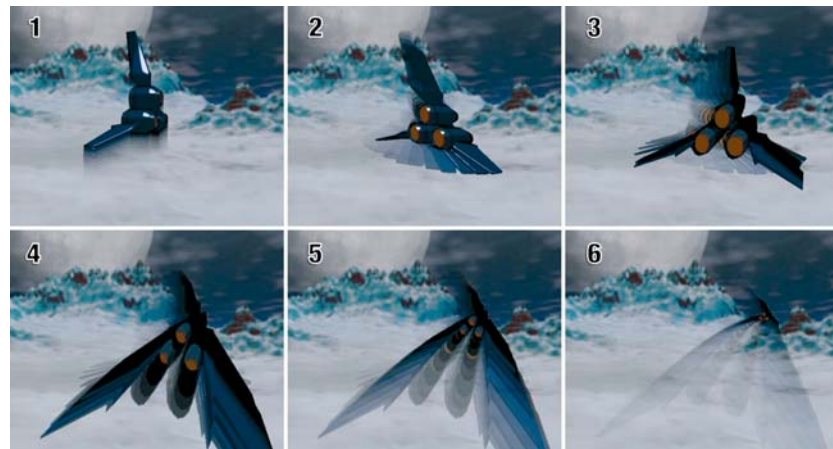
(b)

**Figure 11:** A combination of the motion line technique and the usage of multiple images produces a convincing motion experience

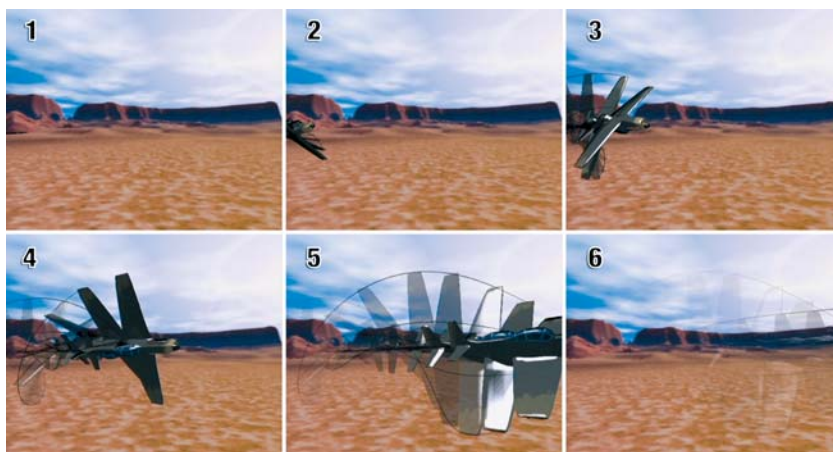
Figure 11 shows a simple motion (a) in which the starship is moving forward in a straight line. In this case, even if the ship is out of the screen (cf. last picture), the user gets the information and the impression that the ship does not change its course. Figure 11 b shows a more complex motion. In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

The squash-and-stretch technique is mostly important to represent the dynamic behavior of an object and to increase 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 hypothesis of Chenny et al. that this ensures that people can see the contact and the collision, because the short-lived events are extended over several frames (cf. [CHENNY02]). On the other hand, this technique has to be chosen carefully: during our tests, it was interesting to see that this technique mostly made sense for spaceships or for objects that are really elastic – but it seemed to be a very unrealistic (and unbelievable) technique for other objects, such as an airplane that was stretched during its acceleration.

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



(a)



(b)

**Figure 12:** In the first pictures of figure (a), the squash-and-stretch technique is mainly used for rendering a starship that starts moving at the speed of light. In the second pictures of figure (b) the complex motion of the airplane is visualized by a combination of motion lines and multiple images.

Finally, the usage of multiple images of the object which are blurred helps to emphasize the scene. We distinguished between the object that is blurred and the background – provided that the camera is tracking and following an object.

Even though most of the presented metaphors and techniques are mainly used for still images, their usage in dynamic images was highly interesting and attractive enough for further research. At the moment, the framework is primarily designed for testing the different techniques and the main goal was to use the different techniques in a game environment. But it would also be conceivable to use these techniques in an Augmented Reality environment in which the superimposed (virtual) objects can be perceived by the users more easily if they are rendered non-photorealistically. This field in particular offers great potential for the use of such non-photorealistic techniques to visualize virtual objects.

## References

LASSETER, J.: *Principles of Traditional Animation Applied to 3D Computer Animation*. Bd. 21, p. 35–44, July 1987.

COLLOMOSSE, J. P., D. ROWNTREE UND P. M. HALL: *Cartoon-Style Rendering of Motion from Video*. In: *Vision, Video and Graphics*, pp. 117–124, July 2003.

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

HANL, CH.: *Echtzeit-Rendering von Bewegung im Comic-Stil*. Diploma thesis, University of Applied Sciences at Hagenberg, Media technology and –design, Hagenberg, Austria, July 2004.

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

MCLOUD, S.: *Understanding Comics-The invisible Art*. Kitchen Sink Press, United States of America, 1993.

ISHAYA V.: Real-Time Cartoon Rendering with Direct-X 8.0 Hardware, GameDec.net, June 2003, <http://www.gamedev.net/reference/articles/article2021.asp>.

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

MASUCH M., SCHLECHTWEG S., SCHULZ R.: *Speedlines, Depicting Motion in Motionless Pictures*. In: SIGGRAPH'99 Conference Abstracts and Applications, S. 277, ACM SIGGRAPH, 1999.

UBISOFT, <http://www.xiii-thegame.com/>, 2004.

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