FlowGames*

Jakob Santner Manuel Werlberger Thomas Mauthner Wolfgang Paier Horst Bischof
Institute for Computer Graphics and Vision, Graz University of Technology
{santner,werlberger,mauthner,paier,bischof}@icg.tugraz.at

Abstract

Computer vision-based interfaces to games hold the promise of rich natural interaction and thus a more realistic gaming experience. Therefore, the video games industry started to develop and market computer vision-based games recently with great success. Due to limited computational resources, they employ mostly simple algorithms such as background subtraction, instead of sophisticated motion estimation or gesture recognition methods. This not only results in a lack of robustness, but also in very limited interaction possibilities and thus reduced gaming experience.

In this paper, we show a couple of concepts to control video games based on optical flow. We use a state-of-the-art optical flow algorithm able to be computed densely in real-time on GPUs, which are in fact built-in in nearly every gaming hardware available. Based on the estimated motion, we develop several computer games with increasing complexity: Starting with using the flow field as force acting on moveable objects, we span the spectrum to more sophisticated concepts such as controlling widgets and action recognition.

1. Introduction

The way a human player and a computer game interact is a crucial element for gaming experience. This led to the development of various input devices, ranging from mice and keyboards to gamepads, from joysticks to steering wheels, from light guns to complex airplane cockpits. While these devices rely on the interpretation of mechanical motion solely, also computer vision-based interfaces have been of great interest recently: *Sony* shipped over 10 million units of their camera systems *EyeToy* and *PlayStation Eye*. *Microsoft* currently develops a camera-based interface under the code name *Project Natal* to replace their *Xbox Live Vision* camera. Besides video game consoles (*Playstation 3*, *XBox 360*, *Wii etc.*) the most important game platforms comprise desktop and laptop computers as well as handheld devices (mobile phones, PDAs, MP3 players *etc.*), which often have built-in camera modules.

*This work was supported by the Austrian Research Promotion Agency (FFG) within the projects VM-GPU (813396) and vdQA (816003) as well as the Austrian Science Fund (FWF) under the doctoral program Confluence of Vision and Graphics W1209. We also greatly acknowledge Nvidia for their valuable support.

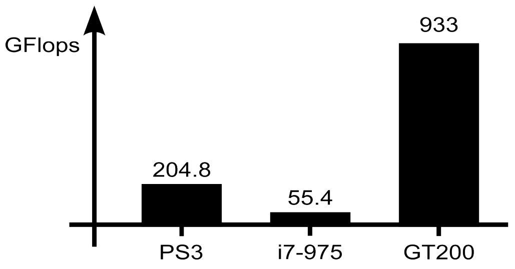

Figure 1. Theoretical peak performance of current gaming systems: Playstation 3 (Cell processor), Intel i7-975 (CPU), Nvidia GT200 (GPU)

To employ a computer vision algorithm for video game control, two basic requirements need to be fulfilled: First, computational efficiency is crucial for convenient user interaction. Second, the method needs to be robust against illumination changes and occlusions, invariant to the player's appearance *etc.* Many state-of-the-art computer vision algorithms are either far from achieving realtime performance, or not robust enough be be employed in end-user software applications. Therefore, computer vision-based games rely on rather basic computer vision methods: McGuire and Jenkins [11] describe color blob detection, face recognition by cross-correlation and background subtraction as the current methods of choice.

Concerning the computational performance of algorithms, one has to take the trend towards multicore processing into account: The *XBox 360* uses a 3-core *PowerPC* processor, recent desktop CPUs have four and more independent processing units, and the *Playstation 3* features a *Cell* processor with seven cores. Above all, nearly every gaming platform has a GPU with up to several hundreds of processing cores. These parallel processing units led to a boost in computational performance, allowing to build a TFlops computer with standard consumer hardware (Figure 1). This vast performance potential can of course only be tapped by highly parallelizable algorithms. For instance, variational motion estimation algorithms, which took minutes to compute in single threaded implementations, run in realtime exploiting the parallel architecture of GPUs.

Hence, in this work we focus on optical flow-based user interaction. In contrast to background subtraction, which only detects movement between frames, optical flow also estimates the direction and speed of the movement. In 1996, Freeman *et al*. [6] developed a basic optical flow algorithm and demonstrated its applicability to control a video game. They designed their system to work under controlled environments with uniform background and constant illumination. Zivkovic [17] used an optical flow algorithm based on the well known work of Lucas and Kanade [10] to create simple interactive control elements. He achieved realtime performance by computing optical flow only on tiny subparts of the whole image. In rather limited experiments, he showed the superiority of his approach over background subtraction. Hämäläinen *et al*. [7] used the same flow algorithm on silhouette pixels for an interactive martial arts game.

As we will show in this paper, the most common optical flow algorithms (*i.e.* Lukas and Kanade [10], Horn and Schunck [8]) yield imprecise results and cannot be computed in realtime for reasonable image sizes. Werlberger *et al*. [15] developed a fast optical flow method yielding high accuracy in the Middlebury optical flow benchmark [1]. They estimated flow densely by minimizing an energy functional using GPUs and reached realtime performance on VGA resolution (640x480).

In this paper, we demonstrate different ways of human–computer interaction using the highly accurate motion estimation technique of Werlberger *et al*. [15]. In Section 2, we will describe their method and compare it to other optical flow algorithms in terms of speed and accuracy. After showing increasingly more sophisticated concepts to use optical flow as input to computer games in Section 3, we will conclude this work and give an outlook in Section 4.

2. Optical Flow

The computation of apparent motion is a key problem in computer vision and has been studied a lot over the past years. For an overview of the vast amount of proposed methods, we refer to the surveys [1, 2, 5, 14].

In their seminal work, Horn and Schunck [8] started to embed the estimation of optical flow into a variational framework: Given two sequential input images I_0, I_1 and the unknown motion field $\mathbf{u} = (u_1, u_2)^T : \Omega \rightarrow \mathbb{R}^2$, they used a quadratic regularization- ($\mathcal{R}(\cdot)$) and data-term ($\mathcal{D}(\cdot)$) in a functional of the form

$$\min_{\mathbf{u}} \Big\{ \mathcal{R}(\mathbf{u}) + \mathcal{D}(I_0, I_1, \mathbf{u}) \Big\} . \qquad (1)$$

While the data term forces the flow field to be consistent with the motion in the input images, the regularization term ensures the smoothness of the flow field. Over time, many modifications have been made to both regularization and data term to improve the quality of the estimated optical flow field. To compare different algorithms, Baker *et al*. [1] initiated the Middlebury evaluation database for optical flow.

For the appliance of optical flow for interactive video game control, we need the estimated flow field as fast and accurate as possible. Investigating the works published through the Middlebury evaluation database yields only a few methods closing this gap. One of these methods is the work of Werlberger *et al*. [15], which is real-time capable of computing accurate dense optical flow. In addition, they provide a GPU-based library on their website [1], which allows for easy integration and testing.

In contrast to previous robust methods, which employed L^1 penalization of the flow-field in the regularization term, Werlberger *et al*. use an anisotropic Huber regularization in their Huber-L^1 model. The anisotropy enables the functional to take directional edges into account and impede the regularization across motion boundaries. In addition, the Huber-norm reduces the zero-filling effects (staircasing artifacts) of the L^1-norm. In detail, they solve the optimization problem

$$\min_{\mathbf{u}} \left\{ \int_\Omega |q_1|_\epsilon + |q_2|_\epsilon \, dx + \lambda \int_\Omega |\rho(x)| \, dx \right\} , \qquad (2)$$

where $|\cdot|_\epsilon$ denotes the Huber regularity, $q_d = D^{1/2}\nabla \mathbf{u}_d$ the diffusivity-driven regularization and $|\rho(x)|$ a robust L^1 data term penalizing the linearized image residual $\rho(x) = I_0(x) - I_1(x + \mathbf{u}(x))$. For details on the used primal-dual optimization scheme, refer to [15, 16].

Qualitative Comparison In order to show the benefit of the method of Werlberger *et al.*, we compare it to other publicly available algorithms. Therefore, we utilized the methods of Horn and Schunck [8], Lucas and Kanade [10] as well as Farnebäck [4], which are implemented in the open-source framework OpenCV.

Figure 2 shows the resulting flow fields of these methods using the color-coding of the Middlebury benchmark database. The hue and saturation of the pixels represent the direction and magnitude of the optical flow vectors respectively. While the implementations included in the OpenCV toolbox show heavy outliers and imprecise motion boundaries (see Figure 2(d)–2(f)), the method of Werlberger *et al.* yields a smooth flow field with sharp motion boundaries.

[1] http://gpu4vision.org

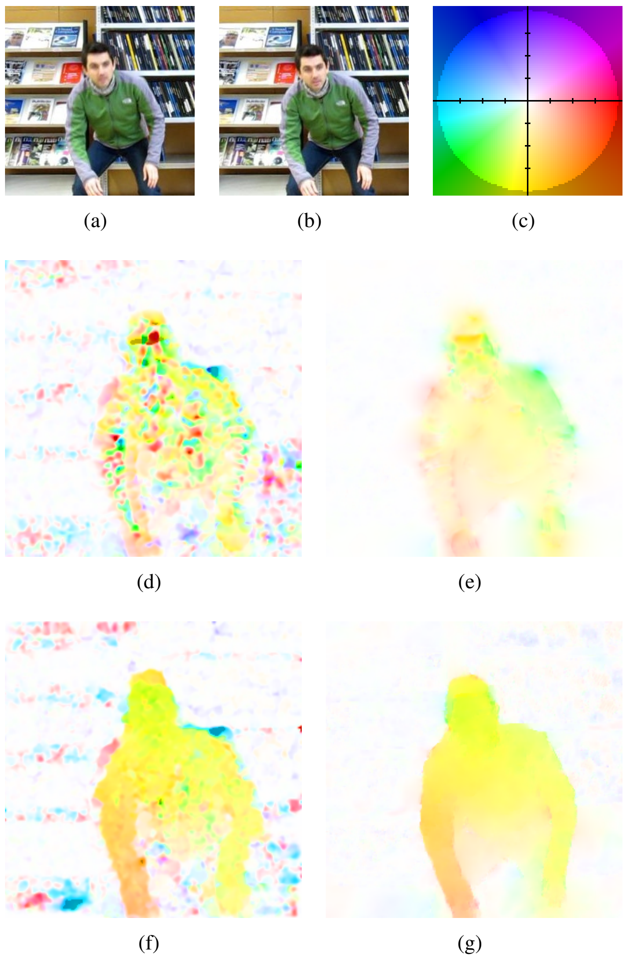

Figure 2. Comparing different algorithms for optical flow estimation on two sequential image frames (a,b): (d) Lucas and Kanade [10], (e) Horn and Schunck [8], (f) Farnebäck [4] and (g) Werlberger *et al.* [15]. The color-coding of the flow fields is given in (c).

Runtime As stated before, computational efficiency is the knock-out criterion for interactive applications. The majority of cited methods in the Middlebury database have a reported runtime of several minutes and above. In Figure 2, we computed optical flow on 256x256 images on a standard desktop PC featuring a 2.6 GHz Intel quadcore processor and a NVIDIA Geforce GTX280 GPU. The OpenCV algorithms take at least several seconds per frame pair, while the GPU-based implementation of Werlberger *et al.* finishes in 0.02 seconds.

3. Interaction

In this section, we show several examples for optical flow-based user interaction. Please note that we do not present quantitative results in terms of *e.g.* usability studies. Instead, we describe interaction methods, that proved to work robustly under different conditions (changing illumination, cluttered background *etc.*) with users of varying sex, age and computer experience.

3.1. Input

The optical flow algorithm computes a dense two-dimensional flow field $\mathbf{u} = (u_1, u_2)^T$ between two images I_0 and I_1 of width X and height Y. Throughout the following examples, we denote the magnitude of the flow field

$$|\mathbf{u}| = \sqrt{u_1^2 + u_2^2} \tag{3}$$

and its direction

$$\angle\mathbf{u} = atan2(u_2, u_1). \tag{4}$$

3.2. Average Flow

The easiest way to control a game using optical flow is to employ the arithmetic mean of the entire flow field

$$\bar{\mathbf{u}} = \frac{1}{XY}\sum_{x=1}^{X}\sum_{y=1}^{Y}\mathbf{u}(x,y). \tag{5}$$

The resulting average displacement in x and y direction can be interpreted the same way as the two degrees of freedom (DOF) signal of a joystick or a mouse: As an example, we reimplemented the control of a game called *Neverball* [2], where the player has to navigate a marble through a maze by tilting the playfield (see Figure 3). Given pitch θ and roll φ angles of the playfield at a given time t, we increment these angles with the average optical flow such that

$$\begin{pmatrix}\varphi\\ \theta\end{pmatrix}_{t+1} = \begin{pmatrix}\varphi\\ \theta\end{pmatrix}_{t} + k\cdot\bar{\mathbf{u}}. \tag{6}$$

The weighing constant k can be used to adjust the sensitivity of the control. Hence, when the player moves to the left in front of the camera, the roll angle of the playfield is changed such that the marble starts to accelerate leftwards. Besides being extremely simple to create, this style of game control is also intuitive to learn and play: In empirical studies, people managed to play this game without any instruction or teaching phase successfully after a few seconds. Furthermore, this concept can be applied to a variety of computer games, *e.g.*simple flight simulators or racing games.

3.3. Moving Objects

The concept presented above can be easily extended to interacting with moving objects such as balls in a ball game. Therefore, we apply basic Newtonian mechanics: Using Newton's second law, we can compute the acceleration $\mathbf{a}$

[2]http://neverball.org

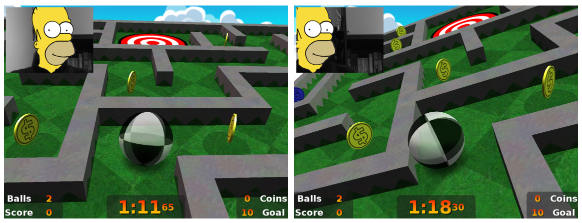

Figure 3. Using the arithmetic mean of a dense flow field as game control: As Homer moves his head to the left, the playfield is tilted according to the average optical flow, making the marble roll leftwards.

of a body of mass m when we know all forces $\mathbf{F}_i$ the body is exposed to:

$$\mathbf{a} = \frac{\sum \mathbf{F}_i}{m} \tag{7}$$

The average optical flow over the sensitive area of an arbitrary body $|\overline{\mathbf{u}_\mathcal{B}}|$ can be interpreted as physical force $\mathbf{F}_\mathcal{B}$. This and a gravity force $\mathbf{F}_\mathcal{G}$ are enough to create a ball game (see Figure 4), where the ball is accelerated according to

$$\frac{\mathbf{F}_\mathcal{B} + \mathbf{F}_\mathcal{G}}{m}. \tag{8}$$

The derivation of a motion equation for that purpose is described in detail in [11]. Figure 5 shows a complete volleyball-like game with two players, a ball and a net.

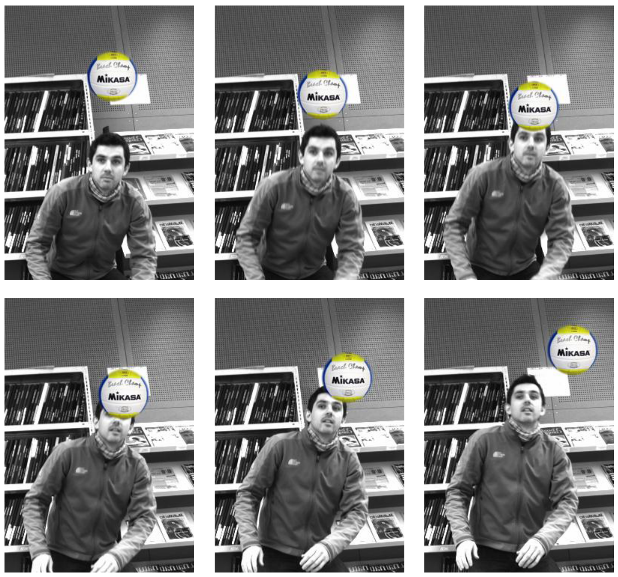
Figure 4. A virtual ball exposed to two forces: In the first row, the gravitational force accelerates the ball downwards. In the second row, the movement of the player creates a force acting in the opposite direction, making the ball change its trajectory.

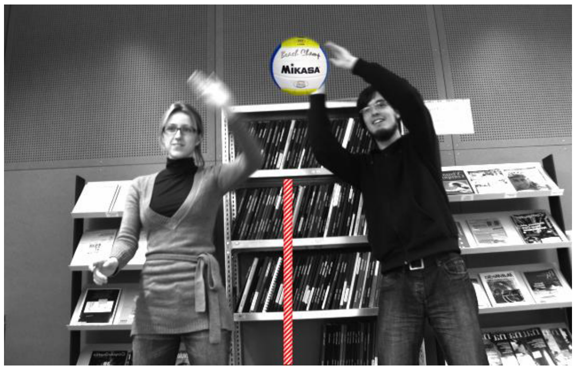
Figure 5. A volleyball-like game controlled with optical flow.

3.4. Widgets

Widgets are the elementary parts of a graphical user interface. Based on well known and widely accepted concepts (such as *e.g.* triggering an action by clicking a virtual button), they provide a basic set of intuitive interaction points between humans and software. In the following, we show how to use optical flow to control buttons, sliders, scrollbars, knobs and jog dials. Note that many other basic widgets such as *e.g.* checkboxes, radiobuttons or spinners can be adapted based on the concepts we present in this section. We then show two different games with interfaces based solely on these widgets.

Buttons Triggering buttons using computer vision has been done before (*e.g.* in [17]), however, we describe it here for the sake of completeness.

Given $\mathcal{B}$ a set of pixel positions x, y representing a sensitive area of arbitrary shape, we denote

$$\mathbf{u}_\mathcal{B} = \mathbf{u}(x, y) \quad \forall \quad x, y \in \mathcal{B} \tag{9}$$

as all flow vectors within that area. The number of flow vectors denote as the magnitude of the set $|\mathcal{B}|$. See Figure 6 for an example: There, $\mathbf{u}_\mathcal{B}$ consists of all flow vectors lying under the button represented by the green rectangle. A button is considered as being activated, when a trigger function

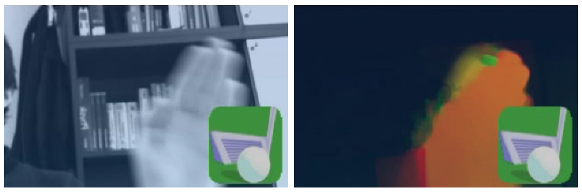
Figure 6. A button operated with optical flow: The first image shows a hand moving towards a virtual button, the second image shows the corresponding optical flow in color coded representation. When the average flow magnitude under the button exceeds a certain threshold, the button is triggered.

exceeds a given threshold $f(\mathbf{u}_{\mathcal{B}}) > \theta$. The arithmetic mean of the flow magnitude

$$f(\mathbf{u}_{\mathcal{B}}) = \overline{|\mathbf{u}_{\mathcal{B}}|} = \frac{1}{|\mathcal{B}|}\sum |\mathbf{u}_{\mathcal{B}}| \tag{10}$$

can be used as trigger function, other possibilities include *e.g.*, the maximum or the median of the flow magnitude. We add hysteresis to the trigger function in order to avoid multiple activations per movement. By taking the angle of the flow vectors into account, direction-sensitive buttons can be realized.

Scrolling In conventional graphical user interfaces, widgets such as scrollbars, sliders and spinboxes are employed to input arbitrary values (see Figure 7).

These scrolling widgets exhibit a preferred orientation γ, which is usually the horizontal or vertical direction. They represent an arbitrary value ϕ, which can be changed by clicking arrows or dragging sliders: Spinboxes have two discrete buttons, one to increase and one to decrease the widget's value. The benefit of sliders and scrollbars over spinboxes lies in the dragging mode, where a user can quickly change the value of the slider over a wide range with a single interaction. This dragging mode can be easily adapted using optical flow: Considering $\mathbf{u}_{\mathcal{B}}$ as all flow vectors within the sensitive area of a scrolling widget, we denote the average direction of $\mathbf{u}_{\mathcal{B}}$ as

$$\overline{\angle \mathbf{u}_{\mathcal{B}}} = \frac{1}{|\mathcal{B}|}\sum \angle \mathbf{u}_{\mathcal{B}}. \tag{11}$$

We can add the average flow magnitude $\overline{|\mathbf{u}_{\mathcal{B}}|}$ to the value ϕ of the scrolling widget, when the mean flow direction aligns with the preferred orientation γ up to a threshold θ:

$$\phi_{t+1} = \begin{cases} \phi_t + \overline{|\mathbf{u}_{\mathcal{B}}|} & \text{if} \quad |\gamma - \overline{\angle \mathbf{u}_{\mathcal{B}}}| < \theta \\ \phi_t - \overline{|\mathbf{u}_{\mathcal{B}}|} & \text{if} \quad |\gamma + 180^\circ - \overline{\angle \mathbf{u}_{\mathcal{B}}}| < \theta \\ \phi_t & \text{else} \end{cases} \tag{12}$$

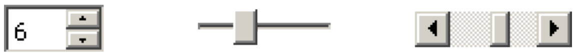

Figure 7. Different widgets used for the input of arbitrary values: Spinbox, slider, scrollbar. Note that each widget has either the horizontal or vertical direction as preferred orientation.

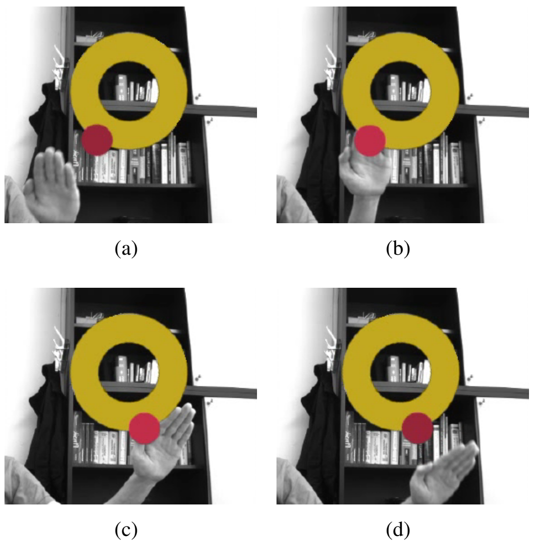

Figure 8. An interactive jog dial widget: In (a), the hand is outside of the sensitive area, thus nothing happens. As the hand moves under the red dot (b,c), the jog dial starts to turn according to the movement, until the hand leaves the sensitive area again (d).

Jog Dial, Knob In contrast to linearly shaped sliders, jog dials and knobs are widgets with circular shape. Considering $\mathbf{u}_{\mathcal{B}}$ as the flow values under the red dot in Figure 8, we can apply the same update rules as for the sliders in the previous paragraph, except that the orientation γ depends on the current value ϕ. Figure 9 shows a steering wheel based on the same principles using two sensitive areas.

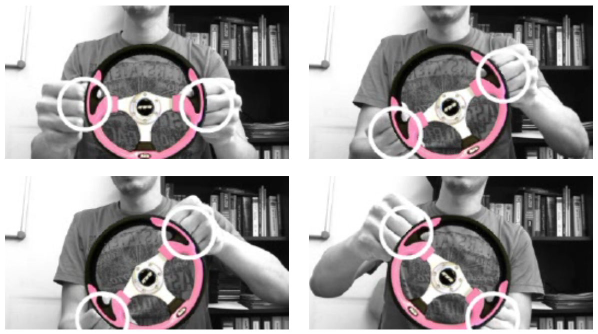

Figure 9. A steering wheel based on the same principles as the jog dial in Figure 8. Two sensitive areas allow for robust and convenient usage.

Minigolf Game With the presented widgets, we control an open source minigolf game called *Neverputt* [3]: We use a slider to adjust the speed of the hit, a jog dial to set its direction and a button to perform the hit. We overlaid the camera input and the widgets to the game display (see Figure 10).

Racing Game A more sophisticated example in this work is a kart-racing game based on the open source project *SuperTuxKart* [4]. Here we need five widgets: A steering wheel,

[3]http://neverball.org
[4]http://supertuxkart.sourceforge.net

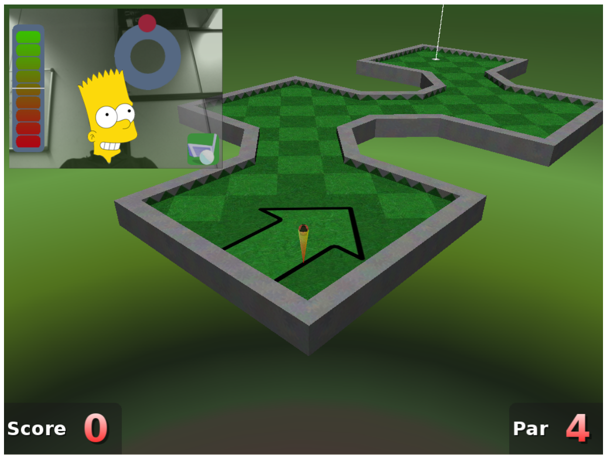

Figure 10. A minigolf game based on optical flow: Notice the interaction panel showing Bart Simpson on the top left: The speed of the golf ball can be controlled with the slider on the left, the direction with the jog dial in the middle. To play the ball, there is a green button on the bottom right side of the interaction panel.

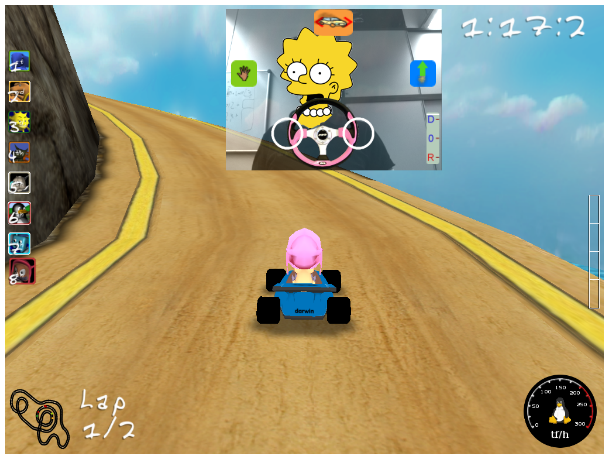

Figure 11. Controlling a cart racing game with optical flow only: The interaction panel shows a steering wheel in the middle, and a slider for throttle to the right. There are also three buttons for attack (green), drifting (orange) and boost(blue).

a slider for the throttle and three buttons for attack, drift and boost. As the amount of controls increases, their position gets more and more important: Considering the game screenshot in Figure 11, one might notice that the blue button (boost) and the slider (throttle) on the right side are very close together. Given a user sitting relaxed with both hands on the steering wheel, it gets very difficult for him to trigger the blue button without accidentally touching the throttle slider. It turned out during experiments, that users have to pay too much attention on triggering the right widget, when their positions are badly chosen. This is where ergonomics start to play an important role: When designing an optical flow-based user interface with several controls, one needs to choose the widget positions very carefully.

3.5. Human Action Recognition

A more sophisticated interaction method is to directly classify specific human actions. In contrast to the concepts presented so far, where optical flow information controls elements or widgets directly, human action recognition can be employed to trigger events by detecting and recognizing a set of predefined distinctive movements. Although such methods are of growing interest in the computer vision community (for an overview refer to the survey of [13]), their utilization for game control is very limited so far.

Many action recognition methods rely on the estimation of optical flow: Ke *et al*. [9] classified human actions by segmenting and describing large spatial-temporal volumes. They showed impressive results, however, due to the computational complexity such representations are impracticable for real-time interaction. Computationally more efficient is the method of Roth *et al*. [12]. They demonstrated accurate classification of specific human actions by incorporating the information of only a small number of subsequent frames.

In their work they estimated Histograms of Oriented Gradients (HOG) for the current frame as well as Histograms of Flow (HOF) for the corresponding dense flow field estimated from two subsequent frames. Both descriptors are then efficiently represented by the coefficients of a Nonnegative Matrix Factorization (NMF). For the final action classification, they applied an efficient cascaded Linear Discriminant Analysis (CLDA) classifier.

In order to demonstrate the applicability of human action recognition for controlling video games, we employed the approach of Roth *et al*. [12] to trigger the four actions of a Tetris game. Therefore, we modified an open source Tetris implementation called Snip[5]. There are four commands needed to control a Tetris game: Shifting the blocks left- and rightwards, rotating the blocks and placing them. We chose four actions that match these commands intuitively: Waving the left or right hand shifts the block leftwards or rightwards. Jumping in place rotates the block, a jumping-jack places it (see Figure 12).

The training of the action recognition was performed offline on the publicly available Weizmann[6] dataset [3], containing 81 low resolution videos (180×144) of nine subjects performing ten different actions. We trained a five class classifier for wave-left, wave-right, jumping in place, jumping-jack and an additional neutral class incorporating all other actions covered in the database.

[5]http://snip.sourceforge.net/index.html

[6]http://www.wisdom.weizmann.ac.il/ vision/SpaceTimeActions.html

We reach about 1 classification per second with unoptimized code on a single core Pentium 4 processor, which allows to play Tetris in earlier levels (where the velocity of the bricks is moderate) conveniently.

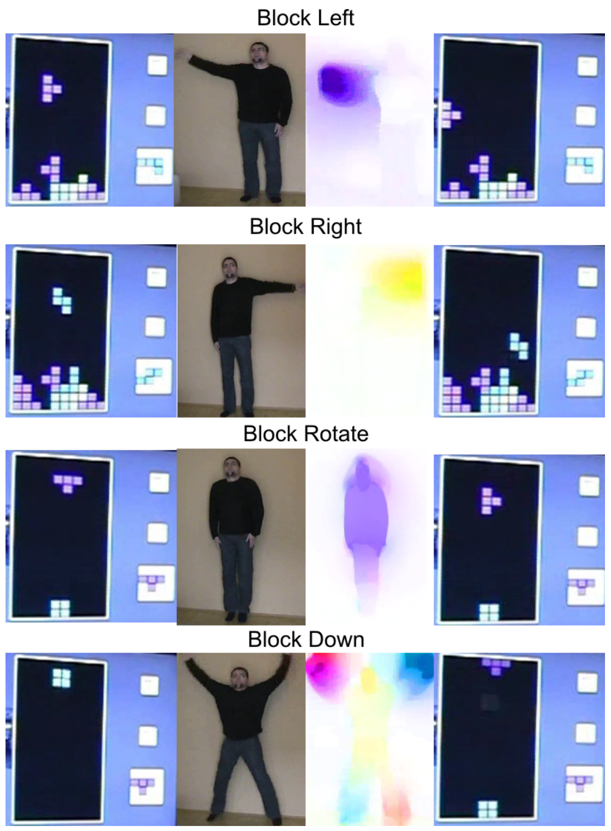

Figure 12. Action Tetris: The left column shows screen shots of a Tetris game situation. Performing one of the pretrained actions (shown in the middle as appearance and corresponding flow field) triggers the desired command (right column).

4. Conclusion

In this work, we demonstrated the applicability of state-of-the-art optical flow algorithms to control video games. We discussed and evaluated several motion estimation methods in terms of speed and accuracy, and selected a GPU-based energy minimization algorithm of Werlberger *et al*. [15] for its ability to compute accurate dense flow fields in realtime. Based on these flow fields, we developed several interaction mechanisms, ranging from controlling basic widgets over applying forces on moving objects up to triggering actions by detecting distinctive movements. We showed the feasibility of these mechanisms with several small video games. The only requirement for running these games is a standard webcam and a GPU, both of these are readily available in most gaming systems.

References

[1] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. Technical Report MSR-TR-2009-179, Microsoft Research, 2009. 2

[2] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Computing Surveys*, 27(3), 1995. 2

[3] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. In *Proc. ICCV*, pages 1395–1402, 2005. 6

[4] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proc. of the 13th Scandinavian Conf. on Image Analysis*, LNCS 2749, pages 363–370, Gothenburg, Sweden, June-July 2003. 2, 3

[5] D. J. Fleet and Y. Weiss. Optical flow estimation. In N. Paragios, Y. Chen, and O. Faugeras, editors, *Handbook of Mathematical Models in Computer Vision*, chapter 15, pages 239–258. Springer, 2006. 2

[6] W. T. Freeman, K. Tanaka, J. Ohta, and K. Kyuma. Computer vision for computer games. *IEEE Int. Conf. on Automatic Face and Gesture Recognition*, 0:100, 1996. 2

[7] P. Hämäläinen, T. Ilmonen, J. Höysniemi, M. Lindholm, and A. Nykänen. Martial arts in artificial reality. In *CHI '05: Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 781–790, New York, NY, USA, 2005. ACM. 2

[8] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence, 17*, pages 185–203, 1981. 2, 3

[9] Y. Ke, R. Sukthankar, and M. Hebert1. Event detection in crowded videos. In *Proc. ICCV*, 2007. 6

[10] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI81*, pages 674–679, 1981. 2, 3

[11] M. McGuire and O. C. Jenkins. *Creating Games: Mechanics, Content, and Technology*. A. K. Peters, 2008. 1, 4

[12] P. M. Roth, T. Mauthner, I. Khan, and H. Bischof. Efficient human action recognition by cascaded linear classifcation. In *1st IEEE Workshop on Video-Oriented Object and Event Classification*, 2009. 6

[13] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 18:1473–1488, 2008. 6

[14] J. Weickert, A. Bruhn, T. Brox, and N. Papenberg. A survey on variational optic flow methods for small displacements. In *Mathematical Models for Registration and Applications to Medical Imaging*. Springer, Berlin, 2006. 2

[15] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic Huber-L1 optical flow. In *Proc. of the British Machine Vision Conf.*, 2009. 2, 3, 7

[16] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *Pattern Recognition (Proc. DAGM)*, 2007. 2

[17] Z. Zivkovic. Optical-flow-driven gadgets for gaming user interface. In *3rd Int. Conf. on Entertainment Computing*, pages 90–100, 2004. 2, 4