Video-Based Head Tracking for High-Performance Games

Conference Paper · May 2013

DOI: 10.13140/RG.2.1.3732.7843

CITATIONS
0

READS
49

3 authors:

Andrei Sherstyuk
University of Hawai'i System
52 PUBLICATIONS **594** CITATIONS
SEE PROFILE

Anton Treskunov
30 PUBLICATIONS **220** CITATIONS
SEE PROFILE

Vladimir Savchenko
Hosei University
118 PUBLICATIONS **1,741** CITATIONS
SEE PROFILE

Video-Based Head Tracking for High-Performance Games

Andrei Sherstyuk
Sunflower Research
andrei.sherstyuk@gmail.com

Anton Treskunov
Samsung
anton.t@sisa.samsung.com

Vladimir Savchenko
Hosei University
vsavchen@k.hosei.ac.jp

Abstract

The recent advent of video-based tracking technologies allowed to bring natural head motion to any 3D application, including games. However, video tracking is a CPU-intensive process, which may have a negative impact on game performance. In this work, we examine this impact for different types of 3D content, using a game prototype built with two advanced components, CryENGINE2 platform and faceAPI head tracking system. Our findings indicate that cost of video tracking is negligible. We provide detail on our system implementation and performance analysis. Also, we present a number of new techniques of controlling user avatars in social 3D games, based on head motion.

1 Introduction

Bringing natural human motion into virtual environments involves balancing between hardware cost, the tracking range and the fidelity of the motion data. For example, both Sony EyeToy and Microsoft Kinect systems are capable of full body tracking. However, Sony EyeToy, which is ten times less expensive, is designed for processing wide motions and gestures and lacks precision needed for tracking subtle head movements.

Until recently, reliable tracking of head motion required special hardware: accelerometers, magnetic or optical trackers. For certain systems, per-user calibration or special operating conditions were also needed. Yet, use of natural head motion was widely regarded as the future of games [1].

New webcam-based tracking technologies deliver high-quality motion data on consumer hardware, practically on every desktop. That, in turn, allows to integrate natural head motion into 3D games easily. However, most modern game engines are computationally demanding, often pushing host systems to their limits, both for CPU and GPU tasks. Will head tracking still be useful in such extreme conditions?

1.1 Goals and Methods

In this project, we investigate whether single-camera head tracking is practical and computationally affordable for modern games. We approach this problem by building and testing a game prototype, integrating a high-end photorealistic game platform *CryENGINE2* from Crytek [2] with a state-of-the-art *faceAPI* head tracking software from Seeing Machines [3]. Both systems are briefly described below.

1.2 The System Components

CryENGINE2 made its debut with *Crysis* game, which set new standards of visual quality for first person shooters. CryENGINE2 was used to build *Blue Mars Online* 3D world [5], featuring photo-realistic avatars, with dynamically simulated hair and layers of cloth. CryENGINE2 was also used by creators of *Entropia Universe* adventure game, which holds the record of hosting the most expensive piece of virtual estate ever purchased with real money.

FaceAPI tracking engine was released to public in 2010. This system generates high-quality head motion data with 6 degrees of freedom at 30 Hz, from a single web camera. Presently, faceAPI is available for Windows OS; the forthcoming release will also have support for Linux and Mac OS. A public license provides head rotation and orientation data; a commercial version will also track position of facial features, such as eyes, mouth and eyebrows. Detailed technical specifications of faceAPI engine are available at the manufacturer's web-site [4].

2 Previous Work

Since it release, faceAPI was rapidly gaining recognition in research community. It was used in studies on estimating gaze direction from eye appearance, with free user head rotation [6]. Marks *et al* investigated optimal operating conditions for faceAPI [7]. FaceAPI was used to implement gestural communications in shared virtual training environments [8].

FaceAPI received much attention from game developers as well. Sko and Gardner described a number of ways how faceAPI can be applied for gaming tasks [9], using Valve game engine. There are multiple reports in integration of faceAPI with Unity3D engine (e.g., [10]). However, as of this writing, there is no published work on evaluating faceAPI with "heavy-weight" game engines, such as CryENGINE, Unreal or Unigine. Thus, the question whether faceAPI is an affordable addition to high-quality games remains open.

We approached this question in a practical manner, by building and evaluating a game prototype, using faceAPI and CryENGINE2 components.

3 Game Prototype Implementation

Software: We used a non-commercial version of *faceAPI* which tracks position and rotation of user head at 30 Hz. For gaming environment, we used *City Editor* from Blue Mars SDK, built with CryENGINE2 and used with permission from Avatar-Reality Inc, the creators of Blue Mars Online world [5]. The Editor allows to load and explore 3D scenes in a game mode, with added options related to head tracking.

Configuration: Data exchange between the Editor and faceAPI was implemented and tested in two configurations: (a) client/server configuration, with faceAPI running as a separate application and serving head pose data upon request from the Editor via dedicated socket; (b) faceAPI compiled directly with the Editor. The latter solution appeared more convenient and allowed faster initialization of tracking: 1-2 seconds as compared to 2-3 seconds in case (a).

Denoising tracking data: Each head pose (rotation and translation) provided by faceAPI is accompanied by *confidence* value, which indicates how much the data can be trusted. Confidence drops to near-zero values in poor lighting conditions and when the user is moving away from the camera field of view. As defined, confidence could be used to attenuate the pose data, in order to reduce noise in peripheral areas. However, confidence itself is noisy and needs to be smoothed prior using. Instead, we opted to use an explicitly defined bell-shaped attenuation function:

$$a(d) = (1 + s^2 d^2)^{-2}$$

Here d is the azimuth of user head in camera space, parameter s defines the slope. Setting $s = 0.08$ yielded useful results (see Figure 1). For every pose, the head rotation and translation values are multiplied by $a(d)$, which allowed to fade the effect of tracking in and out, as the head enters and leaves the camera viewing range.

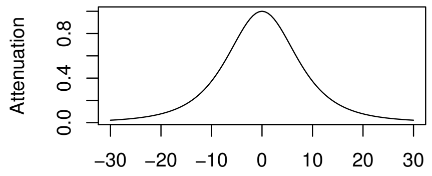

Figure 1. Attenuation function, defined over camera horizontal viewing range (degrees).

Hardware: For testing, two systems were used:

1. Sony Vaio CW laptop, 4 GB RAM, 2.53 GHz Intel Core2 Duo CPU, nVidia GeForce GT 230M card.
2. Samsung RF510 laptop, 4 GB RAM, 1.60GHz Intel Core i7 CPU, nVidia GeForce GT 330M card.

As indicated, both systems have relatively modest hardware, which made them sensitive to variations in computational load. This turned out to be an advantage, allowing for detecting and measuring changes in game engine performance due to tracking.

4 Tracking and Game Performance

The preliminary tests were conducted on scenes with mixed content, that included both static objects and dynamically deformed objects, such as user avatars. One of those scenes is shown in Figure 2. The results showed that the impact of tracking on rendering speed was negligible. In simple scenes that were rendered faster than 30 FPS, frame rate dropped 10% each time when faceAPI was started, and remained low until the user face was acquired by the engine (1-2 seconds). Then, FPS recovered to its initial value. In complex scenes (over 1.5 M triangles, FPS < 30), starting and stopping tracking had no noticeable effect on FPS.

To investigate further, we measured the system performance in scenes with separate types of content: static meshes that use little of CPU time and dynamic objects that require processing on CPU. For that purpose, we used the standard avatar model from Blue Mars SDK, with cloth and hair simulated by mass-spring models. In static tests scenes, simulation was turned off and avatars were rendered as static meshes. In dynamic tests, the avatars were running the "standing-idle" animation, with cloth and hair deformed on CPU at every frame, illustrated in Figure 3.

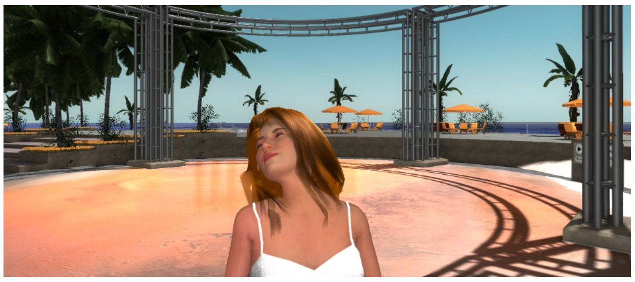
Figure 2. Welcome Area in Blue Mars 3D world (1.51 M triangles @ 30 FPS). The avatar head is tilted back and sideways, copying user head pose. Tracking has no impact on FPS in this scene.

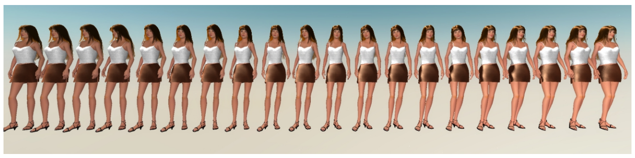
Figure 3. Test scene with 20 clothed hairstyled animated avatars, processed on CPU.

Table 1. Game performance (FPS): tracker off (-) and on (+). FPS values in static scenes, shown as single numbers, were not affected by tracking.

objects : tris	system 1		system 2	
	static	dynamic - / +	static	dynamic - / +
1 : 0.13 M	76	73 / 73	81	76 / 76
2 : 0.25 M	73	71 / 71	79	72 / 72
3 : 0.37 M	72	56 / 55	77	71 / 70
4 : 0.49 M	71	48 / 44	76	69 / 65
5 : 0.61 M	71	40 / 36	75	60 / 55
6 : 0.73 M	70	33 / 32	74	50 / 45
7 : 0.85 M	70	30 / 27	73	44 / 39
8 : 0.97 M	69	27 / 24	72	40 / 36
9 : 1.09 M	69	25 / 19	72	35 / 30
10 : 1.21 M	69	20 / 14	71	31 / 28
11 : 1.33 M	68	16 / 12	70	29 / 24
12 : 1.45 M	68	13 / 9	70	25 / 20
13 : 1.57 M	68	9 / 7	69	23 / 18
14 : 1.69 M	67	8 / 6	69	20 / 15
15 : 1.81 M	67	8 / 5	69	16 / 12
16 : 1.93 M	67	7 / 4	69	15 / 11
17 : 2.05 M	66	6 / 4	68	14 / 11
18 : 2.17 M	66	5 / 3	68	14 / 10
19 : 2.29 M	66	5 / 3	68	13 / 9
20 : 2.40 M	66	4 / 3	68	11 / 8

For static and dynamic modes, the scene size was varied from 1 to 20 avatar objects. Each scene was rendered twice, with and without head tracking. The test results are plotted in Figure 4 and listed in Table 1.

The test results allowed us to draw two conclusions.

Non-existing impact in static scenes. For static 3D content with GPU-bound rendering, tracking has zero impact on performance. As Table 1 shows, frame rate slowly decays from 76 to 66 FPS (81 to 68, for system 2), as the number of objects increases from 1 to 20. These values remain unchanged, when tracking is turned on and off.

Insignificant impact in dynamic scenes. When the scene content is predominantly dynamic, the game performance is limited by CPU. In this case, tracking has a measurable effect on rendering speed, with mean slowdown of 2.6 and 3.7 FPS, for systems 1 and 2, respectively. On both systems, the cost of tracking does not depend on the scene size.

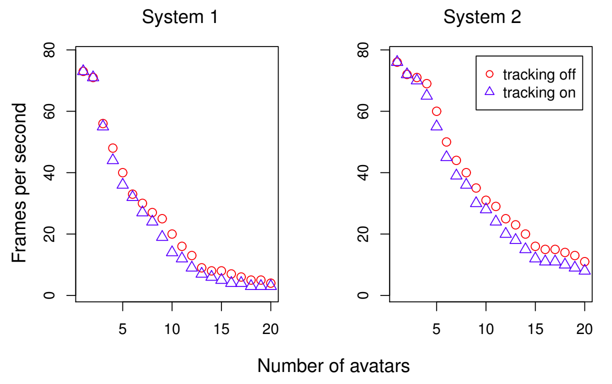

Figure 4. Impact of tracking on rendering dynamic objects (see Figure 3). Tracking becomes noticeable in scenes with more than 2 avatars, at average costs of 2.6 and 3.7 FPS.

The dynamic test scenes used in our study represent the worst case scenario, which practically never happens in real game scenes. Normally, game engines try to minimize the CPU load by lowering update rate on dynamic objects when their pixel footprint is low. In the Editor, no such optimizations take places, showing the overly conservative measurements.

To summarize our findings: on multi-core platforms, the impact of video-based head tracking varies from non-existing to low, including the worst case scenario with all-dynamic scene content. We conclude that video-based head tracking is computationally affordable for high-end 3D games. In the next section, we will present several cases of practical application of natural head motion, in the context of 3D social games.

5 Practical Application of Natural Head Motion in Social Games

The Blue Mars SDK provides a number of techniques for controlling avatar behavior and appearance. By adding head tracking, we created several novel applications of these techniques, that will be described next.

5.1 Avatar Pose Control

This is the most straightforward example of motion data transfer. The user head rotation is applied to the avatar neck joint, making the avatar reproduce user head movements. Head rotation is added in a layered fashion, blending user motion with the current avatar pose. Although the technique is simple, it allows to create expressive poses, demonstrated in Figure 5.

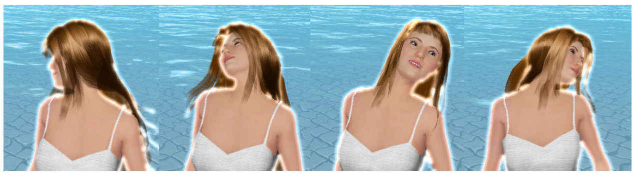

Figure 5. Direct transfer of player's head rotation to avatar's neck joint. The rotation is added to the currently active animation (e.g., idle motion), blending the two motions smoothly and yielding a variety of natural looking movements.

5.2 Changing Avatar Facial Expressions

In Blue Mars, avatars have an in-built social behavior that makes them temporarily direct their heads and eyes towards other avatars, when they enter the avatar's viewing range. This feature is called "look-around", and it helps to convey a message to other players that their presence is noted.

We improved this automatic behavior using head rotation. While the avatar's eyes remain locked on the object of interest (i.e., the other avatar's face), additional head rotations produce new facial expressions, such as teasing, disbelief, or turning someones nose up, as illustrated in Figure 6.

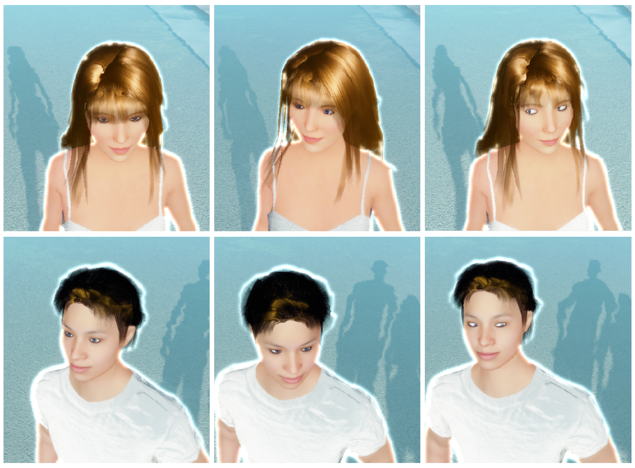

Figure 6. Two avatars are facing each other. Left pair: normal idle pose when neither avatar shows signs of noticing its neighbor. Middle pair: "look-around" feature is in effect, making the avatars look at each other. Right pair: user head rotation is added, changing the avatar's facial expressions.

5.3 Avatar Awareness of Player Presence

In third-person view, the "look-around" behavior can also be directed towards the players themselves. In this mode, the player's location is defined by the virtual camera position. By turning its head and eyes towards the camera, the avatar appears looking straight in the player's eyes, as shown in Figure 7. This feature can be refined by adjusting the camera position by the physical displacement of the player's head. As the result, the avatar will trace the player's head movements, while the "look-around" feature is in effect. This behavior will strengthen the player's impression that their avatars are aware of the player's presence.

5.4 Personalizing Avatar Behavior

All user-created motions can be recorded and later reused, either on explicit command, such as key press, or embedded into autonomous behaviors, as the look-around feature, described above. In latter case, one can record a personalized head gesture, for example, a friendly nod, that will be displayed when a recognized "friend" avatar appears nearby. Conversely, recently un-friended players may be greeted by a pre-recorded head motion, indicating displeasure, for instance, turning head away. Such personalized autonomous behaviors will support the illusion of presence, even when the player is temporarily away from keyboard.

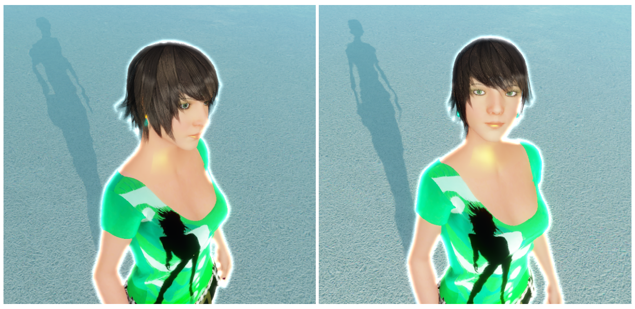

Figure 7. Avatar awareness of player presence. Left: idle behavior. Right: attention on player. Tracking will make the avatar continuously maintain eye contact with the player.

5.5 Head Motion and Camera Control

In immersive VR systems that utilize head mounted displays (HMD), head motion is almost always directly transferred to camera position and orientation, using one-to-one mapping. Exceptions are made only for systems that aim to compensate for limited field of view of an HMD, by amplifying horizontal or vertical head rotation. In Augmented Reality systems the rule of direct motion transfer is even more strict.

On the contrary, non-immersive games are more flexible about camera controls, providing a variety of viewing options, such as free-camera mode, third-person or aerial view. Thus, user head motion can be treated as *loosely coupled* with various viewing tasks. As an example, we implemented camera-sliding technique, which moves the virtual camera sideways, when the user rotate their head left of right, for more than 30 degrees. This technique appears useful in scenes where occluding objects are present, at close range. The test scene is shown in Figure 8.

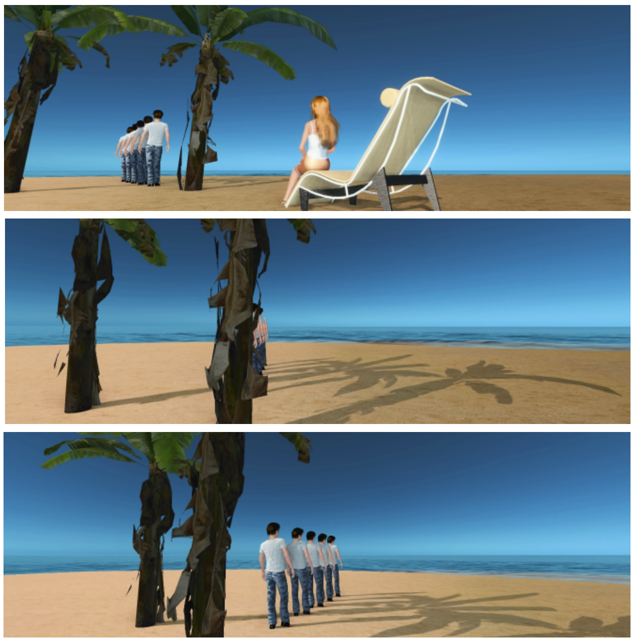

Figure 8. Camera sliding for a simple counting task. Top: external view. Center: female player's view, with all objects of interest occluded by a tree. Below: prompted by used head rotation, camera slides to the right, removing occlusion.

6 Conclusions

We have presented results of the case study on integrating *faceAPI* tracking system with *CryENGINE2* high-performance game engine and these results are very encouraging. In our experimental settings with multi-core platforms, the impact of motion tracking varied from non-existing to low, including the worst case scenario with all-dynamic scene content. That proves that camera-based motion tracking is an affordable technology for photo-realistic 3D games.

We also presented a number of novel techniques for controlling user avatar appearance and behavior, based on natural head motion. These techniques demonstrate that head tracking is a powerful extension to traditional game controls, especially in the context of 3D social worlds, where head motion can be particularly effective.

We conclude that motion tracking is not only a practical, but also an enabling technology for 3D games. We showed how user head movements can enhance players' interaction in social worlds, when their avatars are in close proximity to each other. Nodding, head shaking and more subtle uses of body language, such as gaze averting or seeking eye contact – these are but a few examples of new ways for players to express themselves. These new interfaces, enabled by head tracking, constitute a rich ground for further research.

References

[1] J. J. LaViola Jr.: "Bringing VR and Spatial 3D Interaction to the Masses Through Video Games," *Computer Graphics and Applications*, pp. 10 - 15, 2008.

[2] Crytek, `http://www.crytek.com/`

[3] faceAPI, `http://www.seeingmachines.com/product/faceapi/`

[4] faceAPI Specifications, `http://www.seeingmachines.com/product/faceapi/specifications/`

[5] Blue Mars Online, `http://www.bluemars.com`

[6] F. Lu, T. Okabe, Y. Sugano, Y. Sato: "A Head Pose-free Approach for Appearance-based Gaze Estimation," *BMVC 2011*, http://dx.doi.org/10.5244/C.25.126, 2011.

[7] S. Marks, J. Windsor, B. Wünsche: "Optimisation and Comparison Framework for Monocular Camera-based Face Tracking," *IVCNZ'09 24th International Conference*, pp. 243 - 248, 2009.

[8] S. Marks, J. Windsor, B. Wünsche: "Head Tracking Based Avatar Control for Virtual Environment Teamwork Training," *International Conference on Computer Graphics Theory and Applications (GRAPP)*, pp. 257 - 269, 2009.

[9] T. Sko and H. Gardner: "Head Tracking in First-Person Games: Interaction Using a Web-Camera," *International Conference on Human-Computer Interaction*, pp. 342 - 355, 2009.

[10] Unity 3D faceAPI Tutorials, `http://forum.unity3d.com/threads/69364-Unity-3D-faceAPI-Tutorials`