Computers & Graphics 36 (2012) 651–657

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Stream-based animation of real-time crowd scenes [☆]

Wayne Daniel Lister [*], Andy Day

School of Computing Sciences, University of East Anglia, Norwich NR4 7TJ, United Kingdom

ARTICLE INFO

Article history:
Received 10 May 2011
Received in revised form
16 February 2012
Accepted 20 February 2012
Available online 13 March 2012

Keywords:
Crowd simulation
Animation
OpenCL

ABSTRACT

We present a new way of drawing a crowd of animated characters in real-time. Previous work has focused almost exclusively on how to visualize ever larger crowd scenes and the current state-of-the-art can display tens of thousands of virtual humans with ease. The associated trade-off, however, is that crowd members can do little more than play a set of scripted motion clips. It follows that designating individuals to be members of a crowd instantly limits the techniques that can be used, the behaviours that can be depicted and ultimately, the perceived realism of a scene. Our approach differs from the state-of-the-art in that we do not propose a crowd-specific technique but instead a bone-parallel, OpenCL-accelerated interpretation of the traditional character pipeline. The method does not rely on pre-processing; provides fine-grained control over the animation of a crowd (support for motion blending and varied skeletons, for example) and crowd members and user-controlled 'hero' characters can be handled without distinction.

1. Introduction

Many real-time applications have the need to display a crowd of autonomous characters. Examples can be found in the fields of cultural heritage visualization [1,2] and urban simulation [3,4] whilst recent years have seen the gaming industry adopt an increasingly prominent role [5,6]. Here members of a crowd can be cast as soldiers in battle, pedestrians in a city, and stadium or trackside spectators. Crowd simulation continues to elicit a significant body of work as developers strive to make their virtual worlds ever more realistic. Believable crowds are an important addition because even the most immersive of environments can appear post-apocalyptic without its inhabitants [7].

An effective crowd visualization consists of two main parts; behaviour simulation and rendering. In this paper we consider the rendering side of the problem and here the longstanding requirement has been to display as large and as diverse a crowd as possible. Processing a large number of animated characters is a computationally demanding problem and early researchers had little choice but to explore aggressive level-of-detail techniques. These were then relaxed in view of graphics hardware advances so as to improve the quality of individuals. First impostors were used to represent members of a crowd [8,9], then hybrid impostor/geometry systems [3,4] and finally standalone mesh-based techniques once GPUs were shown to deliver the required character throughput [10]. The current state-of-the-art [11,12] can display tens of thousands of skinned virtual humans in real-time but the associated trade-off is that crowd members can do little more than play a set of scripted motion clips. It follows that by designating individuals to be members of a crowd, one instantly limits the techniques that can be used, the behaviours that can be depicted and ultimately, the perceived realism of a scene.

In this paper we describe a new way to visualize a crowd of animated characters in real-time. Our approach differs from the state-of-the-art in that we do not propose a crowd-specific technique but instead a bone-parallel, OpenCL-accelerated interpretation of the traditional character pipeline. The key advantages of our method when compared to crowd rendering techniques such as those in [11,13] are that it does not rely on pre-processing; it provides developers with far more control over the animation of their crowds (by way of support for motion blending and varied skeletons for example) and crowd members and user-controlled 'hero' characters can be handled without distinction.

2. Motivation

Drawing a crowd presents two separate challenges: that of rendering a large number of virtual humans and that of introducing plausible crowd variety. Significant progress has been made in recent years and many of the fundamental problems can now be considered solved. Work in [11,12] uses a combination of

[☆] This article was recommended for publication by W. Wang.
[*] Corresponding author.
E-mail addresses: w.lister@uea.ac.uk (W.D. Lister), Andy.Day@uea.ac.uk (A. Day).

0097-8493/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cag.2012.02.014

hardware skinning and geometry instancing to visualize many thousands of skin-animated characters. Work in [14,15] presents a number of techniques to achieve crowd variety and in [16,17] the authors investigate where best to focus artistic and computational resources so as to maximize the perception of variety by a user. LOD techniques in terms of geometry, skeleton and motion simplification are considered in [18] but despite all of these advances, there still remains a clear disparity between members of a crowd and 'regular' characters processed by the traditional character pipeline. One of the reasons for this disparity can be found by examining the skeletal animation equation (1). The position $v(t)$ of a vertex at time t is given by its position, u, in pose-space $T_{o,i}$ and a weighted blend of k bone transformations T^t_i. Skinning then transforms each vertex to follow the skeletal motion:

$$v(t) = \left(\sum_{i=1}^{k} w_i T^t_i T^{-1}_{o,i} \right) u \tag{1}$$

Drawing a crowd in this way requires a significant amount of computation but performance can be improved by evaluating the set of all possible $T^t_i T^{-1}_{o,i}$ terms as a pre-process [11]. This method forms the state-of-the-art but its use of pre-computation introduces several limitations. First of all, animations are bound to a specific skeleton and a copy of the data must be stored for each character model, which limits the number of models that can be used. Second, all animations have to be known in advance and techniques such as motion blending, IK and ragdolls cannot be supported. None of this has been considered a problem in previous years because most real-time applications have demanded only very basic functionality from their crowd systems. However it is inevitable that users' expectations will soon start to rise. In a typical urban simulation for example, it is desirable for pedestrians to be able to look in shop windows, get in and out of vehicles and check for traffic before walking across a road. These scenarios are difficult to depict by way of crowd rendering techniques alone and it follows that a truly believable visualization must incorporate two separate character systems; a fast crowd pipeline for the many thousands of 'unimportant' crowd members and the traditional 'hero character' pipeline for everybody else.

In this paper we target crowds of up to around 10,000 virtual humans in size and argue that using dedicated crowd techniques to animate and render these scenes no longer makes any sense. Work in [19] considers crowd motion blending for example and in [20] a specialist gaze controller is used to make crowd members look-at points of interest in their environment. Such techniques tend to be required by very few crowd members at a time and yet handling these individuals requires a significant amount of effort, often duplicating functionality that is already provided by the traditional character pipeline. We therefore approach the problem of animating a crowd from the opposite direction. Instead of developing crowd-specific techniques, why not take the traditional character pipeline and try to build a crowd system?

3. Stream-based skeletal animation

Our technique differs from the state-of-the-art in that Eq. (1) is evaluated at runtime and for all crowd members, every frame. The problem is modelled by using a pipeline of three OpenCL kernels called *Pose*, *Transform* and *Write-Back* (Fig. 1). The *Pose* kernel orients bones in pose-space by sampling an animation clip; the *Transform* kernel propagates bone transformations down a skeleton hierarchy and the *Write-Back* kernel multiplies bones by the inverse bind pose to create matrix palettes. The results are passed

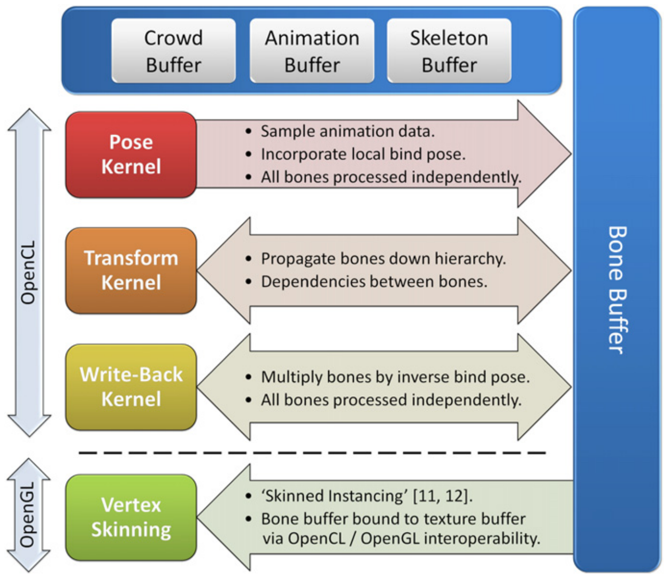
Crowd Buffer
Animation Buffer
Skeleton Buffer
Bone Buffer
OpenCL
Pose Kernel
• Sample animation data.
• Incorporate local bind pose.
• All bones processed independently.
Transform Kernel
• Propagate bones down hierarchy.
• Dependencies between bones.
Write-Back Kernel
• Multiply bones by inverse bind pose.
• All bones processed independently.
OpenGL
Vertex Skinning
• 'Skinned Instancing' [11, 12].
• Bone buffer bound to texture buffer via OpenCL / OpenGL interoperability.

Fig. 1. Skeletal animation is modelled by a three stage pipeline of OpenCL kernels; *Pose*, *Transform* and *Write-Back*.

(on-chip) to an OpenGL context and are then used by a vertex shader to skin the corresponding meshes.

3.1. Skeleton data

The pipeline makes use of two main data-structures; a skeleton buffer and a bone buffer. The skeleton buffer stores a set of template skeletons that can be shared by all members of a crowd. Whereas skeletons are often modelled by a tree-based hierarchy of bone nodes it is known that this will impede the flow of data due to non-contiguous memory accesses and explicit hierarchy traversals [21]. We perform a level-order flattening of the hierarchy and organize data in structure-of-arrays form (Fig. 2).

When a crowd member is spawned they are assigned a skeleton from the skeleton buffer. All skeletons are considered to be generic in the sense that they describe only constant attributes. To store data specific to each crowd member, namely their current pose, we define a bone buffer. Kernels can read bones from the bone buffer, process them and write back to the bone buffer in place of the original data. We store the bone buffer in global memory and take care to avoid strided and misaligned memory access patterns (non-coalesced) by concatenating data as shown in Fig. 3. Each bone is represented by a translation, rotation and scale component but interleaving them in this way implies a strided access pattern. We therefore separate the components and store them contiguously in memory. A further concern relates to how bones are packed. Data must reside on a boundary 16 times the size of the type being accessed to exploit coalescing [22] and this implies 256-byte alignment. Fig. 3 depicts a skeleton with 13 bones as an example. The first crowd member stores translation components from location 0 onwards, leaves a 3 bone padding and begins storing rotation components at location 16. Further padding surrounds the scale components and a second crowd member can then be appended.

3.2. Pose kernel

Having presented the underlying data structures we now introduce our first kernel. The *Pose* kernel (listed in Fig. 4) loads a crowd member's skeleton and orients its bones in pose-space by sampling keyframe data from an animation set. Animation sets are discussed later in Section 3.3. To run the kernel a crowd must