从 SSE 到 OpenCL：多核 CPU 上骨骼动画并行算法对比研究

刘寿生，陈戈，马纯永，韩勇

（中国海洋大学信息科学与工程学院，青岛 266100）

摘要：拥有高精度蒙皮和复杂骨骼绑定关系的骨骼动画，渲染时存在很大的性能瓶颈。以往研究采用 GPU 加速动画，但高端 GPU 成本过高，而中低端 GPU 的通用计算性能有时不如高端 CPU。为了充分挖掘算法在多核 CPU 上的执行性能，*弥补中低端 GPU 通用计算性能的不足，提出了基于 OpenCL 的针对指令和线程的新兴集成并行方案，并与基于 SSE 结合 OpenMP 针对指令和线程的传统独立并行方案展开对比。*实验结果表明，在多数 CPU 和多种复杂度的数据上，基于 OpenCL 的新兴并行方案的性能明显高于基于 SSE 的传统并行方案，并且性能优势随着数据复杂度的增加而提升。

关键词：骨骼动画；并行计算；OpenCL；SSE

中图分类号：TP391.9　　文献标识码：A　　文章编号：1004-731X (2014)

From SSE to OpenCL: Comparison of Parallel Algorithms for Skeletal Animation on Multi-core CPUs

LIU Shou-sheng，CHEN Ge，MA Chun-yong，HAN Yong

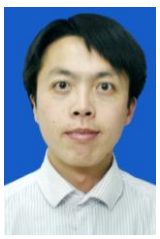
(College of Information Science and Engineering, Ocean University of China, Qingdao 266100, China)

Abstract: While skeletal animations consist of high fidelity mesh and complex vertex binding, there is a huge performance bottleneck. Previous studies using GPUs to accelerate animation, but the general purpose computing performance of cheap GPUs is lower than high end CPUs. In order to explore the performance for multi-core CPUs *to make up the shortfall of cheap GPUs for general purpose computing, the latest parallel scheme based on OpenCL which integrated the former two levels of parallelization is proposed, comparing to the traditional parallel scheme formed by SSE and OpenMP oriented to instruction parallelization and thread parallelization separately.* The experimental results show that on most CPU with different complex data the performance of the parallel scheme based on OpenCL is better than SSE. And the more complex the data is, the greater the performance advantage grows.

Key words: skeletal animation; parallel computing; OpenCL; SSE

引言

骨骼动画是一种形变模型，在学术上也被称为网格蒙皮、骨骼子空间形变、矩阵调色板蒙皮[1]。骨骼动画用于模拟虚拟人物或动物，广泛用于医学、影视和游戏等领域[2]。早期计算机动画基于关键帧顶点混合，每个关键帧需要一个独立的模型，所以一段动画需要多个模型，占用很大的内存和外存。



作者简介：刘寿生(1985-)，男，江西宁都人，汉族，博士生，研究方向为虚拟现实、并行计算、高性能；陈戈(1965-)，男，浙江宁波人，汉族，博士，教授，博导，研究方向为海洋遥感、地理信息系统、虚拟现实；马纯永(1984-)，男，山东潍坊人，汉族，博士，讲师，研究方向为虚拟现实；韩勇(1969-)，男，陕西商南人，汉族，博士，教授，研究方向为虚拟地理环境、海洋地理信息系统；

收稿日期：2014-01-22　　修回日期：2014-04-30

基金项目：国家海洋局数字海洋科学技术重点实验室开放基金（KLDO201303）

为了压缩动画数据量，Burtnyk[3]于 1976 年首次提出骨骼动画的概念，骨骼动画只需一个模型，运动效果由骨骼的运动驱动。相比顶点混合动画，骨骼动画是一种以时间换空间的方法，节省空间的代价是大幅增添计算负担，造成性能瓶颈问题。

为了解决骨骼动画的性能问题，国内外学者普遍把研究焦点放在并行加速方法上面。世界上一些著名研究机构和大学建立了专门的并行计算研究小组：1）斯坦福大学的普适并行实验室(The Pervasive Parallelism Laboratory, http://ppl.stanford.edu/, Stanford University)[4, 5]。该实验室采用应用驱动的方法使并行计算研究范围从编程模型拓展到硬件体系，核心的技术概念是特定域语言，该语言能够提升编程效率，并集成了一个可以给并发和局部化管理提供动态或静态方案的通用并行执行环境。该实验室已经为多个领域提供了特定域的并行抽象语言，包括人工智能和机器人、商业数据分析、虚拟世界和游戏娱乐。2）佐治亚理工学院的 Cell 宽带引擎(Cell Broadband Engine, http://sti.cc.gatech.edu/, Georgia Institute of Technology)[6]。该引擎的研究得到索尼、东芝、IBM 三家厂商的支持，并用于驱动它们的工作站服务器等大型并行计算设备。它们主要的研究目标是 1 台附带了 8 台协处理器的多核处理器，应用领域涉及图像压缩和金融模型构建。

并行技术在软硬件两方面的发展都非常迅速，特别在 GPU 并行方面，并行技术先后经历了 Program 汇编、Shader 着色语言、CUDA 三代更迭。Lindholm[7]于 2001 年提出基于汇编语言的骨骼动画 GPU 并行算法。季卓尔于 2008 年提出基于 Shader 的 GPU 骨骼动画[8]。胡前亮于 2011 年提出基于 CUDA 的骨骼动画实时阴影仿真方法[9]。GPU 并行突破了骨骼动画的性能瓶颈以后，衍生出骨骼自动生成[10]、骨骼与皮肤自动绑定[11]、皮肤切割变换[12]等研究热点。

虽然高端 GPU 可以获取较好的加速效果，但高端 GPU 代价昂贵，而某些中低档 GPU 的通用计算性能不如高档 CPU。相比骨骼动画的 GPU 并行研究渐渐成熟，反观骨骼动画的 CPU 并行，国内外缺少学术上的研究。研究 CPU 上的并行方法，一方面可以作为 GPU 并行方法在软硬件无法支持时的备用方法，另一方面有助于探寻 CPU 最高性能从而为 GPU 加速提供评价基准[13]。工程应用上著名引擎 Ogre 针对骨骼动画于 2006 年提出了流式单指令多数据扩展(Streaming SIMD Extensions, SSE)指令并行方法。

SSE 系列指令集是 Intel 设计的支持 128 位操作数的指令集。SSE 指令并行技术的出现，主要是因为现代 CPU 拥有多个长度为 128 位的寄存器单元，支持一个指令周期同时操作 128 位数据，即 8 个 float 或 4 个 double，1 条加指令可以同时作用于 8 对 float 加数和被加数。CPU 指令集提供相应的数据类型和指令，支持同时操作多个操作数，可以成倍提升浮点运算的执行效率[14]。

开放多线程处理(Open Multi-Processing, OpenMP)是一个开放的多线程方法库[15]，每个 CPU 常常含有 4 个甚至更多的核心处理器单元，开发多线程并行程序理论上将获得跟核心数目一样多的加速比。在不同的 CPU 硬件厂商型号之间，以及不同的编译器软件之间，不同的操作系统之间，都具有良好的代码可移植性和性能可移植性。

随着开放并行计算语言(Open Computing Language, OpenCL)的提出，主流 CPU 厂商 AMD 和 Intel 先后于 2009 年和 2010 年在多核 CPU 上支持 OpenCL 技术[16]，从而为 CPU 并行技术的研究开辟了新的领域。OpenCL 是一个标准的规范框架[17]，根据该规范编写的并行程序，可以在多种异构处理器设备端上执行，包括 CPU、GPU、DSP、FPGA 和其它处理器[16]。各种主流处理器发布了各自的软件编程工具和硬件可编程能力，例如 Intel 的 Intel SDK for OpenCL Application、AMD 的 APP SDK、NVIDIA 的 CUDA，使得基于 OpenCL 规范编写的并行程序可以在 Intel、AMD、NVIDIA 的 CPU 和 GPU 上通用。OpenCL 基于 C 语言设计了

独特的编程语言，相比其它并行方案所用特有语言具有明显优势[18]。OpenCL 还包含一套规范 API 供主机端 host 调用，用于配置 OpenCL 运行环境并为设备端传递数据。

在并行性能评价理论方面，Amdahl 提出固定问题规模的并行加速比定律即阿姆达尔定律（Amdahl's Law）[19]，该定律定义加速比 *A* 的上限为 1/(1-*P*)，其中 *P* 为可并行模块的百分比，*S* 为并行模块的局部加速比，*A* 为全局加速比。阿姆达尔定律对应的加速比计算公式如下：

$$A = \frac{1}{1-P+\frac{P}{S}} \tag{1}$$

假设局部加速比 *S* 趋向∞，理论峰值加速比公式如下：

$$A_{max} = \frac{1}{1-P} \tag{2}$$

在因为代价昂贵等原因缺乏高端 GPU 的情况下，在处理大数据通用问题时，存在中低端 GPU 通用并行计算性能不足的问题。本文探索从中低端 GPU 并行回归 CPU 并行的方案，尝试用 CPU 并行弥补中低端 GPU 并行性能不足的问题。本文研究基于 OpenCL 的骨骼动画 CPU 并行方法，并与 SSE 叠加 OpenMP 的传统并行方法展开对比，从中选取较优方案。

1. 骨骼动画串行算法

1.1. 算法描述

骨骼动画渲染过程可以划分为以下四个环节，每个环节作为一个独立的子模块，四个环节分别是：1)从文件解析动画数据主要包括骨骼关节矩阵和顶点坐标；2)更新关节矩阵；3）更新顶点坐标；4）以三维面片的方式输出顶点坐标最终形成三维动画。

第三环节更新顶点坐标，占用大部分的完整渲染时间。该环节是骨骼动画的性能瓶颈，本文重点关注这个环节。本文采用矩阵调色板这一顶点更新算法，骨骼动画顶点更新算法如下：

Step 1. 动画模型具有 *T* 个三角面片，每个三角面片以 3 个索引值代表，索引值指向某个顶点；

Step 2. 获取 1 个三角面片；

Step 3. 获取 1 个顶点索引，通过该索引获取顶点信息，包括 1 个坐标(*x,y,z*)和 *B* 个骨骼索引及对应的权重索引；

Step 4. 对每个顶点调用矩阵调色板算法；

Step 5. 返回步骤 3，重复步骤 3~4，直到遍历完一个面片的 3 个顶点；

Step 6. 返回步骤 2，重复步骤 2~5，直到遍历完所有面片。

矩阵调色板算法描述如下：

Step 1. 读入 1 个顶点(*x,y,z*)，读入 *M* 个骨骼关节构成变换矩阵列表 mL，每个顶点绑定 *B* 个关节，索引和权重分别是 *d(i)*，*w(i)*，*i*=1,2…*B*。

Step 2. 为每个顶点求取融合矩阵 *m*，*m* 是 4*4 的矩阵，其中包含对顶点的旋转平移缩放信息。*m* 由顶点绑定多个骨骼的多个矩阵加权求和得到，计算公式如下：

$$\boldsymbol{m} = \sum_{i=0}^{B}(mL[\, d(i)\,] * w(i)) \tag{3}$$

Step 3. 按照以下变换公式为每个顶点进行坐标变换。

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m00 & m01 & m02 & 0 \\ m10 & m11 & m12 & 0 \\ m20 & m21 & m22 & 0 \\ m30 & m31 & m32 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} x*m00 + y*m10 + z*m20 + m30 \\ x*m01 + y*m11 + z*m21 + m31 \\ x*m02 + y*m12 + z*m22 + m32 \\ 1 \end{bmatrix} \tag{4}$$

1.2. 串行实现与瓶颈定位

骨骼动画顶点更新模块以及内嵌的矩阵调色板算法的串行版本流程图如图 1 所示。

问题规模相关的数据复杂度不一样时，各个子模块的时间分布会有所不同，瓶颈的位置和大小也会不一样。本文采用骨骼动画数据的顶点数目为 0.1M，即百万级别；单个顶点绑定骨骼的数目设定为 2；采用的 CPU 是 Intel i7 3770k。

通过测量算法各个模块的时间分布可以看到，耗时最大的模块是 CPU 计算顶点这一模块，比例

是 84%，说明它是整个算法的性能瓶颈。另外 CPU 计算骨骼虽然算法比较复杂但是所占时间比例极小，没有优化的必要。在进行瓶颈定位和并行优化之前，还需要对算法进行串行优化，使它更加符合并行运算的特点。并行运算特别适用于以下类型的算法：结构方面，适于顺序语句不适于循环嵌套以及分支；索引方面，适于相邻索引是连续地址的情形，不适于随机地址索引；函数调用方面，适于少量或没有函数调用，不适于频繁或嵌套调用函数。根据以上原则，从矩阵变换模块算法可以看到，算法中存在多重嵌套循环和多处随机索引，其中多重循环嵌套是面片循环和顶点循环以及更深层的骨骼循环，随机索引包括顶点随机索引和骨骼随机索引。对算法进行初步改进，使得串行代码本身获取最优性能，在此基础上进行后文的并行优化。串行优化后，CPU 计算顶点所占时间比例是 81%，仍然是整个算法的性能瓶颈。根据阿姆达尔定律公式，可以初步估算骨骼动画并行优化之后的理论峰值加速比。CPU 计算顶点这一性能瓶颈占全部时间的比例约为 P=80%，剩余部分只占 1-P=20%。根据公式 2 得出理论峰值加速比为 5。

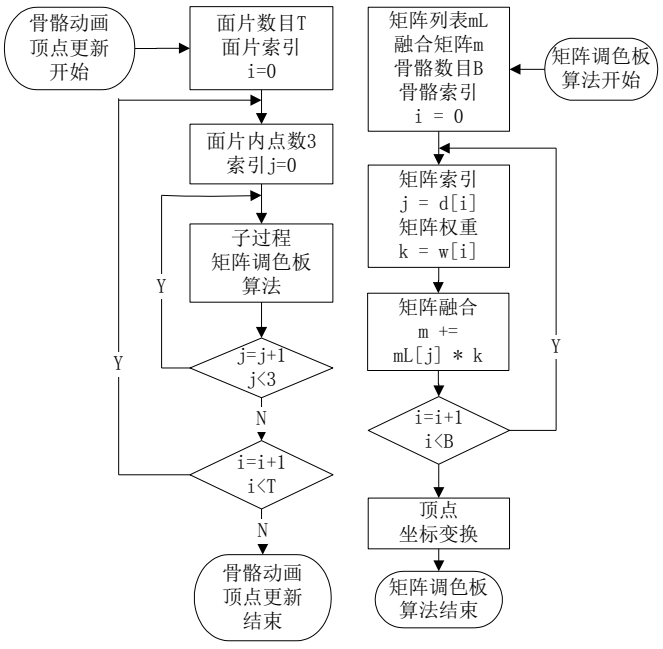


图 1 骨骼动画顶点更新和矩阵调色板算法流程图

2. 传统 SSE 并行

2.1. SSE 指令并行

SSE 指令早在 1999 年的 Pentium 3 与 2000 年的 Pentium 4 处理器中已经被采用，所以说现在的绝大多数处理器都支持 SSE 指令集。为了支持 SSE 操作并发挥最优性能，SSE 所访问的内存必须是按照 16 字节对齐，骨骼动画的顶点及矩阵对应的指针通过_aligned_malloc/_aligned_free 进行堆内存的动态分配和释放，对于静态数组，需要用到指示符" __declspec(align(16))"将数组显示声明 16 字节对齐。16 字节对齐并不是强制的，SSE 提供另一套支持非对齐的内存读写指令，只是性能不如对齐方式（下降 10%）。

本文采用 SSE 指令并行技术，提出了一种基于 SSE 指令并行的矩阵调色板算法，使得多条指令可以同时执行，节省指令执行时间。从 C/C++ 串行指令转变为 SSE 并行指令，需要做以下 2 方面的移植：

第一处移植，矩阵按权重累加改为：

// __m128 pMatOne 代表单个矩阵，pMatLast 代表累加结果

```
for (int j=0;j<4;j++)
{
  pMatLast[j] = _mm_add_ps(pMatLast[j], _mm_mul_ps( pMatOne[j], weight) );
}
```

第二处移植，坐标矩阵变换改为：

Step 1. 输入坐标 pIn 构造 __m128 xxxx, y, z

```
__m128 xxxx =_mm_shuffle_ps( pIn, pIn, _MM_SHUFFLE(0,0,0,0) ); //y, z 同理
```

Step 2. 矩阵构造__m128 m[4];

Step 3.计算输出坐标：

pOut=m[0]*xxxx+m[1]*yyyy+m[2]*zzzz+ m[3]

因为 SSE 不支持多项式连续相加，所以拆成多步。其中，__m128 乘法调用_mm_mul_ps，__m128 加法调用_mm_add_ps。

2.2. OpenMP 线程并行

若要发挥 OpenMP 多线程的并行加速能力，算法应符合以下特征。首先，线程之间很少通信或没有通信，避免线程依赖，一个线程等待另一线程的运行结果；其次，线程之间数据共享很少或没有共享，避免因资源竞争导致线程处于等待状态耗费时间，尽量通过 private 子句将多线程外部引入的全局变量私有化；第三，相邻线程的数据访问地址最好连续或相近，充分发挥 cache 缓存数据的预读和重用效率。本文研究的骨骼动画矩阵调色板算法，相邻面片和顶点之间没有通信和数据共享，而且数据存储结构紧凑，数据结构和算法都符合 OpenMP 的并行要求。在 Windows 7 操作系统下通过 VC 语言调用 OpenMP 实现算法的多线程转化，转化后的多线程算法可以很方便地移植和重用。调用 OpenMP 无需额外的库，只需在编译时添加编译选项"/openmp"，或在 VC 工程属性中开启"OpenMP Support"设置，该设置位于 C/C++语言 Language 页面，将"OpenMP Support"由默认的"No"改为"Yes"。接着在需要执行多线程的文件头引入”omp.h”，接着就可以调用 OpenMP 提供的 API 方法和#pragma omp 系列宏指令，这些方法包括修改线程数目、获取线程 id 等等，#pragma omp 指令控制多线程的启停。OpenMP 最常用的一个场合是并行执行循环，将循环体拆分到多个线程同步并行执行，实现这种多线程，只需要调用一条#pragma omp 指令，即在 for 语句前一行调用"#pragma omp parallel for "。

本文对骨骼动画矩阵调色板执行多线程算法改进，开启 OpenMP 设置，在 for 循环调用 omp parallel 宏，关键改动在于：在遍历每个顶点的循环体前面，使用宏指令”#pragma omp parallel for”。OpenMP 可以单独使用作为线程并行方案，也可以叠加之前的 SSE 指令并行形成组合方案。

3. 新兴 OpenCL 并行

面向 CPU 的 OpenCL 并行方案，具体底层的实现隐式调用传统的多线程并行及多指令并行，但编程人员只需按 OpenCL 规范编写程序，无需手动调用之前的多线程（例如 OpenMP）以及多指令（例如 SSE）。OpenCL 跟之前用到的 OpenMP 和 SSE 并行技术在性能上普遍存在差异[15]，另外之前技术不用另外安装第三方库，编译环境内置支持，只要开启编译选项并包含头文件，在代码中就可以调用 OpenMP 和 SSE 的功能 API。而 OpenCL 必须安装设备驱动和软件开发包 SDK，本文选用 Intel(R) SDK for OpenCL Applications 2013。因为 OpenCL 是开放标准，同一台系统里面可以安装多套 OpenCL 驱动和 SDK，比如 Intel OpenCL、AMD OpenCL、NVIDIA OpenCL。clGetPlatformIDs 可以获取已经安装 OpenCL 的数目以及详细信息，当数目为 0 时，说明未安装 OpenCL；当数目不为 0 时，按照需要选用特定的某套 OpenCL，比如本文选用"Intel(R) OpenCL"。

调用 OpenCL 之前，需要初始化 OpenCL 运行环境。初始化对于不同工程差别不大，主要选定 OpenCL 驱动类型、选定 CPU 或是 GPU、指定 cl 文件及 kernel 函数名。初始化完成后，接下来设计 kernel 函数，设计过程可以参照 SSE 算法版本。

3.1. 常规 OpenCL 并行

先实现单骨骼情况，即每一个顶点只关联 1 块骨骼，坐标变换时只有一次矩阵变换。OpenCL 的 kernel 函数实现如下：

Step 1. 获取 OpenCL 线程 ID，当做顶点索引，线程 ID 获取公式如下：

$$id = get\_global\_id(0) + get\_global\_id(1) * get\_global\_size(0) \quad (6)$$

Step 2. 索引输入顶点 pIn 和矩阵 pMatrix，其中每个矩阵由 4 个 flaot4 组成；

Step 3. float4 数据结构封装，将 pIn (x,y,z,1) 拆分为 float4 (x,x,x,x) 、 (y,y,y,y) 、 (z,z,z,z)

Step 4. 顶点矩阵变换结果输出，float4 多项式相乘累加，计算方法如下：

pOut [index] = xxxx * pMatrix[offset+0] + yyyy * pMatrix[offset+1] + zzzz * pMatrix[offset+2] + pMatrix[offset+3]

OpenCL 与 SSE 对比主要有如下两个不同点：

1）数据类型不同。SSE 的__m128，OpenCL 对应 float4；SSE 给__m128 赋值时采用 _mm_load_ps 指令，OpenCL 给 float4 赋值时采用 (float4)(x,y,z,w)；

2）运算符不同。SSE 因为不支持加法运算符所以需要通过指令实现，而 OpenCL 直接支持多项式相乘累加，OpenCL 算法非常简洁，拥有很高的可读性和可维护性。

对于每个顶点关联多个骨骼的情况，在获取变换矩阵时，需要索引多个矩阵，按照权重累加。所以在 kernel 参数列表中需要添加矩阵索引和权重，在 kernel 函数体中仿照 SSE 按权重累加矩阵即可。

OpenCL 执行之前，除了指定供 OpenCL 设备调用的指令代码以外，还需指定工作线程结构 workgroup/workitem，包括全局 Global WorkSize 和局部 Local WorkSize。它们都支持 2 到 3 维，一般只设置 GlobalWorkSize，设置时每个维度支持最大数目不同，通过 clGetKernelWorkGroupInfo 可以动态检测到这个值，各个维度乘积大于线程总需求量即可，本文采用二维线程结构，第一维设置为最大值，第二维用总顶点数除于最大值；而 Local WorkSize 采用系统默认设置。kernel 并行指令、数据参数以及并行参数设好后，在主机端 host 调用 clEnqueueNDRangeKernel 将 kernel 编排的指令任务发布到 OpenCL 并行设备端。该指令调用属于异步调用，即调用后主机端所在 CPU 线程不等待设备处理结果，CPU 接着执行下一句指令。在测量时间或者需要把处理结果返回内存时，需要等到设备端的指令彻底执行完成，可以调用 clWaitForEvents 阻塞主机端线程使之进入空闲等待状态。

3.2. 与 OpenGL 互操作

为了优化骨骼动画渲染和数据传输的效率，本文采用 OpenCL 与 OpenGL 的互操作特性(Interoperability)，共享顶点缓存对象(Vertex Buffer Object, VBO)等存储对象(Memory Object)。先检测当前环境是否支持 CL-GL 互操作，检测步骤如下：1）因为互操作性属于 OpenCL 1.2 扩展，并不是 OpenCL 核心功能，所以需要动态获取 clGetGL-ContextInfoKHR 功能函数的实际地址，如果获取成功，说明支持 CL-GL 互操作；2）通过 OpenCL 扩展接口 clGetGLContextInfoKHR 获取某个特定的 OpenCL 设备 cl_device_id，该设备支持互操作；3）最后通过 clCreateContext 创建 OpenCL 上下文 cl_context。

检测并开启 OpenCL-OpenGL 互操作以后，需要修改常规 OpenCL 算法以实现互操作。互操作主要体现在内存空间上面，即 OpenCL 设备内存 cl_mem 与 OpenGL 设备内存 Buffer Object。为了支持内存共享以及互操作，在 OpenGL 和 OpenCL 两端都需要做相应的调整。在 OpenGL 端，需要改变 OpenGL 渲染数据的存储方式。原先是立即存储方式，即存在主机端内存，每次渲染时发送到 OpenGL 设备；现在要改为 Vertex Buffer Object(VBO)方式，数据一直存在 OpenGL 设备，而且可以通过 VBO 的 Buffer 资源 ID 进行读写访问。本文将骨骼动画顶点数据存储在 VBO 当中，创建 VBO 的同时通过 clCreateFromGLBuffer 映射一块 OpenCL 内存 cl _mem，VBO 和 cl _mem 共享顶点数据。当顶点需要通过 OpenCL 更新时，通过 clEnqueueAcquireGL -Objects 获取数据，修改完成后通过 clEnqueueReleaseGL -Objects 返还给 VBO。

通过在 OpenGL 和 OpenCL 分别进行以上改造，从 2 个方面对算法性能进行了改进。一方面，使得 OpenGL 和 OpenCL 之间可以共享数据，即实现了 OpenGL 的 VBO 与 OpenCL 的 cl_mem 之间的双向转化，从而免去主机端参与中转，可以节省数据传输时间；另一方面，因为采用 VBO 渲染方式，所以提升了渲染性能。

4. 动画仿真实验

本文实验数据即骨骼动画数据的规格如下：顶点数目为 10 万；骨骼关节点的数目为 77；单个顶点绑定骨骼的数目设定为 2。实验平台配置如下：CPU——Intel i7 3770K，GPU——NVIDIA Geforce GTX 670，编程环境——Visual C++ 2010。骨骼动画渲染效果如图 2。

图 2 小海龟角色动画渲染效果

4.1. 并行方案对照

本文在骨骼动画矩阵调色板串行算法的基础上，面向 CPU 设计了 5 套互为对照的并行版本，其中 3 套基于传统并行技术，2 套基于新兴并行技术。前 3 套分别是 OpenMP 多线程并行、SSE 多指令并行、SSE 与 OpenMP 叠加并行，后 2 套分别是常规 OpenCL 并行、与 OpenGL 互操作的 OpenCL 并行。

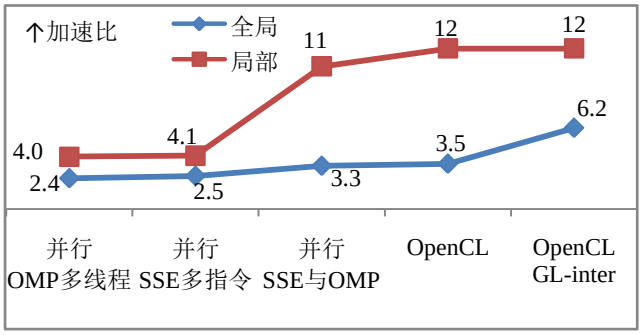


图 3 骨骼动画对比 5 组并行算法的加速比

如图 3 所示，5 套并行算法在选定的数据和平台上运行，统计得到 5 组并行加速比。从以上实验结果图得知：1）与 OpenGL 互操作的 OpenCL 版本(OpenCL GL-inter)，算法性能最优，选定为最终的 OpenCL 方案；2）SSE 叠加 OpenMP 并行版本，在 3 种传统并行方案中算法性能较优，选定为传统并行方案；3）新的 OpenCL 并行方案与传统并行方案相比，加速比从 3.3 提升到 6.2，增幅达 88%。

骨骼动画算法 OpenCL 性能将近达到 SSE 的 2 倍，也付出了一些额外的代价。除了性能，本文对 OpenCL 和 SSE 的内存占用量、CPU 占用率、GPU 占用率以及渲染画质等其它 4 方面做了全面的对比观测，结果如表 1：

表 1 骨骼动画并行算法 OpenCL 和 SSE 的全面对比

	加速比	内存/点	CPU %	渲染画质
单线程	1	68 Bytes	25%	无变形
SSE	3.3	68 Bytes	100%	同上
OpenCL	6.2	100 Bytes	30%	同上

通过全面对比观测得知，OpenCL 相比 SSE 叠加 OpenMP，以 1.5 倍内存换取 2 倍性能，并且 CPU 占用率下降 70%。同时 GPU 占用率和渲染画质基本不变，画面没有变形。由此可见，研究人员可以放心地采用 OpenCL 挖掘 CPU 的并行能力，代价小且不会引进副作用或新的隐患。

4.2. 数据复杂度对比

数据复杂度由以下 2 个参数决定，包括单个模型的顶点数目、单个模型内每个顶点绑定的骨骼数目。本文为 2 个参数分别设定 4 个数量级，顶点数范围为 25k~1600k、单个顶点绑定的骨骼数为 1 至 4，交织在一起组成 4 乘 4 矩阵共 16 组不同复杂度的数据。测量在 16 组数据下的 OpenCL 加速比和 SSE 加速比，并统计 OpenCL 相对 SSE 加速比的增幅。

从表 2 和表 3 可以看出，并行算法 OpenCL 对比 SSE 有明显的性能优势，OpenCL 加速比在 1.9~9.1 之间，而 SSE 加速比在 1.1~4.0 之间。

表 2 骨骼动画对比 16 组数据的 OpenCL 并行加速比

顶点 / 骨骼	25k	100k	400k	1600k
1	1.9	1.9	3.4	2.7
2	3.4	6.2	7.9	7.6
3	3.9	7.2	8.0	8.3
4	4.5	7.7	8.7	9.1

表 3 骨骼动画对比 16 组数据的 SSE 并行加速比

顶点 / 骨骼	25k	100k	400k	1600k
1	1.4	1.2	1.2	1.1
2	2.4	3.3	3.0	3.1
3	2.4	3.6	3.4	3.4
4	2.6	4.2	4.0	4.0

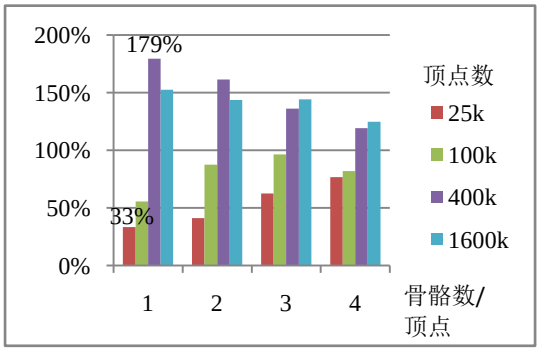


图 4 骨骼动画对比 16 组数据的 OpenCL 相对 SSE 加速比增幅

从图 4 得知 OpenCL 相对 SSE 的加速比增幅在 33%~179%之间，平均增幅 106%，即 OpenCL 是 SSE 的 2 倍。并且性能优势随模型复杂度的提升而稳定线性增长。数据量越大，越能发挥 OpenCL 的并行加速优势。

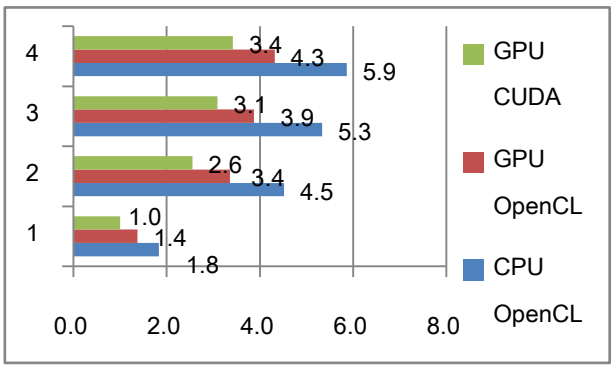


图 5 高端 CPU 与低端 GPU 并行加速比的对比图

4.3. 高端 CPU 对比低端 GPU

本文选定一款高端 CPU（Intel i7 870）和一款低端 GPU（NVIDIA Geforce GTS 250）进行对比试验。在 CPU 上采用 OpenCL 并行，在 GPU 上采用 OpenCL 和 CUDA 两套并行。实验结果如图 5，在骨骼动画加速这个案例上，GPU 的加速性能总是不如 CPU。主要原因是 GPU 计算受限于内存与显存之间的数据传输带宽，数据传输时间开销抵消了处理器加速效果，而 CPU 不需要这份开销，所以高端 CPU 的并行性能可以超过中低端 GPU。

5. 结论

在因为代价昂贵等原因缺乏高端 GPU 的情况下，CPU 并行方案可以顶替中低端 GPU，从而解决中低端 GPU 通用并行计算性能不足的问题。本文为骨骼动画面向 CPU 设计了基于 OpenCL 的新兴并行方案，并与基于 SSE 的传统并行方案展开对比，寻求较优方案。实验结果表明，对于同等数据量，OpenCL 并行算法的加速比是 SSE 的 2 倍，并且性能优势随数据复杂度的增加而呈稳定线性增长趋势。本文工作对并行算法的研究具有重要的理论意义和应用价值。

参考文献：

[1] Kavan L. Real-time Skeletal Animation[D]. Prague: Czech Technical University, 2007.

[2] 赵维，谢晓方. 虚拟人技术发展现状及其在工程中的应用[J]. 系统仿真学报, 2009, 21(17): 5473-5476. (ZHAO Wei, XIE Xiao-fang. Development of Virtual Human Technology and Its Engineering Application[J]. Journal of System Simulation (S1004-731X), 2009, 21(17): 5473-5476.)

[3] Burtnyk N, Wein M. Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation[J]. Communications of the ACM(S0001-0782), 1976, 19(10): 564-569.

[4] Chafi H, Sujeeth A K, Brown K J, et al. A Domain-specific Approach to Heterogeneous Parallelism[C]// Proceedings of the 16th ACM symposium on Principles and practice of parallel programming. ACM, 2011.35-46.

[5] Brown K J, Sujeeth A K, Lee H J, et al. A Heterogeneous Parallel Framework for Domain-specific Languages[C]// Parallel Architectures and Compilation Techniques (PACT). 2011.89-100.

[6] Ismail L, Guerchi D. Performance Evaluation of Convolution on the Cell Broadband Engine Processor[J]. IEEE Transactions on Parallel and Distributed Systems(S1045-9219), 2011, 22(2): 337-351.

[7] Lindholm E, Kilgard M J, Moreton H. A User-programmable Vertex Engine[C]// Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, 2001.149-158

[8] 季卓尔，张景峤. 基于可编程 GPU 的骨骼动画[J]. 计算机工程与应用, 2008, 44(22): 77-80.

[9] 胡前亮，陈炳发. 一种采用 CUDA 的骨骼动画阴影实时仿真方法[J]. 小型微型计算机系统, 2011, 32(1).

[10] 郝爱民，赵永涛，吴伟和，等. 任意姿态虚拟人网格模型骨骼提取算法[J]. 中国图象图形学报, 2011, 16(6): 1008-1014.

[11] 刘登志. 人体角色的自动绑定与卡通运动[D]. 浙江大学, 2011.
[12] YAN H B, HU S M, Martin R R, et al. Shape Deformation Using a Skeleton to Drive Simplex Transformations[J]. IEEE Transactions on Visualization and Computer Graphics(S1077-2626), 2008, 14(3): 693-706.
[13] Lee V W, Kim C, Chhugani J, et al. Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU[C]// ACM SIGARCH Computer Architecture News. ACM, 2010.451-460.
[14] Shi G, Li M, Lipasti M. Accelerating Search and Recognition Workloads with SSE 4.2 String and Text Processing Instructions[C]// Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on. IEEE, 2011.145-153.
[15] Shen J, Fang J, Sips H, et al. Performance Gaps between OpenMP and OpenCL for Multi-core CPUs[C]// Parallel Processing Workshops (ICPPW), 2012 41st International Conference on. Pittsburgh, PA: 2012.116-125.
[16] Pennycook S J, Hammond S D, Wright S A, et al. An Investigation of the Performance Portability of OpenCL[J]. Journal of Parallel and Distributed Computing(S0743-7315), 2012: 1-12.
[17] Khronos. The OpenCL Specification[EB/OL].(2012)[2012-11-14]. http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf.
[18] Du P, Weber R, Luszczek P, et al. From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming[J]. Parallel Computing(S0167-8191), 2012, 38(8): 391-407.
[19] Sun X H, Chen Y. Reevaluating Amdahl’s Law in the Multicore Era[J]. Journal of Parallel and Distributed Computing(S0743-7315), 2010, 70(2): 183-188.