实验记录

# 实验背景

## 技术背景

### 骨骼动画

骨骼动画是一种计算机三维动画，主要用于模拟拥有骨骼和关节的人和动物。骨骼动画的组成是：三维静态模型，骨骼及骨骼关键帧动画，静态模型顶点与骨骼之间特定的对应关系。渲染骨骼动画时，首先骨骼发生改变，从而带动模型顶点改变，最终形成动态的三维模型，模拟人或动物的动作。

### CUDA

CUDA全称是Compute Unified Device Architecture统一计算设备架构，它是由NVIDIA推出的通用并行计算架构，该架构下的GPU兼有通用计算和图形处理两大功能，GPU能够顶替CPU的多数计算任务。CUDA是GPGPU产品的一个新的基础架构，一个完整的GPGPU解决方案，它采用C语言作为编程语言提供大量的高性能计算指令开发能力，使开发者能够在GPU的强大计算能力的基础上建立起一种效率更高的密集数据计算解决方案；它提供了硬件的直接访问接口，而不必像传统方式一样必须依赖三维图形API接口来实现GPU的访问，从而编程人员在完全不接触图形编程的情况下，能够像使用CPU一样透明地使用GPU进行与图形无关的通用计算。

### VR

VR全称是Virtual Reality虚拟现实，是用于模拟现实世界的一整套计算机技术的集合。现实世界是一个不断运动的世界，特别是人类和其它动物。虚拟现实的服务对象是人类本身，所以人类静态外观和动态行为的模拟是虚拟现实技术的很重要的组成部分，特别是动态行为的模拟。

人类行为的模拟涉及到计算机三维动画技术，骨骼动画可以用来模拟人的各种动作。骨骼动画用骨骼点的位置偏移和旋转角度，控制全身皮肤的运动，以很少的数据量记录模型的动作。在动画模拟过程中，需要通过骨骼更新模型坐标，人体的模型外观主要是皮肤，皮肤是不规则曲面，需要非常大的数据量进行模拟。所以，在骨骼更新模型坐标这个环节，涉及密集数据的密集计算。传统的骨骼动画计算和渲染，需要耗费大量的时间，极大影响整个虚拟现实场景的渲染效率。

结合VR使用骨骼动画过程中遇到的密集数据的密集计算问题，以及CUDA技术在解决该类问题的潜在优势，本文拟用CUDA技术，加速骨骼动画的计算和渲染。

## 项目背景

CUDA加速VRGIS平台的骨骼动画模块。VRGIS平台是一个集VR仿真与GIS分析一体的拥有自主知识产权的国内软件平台，其中包含动态三维模型仿真模块，动态三维模型以骨骼动画实现。VRGIS支持骨骼动画的具体格式是Milkshape的ms3d格式和Autodesk的fbx格式。当前动态三维模型的顶点更新和渲染，没有采用GPU加速，时间效率相当低，使用CUDA在GPU内部实现顶点更新，再搭配VBO实现渲染加速，将是一个提升动画渲染效率的有效方案。

骨骼动画模拟空间站设备虚拟装配过程。虚拟装配模拟各种机械设备的组装与动态操作，机械设备的动作由类似人类的关节所控制，因此骨骼动画被用于实现虚拟装配。与传统的播放预先录制好的骨骼关键帧动画不同，虚拟装配用到的机械关节的动作是实时获取的，需要进行实时交互，具体获取方式是传统的鼠标键盘或者其他的输入设备。

Kinect动作捕捉与实时追踪。Kinect就是上文所谈到的其他输入设备的一种，它是微软所出产的三维体感输入设备，能够捕获真实世界中的人物动作，传递给计算机三维动画模型。使得计算机三维模型能够实时“学习”人类的动作。这里的三维动画模型，也可以用骨骼动画来具体实现，动作捕捉和实时渲染同样是高密度高计算量的操作，适于用CUDA进行加速。

## 实验目的

本文计划将传统的在CPU实现的骨骼动画算法移植到CUDA，从而拓展CUDA应用领域；同时，骨骼动画的高密度高计算量的操作通过CUDA实现加速，从而解决骨骼动画的时间效率问题；最终，总结研究一套以CUDA实现三维动画加速的一般方法。

# 实验基础

## 数据的选取

本文为了实现以上提到的实验目的，先后用到复杂度不同并且在现实应用场合具有广泛代表性的三种典型数据，分别是不含骨骼的静态模型、拥有少数骨骼的简易动画模型、拥有完整骨骼和光滑皮肤的复杂动画角色，数据格式采用Milkshape的MS3D格式。静态模型选用一个立体十字木架，用于测试静态模型渲染算法的可行性和时间效率；简易动画模型选用一个不断游动的海豚，用于测试动态模型骨骼顶点更新算法以及渲染算法的可行性；复杂动画角色选用一个三维动画短片《小海龟漫游海底世界》的主角——小海龟卡西，拥有68个骨骼关节和64K个顶点[v???]，用于测试用CUDA加速骨骼动画的可行性和有效性。

## 实验环境

### 硬件配置

处理器包括CPU和GPU，为CUDA加速前后构造对比，本文拟选用三组同等级别的CPU和GPU，分别为高、中、低三个档次的配置。对比实验时，高端CPU与高端GPU对比，中低端以此类推，从而获取客观和稳定的加速比。否则以高端GPU与低端CPU对比实验将导致加速比被高估；或者，以低端GPU与高端CPU对比实验将导致加速比被低估，以上两种情况，都将失去CUDA加速性能实测的普遍性和参考意义。当今，高端CPU典型配置是4核，高端GPU典型配置是是1024个流处理器；中端CPU典型配置是2核，中端GPU典型配置是256个流处理器；低端CPU典型配置是1核，低端GPU典型配置是64个流处理器。具体配置如下表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | CPU  档次 | 处理器核数 | 型号 | 发行日期 | GFLOPS  浮点运算能力 | 备注 |
| CPU1 | 高端 | 4 | [Intel Core i7 3770K](http://ark.intel.com/products/65523) | 2012年4月 | ？/64 |  |
| CPU 2 | 中高端 | 4 | [Intel Core i5 2300](http://ark.intel.com/products/52206) | 2011年1月 | 66 | 周圣川 |
| CPU 3 | 中端 | 2 | [Intel Core 2 Duo E7500](http://ark.intel.com/products/36503) | 2009年1月 | ??/23 | 江媛媛 |
| CPU 4 | 低端 | 2 | [Pentium Dual E2140](http://ark.intel.com/products/29738) | 2007年6月 | 5 | 马伟霞 |
| CPU 5 | 中高端 | 4 | Intel Core i7 870 |  | 36 | 508 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | GPU  档次 | 处理器核数 | 型号 | 发行日期 | GFLOPS  浮点运算能力 | 备注 |
| GPU1 | 高端 | 1344 | GeForce GTX 670 | 2012年5月 | 2460 |  |
| GPU2 | 中高端 | 384 | GeForce GTX 560 Ti | 2011年1月 | 1263 |  |
| GPU3 | 中端 | 192 | GeForce GTS 450 | 2011年3月 | 601 | 周圣川 |
| GPU4 | 低端 | 64 | GeForce 9600 GT | 2008年2月 | 312 | 马伟霞 |
| GPU5 | 中端 | 96 | GeForce GT 440 | 2011年2月 | 311 | 范晓晔 |
| GPU6 | 中端 | 128 | GeForce GTS 250 | 2009年3月 | 705? | 508 |

马伟霞 GPU4 VS CPU4

新机2 GPU1 VS CPU1

江媛媛 周圣川 换 GPU，江媛媛GPU3 VS CPU3

 周圣川 新机1 换 GPU，周圣川GPU2 VS CPU2

GPU5 vs CPU6 = 4.4 / 14.4

GPU5 vs CPU4 = 19.4 / 16.6

### 软件环境

本实验的软件环境如下，CUDA版本是 4.2，编程环境是Visual Studio 2010，CUDA调试环境是Nsight Visual Studio Edition 2.2 ，CUDA性能分析和优化工具是CUDA Visual Profiler和CUDA\_Occupancy\_Calculator。

# 实验设计

## 文件的解析和绘制

骨骼动画文件格式采用Milkshape的MS3D，骨骼动画文件的详细结构和解析方法参考MilkShape 3D SDK[11]以及论文[10]。本文着重关注骨骼动画的渲染，渲染通过OpenGL实现，渲染过程可以划分以下3个环节：

1. 根据关键帧插值更新骨骼的矩阵，简称骨骼更新
2. 用骨骼的矩阵更新模型顶点坐标，简称顶点更新
3. 模型渲染

当前的骨骼更新、顶点更新算法都是基于CPU进行实现；因为顶点每帧都需要在CPU中重新计算，所以渲染方式只能采用立即方式，而不能用VBO或者显示列表进行加速。无论是模型的更新还是渲染，都没有任何优化。

按照以上设想，本文先实现了骨骼动画模型的动态绘制，作为之后进行效率优化和比较的基准。本文先后渲染了3种类型的骨骼动画，静态的立体十字木架、动作简易的游行海豚、动作和模型都很复杂的动画角色小海龟卡西。效果如下图：

## 优化策略

实施优化之前，首先从理论推导和实际测量两个方面做时间效率分析，根据分析结果，获取优化目标。优化的评价标准，分为横向评价和纵向评价，横向评价用于评测GPU相对CPU的时间效率提升，纵向评价用于评测CPU或GPU各自进行内部优化的时间效率提升。

横向评价时，时间效率提升的具体计算方法如下：假设GPU耗时Tg，CPU耗时Tc，GPU时间效率Eg=1/Tg，CPU时间效率Ec=1/Tc，时间效率提升I = (Eg – Ec)/Ec = Eg/Ec – 1 = Tc/Tg – 1。纵向评价时，目的是分别获得最优的CPU和GPU算法，不涉及GPU与CPU之间的绝对比较，所以不以绝对的时间作为比较的标准，只计算各自所耗费的时钟周期数目。

本文的重点是时间效率的优化，因此时间的测量及其结果的输出，都应当最大程度地避免误差和干扰。横向评价时，在CPU主机端对同一功能模块的CPU算法和CUDA算法分别调用测时函数cutGetTimerValue，在测量时间之前调用同步函数cutilDeviceSynchronize等待CUDA操作执行完成。纵向评价时，clock。

监测方法，clock()系统函数测量时间[v133]，Windows程序调用控制台作为显示

优化过程如下：首先进行CPU优化，包括算法本身的优化和多核多线程并行，CPU多线程拟用OpenMP来实现；其次进行基于CUDA的GPU并行优化，在完成算法从C/C++到CUDA C的移植之后，采用一系列CUDA优化方法，使得CUDA时间效率达到最优。

## 优化前的准备

### CPU优化

通过分析耗时比重确定优化目标。骨骼更新、顶点更新、模型渲染各占3%、82%、4%。顶点更新占用时间比重较大，是重点优化对象。接下来，以顶点更新作为优化对象，用OpenMP进行CPU多核并行加速。使用OpenMP，只需要在循环次数最多的那个循环上面添加一条指令”#pragma omp parallel for”，同时开启编译选项”/openmp”。

为了达到最优的并行运算效率，需要对算法做一定的修改，尽量避免循环结构和分支结构，展开小循环、以三目运算替代小分支。顶点更新原始算法涉及多重循环，需要削减循环的层数，展开由三角面遍历三个顶点的小循环；顶点计算时需要判断是否绑定骨骼，存在分支结构，用三元运算符进行替换。

渲染方面，虽然占用时间很小，但是还是可以做优化，把渲染模式从立即改为交错数组。立即模式逐点向显卡传输顶点数据，交错数组一次性传输所以顶点数据，大大减少内存与显存之间传输数据的次数，从而提升渲染效率。渲染改为交错数组模式，也为以后改写CUDA渲染提供对比，CUDA渲染也是交错数组模式，只是交错数组本身以VBO的方式常驻显存，进一步减少内存与显存之间传输数据的次数。虽然动态模型不能采用VBO加速，但是不含动画的静态模型可以采用VBO，渲染加速比能够达到上百倍。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 优化前 | 优化后 | 提升幅度 |
| GPU1 |  |  |  |
| GPU2 |  |  |  |
| GPU3 |  |  |  |
| GPU4 |  |  |  |

### GPU代码移植

对于CPU优化目标，以耗时比例作为依据，上文已经确定是顶点更新模块。对于CUDA，除了耗时比例作为依据之外，数据量、计算量的双重密度也是一个依据，顶点更新模块存在高密度的数据量和计算量，因此CUDA优化目标也是顶点更新模块。因为用CUDA执行顶点更新是在GPU中进行，所以渲染方式自然采用VBO，从而间接优化了渲染模块。而骨骼更新模块，虽然耗时只占很小的比例，但是顶点更新必须依赖于骨骼更新的结果，所以需要进一步考虑是否对骨骼更新执行CUDA优化。如果骨骼更新仍旧在CPU执行，每帧需要从内存传送骨骼数据到显存，将花费一定的时间；反之，将骨骼数据一次性传输到显存，再用CUDA对骨骼更新模块执行通用计算和并行加速，节省了从内存传送骨骼数据到显存的时间，但是骨骼节点存在内部继承和依赖，导致并行线程局部关联，需要线程通信和同步，算法较为复杂，不适于并行运算。本文采用不对骨骼更新执行CUDA优化的方案。

确定CUDA优化目标后，接下来将CPU的C/C++代码移植为CUDA C。移植过程中，需要对算法做一些修改，比如：显存中局部变量必须初始化，否则运算结果可能出错；针对显存的操作，不支持内存函数memset等，需要重写显存初始化等函数；算法涉及的类必须改为结构体。移植完成后，功能很难保证正确，庆幸的是NVidia提供了调试工具，可以在调试过程中逐渐完善程序。本文采用的调试工具是Nsight Visual Studio Edition 2.2，该调试工具从2.2版本开始支持单显卡单机调试。

顶点更新模块从CPU的C/C++代码移植为CUDA C代码之后，为优化效果做一次横向评价，得到CUDA优化的初步效果，也为后文对CUDA进行内部优化提供纵向评价的参照。横向评价如下表：

根据表格得出横向评价结论，CUDA的时间效率优于CPU单核，但是不如CPU多核并行。这种结论，既是意料之外，也在预想之中，意料之外是让人对CUDA应许的加速能力有些失去信心甚至就此放弃；预想之中是CUDA技术本身是较新技术，它的加速应许不是那么轻易就能兑现，需要恳切地挖掘。接下来就是本文的核心部分，对CUDA算法进行一系列优化，最大程度挖掘CUDA的潜力，力求获得比CPU多核并行更高的性能。

GPU初始，没有任何优化，移植CPU代码，对CPU代码的基本要求是：算法简洁、性能最优。

数据结构：顶点float[4]，矩阵float[4][4]。

算法当前：顶点静态动态2份，矩阵1份。矩阵修改静态顶点，得到动态顶点。算法备选：顶点1份，矩阵新旧2份。新旧矩阵相除，修改顶点。前者算法简单少耗时间多占空间，后者数据节省少占空间多耗时间。暂时采用前者，最后以扼要结论的方式给出后者方案的结果。

CUDA配置：一个线程处理一个顶点。每个块256个线程，块数有顶点数目决定。

## CUDA优化

上文实现了用CUDA执行顶点更新的算法，但是时间效率介于CPU单核和CPU多核之间，没有体现GPU相对应CPU在密集数据密集运算场合的优势。究其原因，没有遵守CUDA使用规范和优化策略，是根本的原因。算法的移植只是一个基础，接下来将按照CUDA使用规范和优化策略，逐步优化CUDA算法，真正发挥CUDA在通用计算和并行加速双方面的优势。CUDA作为支持通用运算和并行加速的软硬件体系，继承了CPU多核并行的特点，同时又有自己的特色。之前在CPU端执行的优化，也作为CUDA优化的一部分，以下针对CUDA特有的属性展开进一步优化。

### 削减循环层数

CUDA为了最大化发挥并行运算的优势，弱化了对分支结构的支持，分支结构严重阻碍CUDA性能的发挥。循环结构内部存在循环条件的判断，隐含分支结构，所以也应该尽量减少循环结构。核函数内部原本存在通过三角面索引三个顶点的小循环以及判断顶点是否绑定骨骼的小分支，CPU算法改进环节已经执行优化。

骨骼动画原始算法将动画角色的身体模型划分为多个部位，用一层循环遍历各个身体部位分别执行顶点的更新，导致多次调用CUDA核函数，本文将全身的顶点融为1个数组存储在1个VBO中，只需调用1次核函数，成功削减1层循环。潜在地削减了VBO的个数，也将提升渲染效率。经过上述优化以后，纵向评价和横向评价结果如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CPU1 | CPU2 | CPU3 | GPU4 |
| GPU1 |  |  |  |  |
| GPU2 |  |  |  |  |
| GPU3 |  |  |  |  |
| GPU4 |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 优化前 | 优化后 | 提升幅度 |
| GPU1 |  |  |  |
| GPU2 |  |  |  |
| GPU3 |  |  |  |
| GPU4 |  |  |  |

削减循环层数以后，时间效率与CPU多核基本持平。[v304]

### 增大并发执行线程的数目

减少寄存器使用量，提升并行BLOCK的数量。kernel节省2个临时变量，时间无变化。[v305]

### 以空间换时间

以空间换时间，就是以耗费更多存储空间为代价减少运算量，从而提升时间效率。顶点变换包括位移和旋转，在执行新的变换之前，需要执行上一次变换的反变换恢复初始状态，再执行新的变换。2次变换，意味着2倍的计算时间，如果将顶点的初始状态保存下来，免去反变换恢复初始状态的环节，理论上可以节省一半的时间。经过上述优化以后，纵向评价和横向评价结果如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CPU1 | CPU2 | CPU3 | GPU4 |
| GPU1 |  |  |  |  |
| GPU2 |  |  |  |  |
| GPU3 |  |  |  |  |
| GPU4 |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 优化前 | 优化后 | 提升幅度 |
| GPU1 |  |  |  |
| GPU2 |  |  |  |
| GPU3 |  |  |  |
| GPU4 |  |  |  |

# 总结报告

展望

v195 cuda加速骨骼，效率不升反降，耗时150ms/渲染100次小海龟，大于CPU95ms

运算量在于关键帧插值、难点在于更新骨骼时考虑关节父子依赖

# 参考文献

1. Comparison of Nvidia graphics processing units, <http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units>
2. Comparison of Intel processors, <http://en.wikipedia.org/wiki/Comparison_of_Intel_processors>
3. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, <http://dl.acm.org/citation.cfm?id=1345220>
4. GPU高性能运算之CUDA 第4章 CUDA程序的优化, <https://skydrive.live.com/?cid=07c1a128483bbe69&id=7C1A128483BBE69%216659>
5. Kamran Karimi, Neil G. Dickson, Firas Hamze (16 May 2011). [A Performance Comparison of CUDA and OpenCL](http://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf). arXiv:1005.2581v3. Retrieved 12 January 2012.
6. Graphics programming for the web, <http://dl.acm.org/citation.cfm?id=2343491>
7. Real Time Skin Deformation with Bones Blending, <http://wscg.zcu.cz/wscg2003/papers_2003/g61.pdf>
8. Simulation of Fencing in Virtual Reality, <http://www.jarmilakavanova.cz/ladislav/diplomka/vf.pdf>
9. 基于可编程图形处理器的骨骼动画算法及其比较, <http://sdrv.ms/OgcgFl>
10. MilkShape 3D SDK, <http://www.milkshape3d.com/>
11. [Global Memory Usage and Strategy](http://developer.download.nvidia.com/CUDA/training/cuda_webinars_GlobalMemory.pdf)
12. [CUDA Warps and Occupancy](http://developer.download.nvidia.com/CUDA/training/cuda_webinars_WarpsAndOccupancy.pdf)

1. [Better Performance at Lower Occupancy](http://nvidia.fullviewmedia.com/gtc2010/0922-a5-2238.html)

1. [Identifying Performance Limiter](http://developer.download.nvidia.com/CUDA/training/cuda_webinars_dentifying_performance_limiters.pdf)