在可编程图形硬件上实现图像高动态范围压缩*

彭 韬[1,2]　胡耀华[2]　李在铭[1]

1. 电子科技大学　通信与信息工程学院　成都　610054

2. 微软亚洲研究院 北京 100080

【摘要】通过在亮度图像梯度域上对大梯度进行衰减，压缩图像亮度的动态范围，可以使高动态范围图像在被显示时，既能够适应常规的显示硬件，同时又充分保留了原始图像的细节信息，使得图像在被观察时能够重现真实场景的亮度效果。本文提出适合由图形处理器加速的快速算法，将整个处理过程通过可编程图形硬件实现，建立了快速的图像动态范围压缩技术，建立起适用于高动态范围图像显示的实时应用框架，使之不仅适用于基于图像的动态范围调整的绝大部分情况，还能够成为各种交互式图形应用的核心技术之一。

关 键 词　　图像处理、高动态范围压缩、泊松方程、可编程图形硬件

引言

逼真图形（图像）产生技术的最终目标是使生成的图像在被显示出来后，观察者所获得的感觉与映象要和他置身于真实环境中获得的一样，即被观察图像与真实场景不但携带的信息一致，给人带来的视觉感觉也要是一致的。

然而，尽管对图像生成技术的研究给人们提供了越来越好的结果和越来越快速的方法，目前的图像产生过程仍然不足以确保感觉上的逼真性。问题很大程度上出自图像的显示过程，由于常规图像显示设备只适合输出小动态范围亮度图像，使人们不能观察到图像源产生图像的全部信息，而这一过程又恰恰是整个图像产生过程的重要部分。特别是近年来高动态范围图像（high dynamic range image-HDRI）在计算机图形学领域变得越来越普遍而且重要，真实场景的HDR图像也变得非常容易获取[1]，图像显示硬件的局限性也就越发暴露出来。

本文通过在梯度域上对图像动态范围进行压缩调整，使图像在显示时既能满足显示设备的局限条件，又充分保留了图像信息，使图像在观察时与真实场景一致。同时，借助可编程图形硬件和先进的图形应用程序加速接口，提出了适合由图形处理器加速的快速算法，建立了快速的图像动态范围压缩技术，建立起高动态范围图像实时显示技术的应用框架。使之不仅适用于基于图像的动态范围调整的绝大部分情况，应用于数字摄影、电影艺术、科学图像增强，还能够成为目前交互式图形应用的核心技术，成为实现虚拟实景、交互 3D 应用的基础之一。

全文共分为四个部分，第一部分阐述图像动态范围压缩所面临的问题以及前人的工作，第二部分提出梯度域上图像动态范围压缩的数理模型，并对该模型进行分析得出优化后的数值解法，第三部分阐述在 GPU 上的实现过程及其优化，第四部分对试验结果进行了分析总结。

*作者简介：彭韬（1978－），男（汉族），四川成都人，硕士，主要从事计算机图形图像、多媒体通信、处理与网络综合服务技术的研究。

Email：pt13@etang.com

1 动态范围和高动态范围压缩

1．1 动态范围

在图像被显示的时候，一幅本身精确的图像并不能够保证带给我们对该场景真实的视觉感受，这是由于标准显示设备的缺陷所致。通常，人眼在观察目标时，可以看清目标的最低照度为 1 lx（勒克斯），而在夏天的中午，当目标照度达到 3×10^5 lx时，人眼仍可以看清目标。

一幅图像亮度级的最大值与最小值之比被称为动态范围（Dynamic Range），定义如下：

$$\alpha = Imax / Imin \qquad (1.1)$$

通过眼睛瞳孔的自动调节，从明亮的日光到星光，人眼分辨物体的动态范围可以达到100000000:1，即使在同一个适应场景内，不需调节，人眼也能分辨 10000:1 的亮度范围。然而，常规显示设备能重建的亮度动态范围仅仅是 100:1[2,3]。

为了准确地对一幅图像进行分析或者与现实场景相比较，人们希望显示出来的图像要与原始场景图像在视觉感受上尽可能相同。理想情况是，当人们在相同条件下观察一个真实的场景和一幅代表这个场景的图像，无论这个图像是计算机生成的还是照像机生成的，产生的图像和真实的场景应该具有相同的色调（tone）对比，即亮度等级是相对匹配的。

1．2 高动态范围压缩

高动态范围（HDR）的场景在现实世界中大量存在着，因此需要以某种方式将其图像的动态范围进行缩放，使之匹配只能输出低动态范围的显示设备。这种方式称为色阶重建 Tone Reproduction 或者色调映射 Tone Mapping，它提供了一种方法将现实场景的亮度值缩放（或者映射）到显示设备能显示的范围。除了压缩亮度范围，它还必须充分保留原始图像的感观质量（Perceptual Qualities），例如，重建算法必须在图像中保留诸如对比度、明亮程度、图像细节等信息，而这些信息往往会在动态范围压缩过程中被丢失。

动态范围的重建方法可以分为两种类型。一种是空域不变（Spatially Uniform）或者叫做全局动态范围压缩[4,5,6]。该类算法在对图像进行动态范围变换时，在每个象素上使用同一条变换曲线，变换曲线可以预先指定或者根据图像的内容获取。这类算法中以Ward Larson等人[5]基于直方图调整的动态范围重建技术为标志，其不足在于不变的变换曲线不能自适应图像的不同区域，导致结果图像在细节，颜色，明亮程度上损失。

另一种则是空域变化的（Spatially Varying）或者叫做局部动态范围压缩。该类算法针对图像不同的区域进行不同的变换。根据人类视觉系统（HVS）的不同模型，各种不同的算法在压缩动态范围的同时都以保留图像质量的某一方面为标准。早在 60 年代，Oppenheim[7]等人建立了一种图像多层（Multi Scale）亮度模型，将亮度图像分成大动态但是低频的成分和小动态但是反映细节的高频成分，然后对低频层在变换域上进行衰减而高频成分则保留下来。一直到 90 年代末期，各种算法都是在多层模型上针对不同的HVS模型进行调整，但是由于低频图像上采用的滤波函数特性不佳，在结果图像中物体边缘会产生严重光晕（halo）一直是困扰该类算法多年的问题[7,8,9]。1999 年，Tumblin[10]等人提出了LCIS算法，通过对图像不同细节的定义，提高了结果图像质量，但却使得算法变得非常低效，速度过慢。

在2002年的ACM SIGGRAPH会议上，三篇论文在该领域同时发表。Durand 和 Dorsey [11]在基于分层模型的基础上采用具备边缘检测的双边滤波（Fast Bilateral Filter）技术，避免了LCIS的缺陷，使基于分层模型的动态范围压缩算法获得了较高的性能和令人满意的结果。Fattal[12]等人则从梯度域上对亮度图像进行多尺度的衰减，再以新梯度图像恢复出亮度图像。Reinhard[13]等人则取法于摄影技术，将亮度范围分成不同区域（Zone），将HDR的不同区域单独映射到LDR的对应区域。

由于梯度域衰减算法在处理流程上的简洁性以及较高的性能和良好的结果，并且由于其相关数值解法具备的特殊性质，本文在对其改进的基础上，提出了针对GPU加速的快速算法，进而创建了对应的实时图像动态范围压缩技术及其应用框架。

2 在梯度域上的高动态范围压缩

尽管对人类视觉系统（HVS）的知识还非常有限，但是我们已经清楚这样一个事实，即HVS对相对亮度非常敏感，对绝对亮度并不敏感。在图像上，高动态范围来源于亮度的大幅度改变，所以，为了压缩动态范围，要对这些大幅度的亮度变化进行衰减。在Fattal[12]等人提出的方法中，首先通过对梯度的检测识别出大幅度的亮度变化，然后只对这些大幅度的变化进行衰减而对小幅度的变化进行保留或者适当提升，最后从新的梯度恢复出结果图像。

2．1 数理模型

设已知亮度图像的对数值为：$H(x,y)$，其梯度图像为 $\nabla H(x,y)$，如果有衰减函数 $\Phi(\|\nabla H\|)$，则可以得到衰减后的新的梯度图像：

$$G(x,y)=\nabla H(x,y)\cdot\Phi(\|\nabla H\|) \qquad (2.1)$$

本文采用与Fattal[12]等人相似的方法在多尺度梯度图像上求得衰减函数 $\Phi(\|\nabla H\|)$。对于二维图像的一般情况，不能对 $G(x,y)$ 直接积分而得到亮度图像，可以采用从所有可能的亮度图像中找出一个最接近的解，即通过最小二乘法，找到一幅亮度图像 $I(x,y)$，它的梯度逼近 $G(x,y)$。这样，结果图像 $I(x,y)$ 满足泊松等式：

$$\nabla^2 I=\operatorname{div}G \qquad (2.2)$$

上式中 $\nabla^2 I=\frac{\partial^2 I}{\partial x^2}+\frac{\partial^2 I}{\partial y^2}$，等式右边为 G 的散度 $\operatorname{div}G=\frac{\partial G_x}{\partial x}+\frac{\partial G_y}{\partial y}$。

求解式 2.2，可以得到动态范围压缩过后的亮度图像。由于对亮度图像的运算是在对数域上进行的，所以还要对结果图像进行指数化。最后，根据源图像的颜色，从结果亮度图像恢复出最终输出的彩色图像。

对于式 2.2 这样的线性系统等式，采用有限差分近似，即：

$$\begin{cases}\nabla^2 I(x,y)\approx I(x+1,y)+I(x-1,y)+I(x,y+1)+I(x,y-1)-4I(x,y)\\ \nabla H(x,y)\approx(H(x+1,y)-H(x,y),H(x,y+1)-H(x,y))\\ divG\approx G_x(x,y)-G_x(x-1,y)+G_y(x,y)-G_y(x,y-1)\end{cases} \qquad (2.3)$$

在边界处，假设图像亮度的差分为 0，例如：$I(-1,y)-I(0,y)=0$。

2．2 改进的泊松方程共轭梯度解法

式 2.2 近似后产生一个 N×N 的稀疏线性系统：

$$A \cdot x = b \qquad (2.4)$$

研究者通常对该系统采用全多重网格（FMG）的数值解法进行求解。由于针对规则稀疏矩阵的全多重网格(Full Multigrid,FMG)算法具备收敛速度较快，无需设定初始解，处理时间与n（这里n为矩阵A的元素个数）成正比的优势，所以目前多数求解泊松方程的系统均采用FMG算法[12,14]。

作为另一种求解该系统的通用方法，共轭梯度算法（Conjugate Gradient Method以下称CG）[15] 则具有迭代过程简单且易于优化的特点。该方法对A的运算只涉及到A或者其转置矩阵与向量的乘法，考虑到在A是稀疏矩阵的情况下，如果对其采取有效的存储方式或者简化其运算，该算法的求解过程同样是比较高效的。

根据泊松方程(2.2)的有限差分近似形式，线性系统（2.4）中矩阵 A 不但是规则的稀疏矩阵，而且具备对称性和单一值的矩阵元素。通过对通用的 CG 算法进行简化和改进，算法效率将得到了大幅度提高，同时由于其迭代步骤简单直接，使之更能在 GPU 上得到进一步优化。

观察矩阵A，由于A是对称矩阵，则A的转置矩阵A^T=A，通用CG算法可以首先简化如下：

设定初始解x_1，计算剩余量 $r_1 = b - A \cdot x_1$；

然后开始迭代求近似解 $x_{k+1} = x_k + \alpha_k p_k$；当 x 满足一定的收敛条件（比如检测误差 $\varepsilon = |A \cdot x - b| / |b|$）或者达到最大迭代次数后，迭代过程就结束。

对于第 k 次（$k \geq 1$）迭代，有

$$\begin{cases} \alpha_k = \dfrac{r_k \cdot z_k}{p_k \cdot A \cdot p_k}, \beta_k = \dfrac{r_{k+1} \cdot z_{k+1}}{r_k \cdot z_k} \\ A' \cdot z_k = r_k \\ r_{k+1} = r_k - \alpha_k \cdot A \cdot p_k, p_{k+1} = z_{k+1} + \beta_k \cdot p_k \end{cases} \qquad (2.5)$$

（其中， $p_1 = z_1$，A’ 为 A 的对角阵）

在不考虑边界条件的情况下，由式 2.3，矩阵 A 的对角元素均相等，使得

$$z_k = (-1/4) \cdot r_k, \quad \beta_k = |r_{k+1}|^2 / |r_k|^2$$

代入式 2.5 可进一步简化为

$$\begin{cases} \alpha_k = (-1/4) \cdot \dfrac{r_k \cdot r_k}{p_k \cdot A \cdot p_k}, \beta_k = |r_{k+1}|^2 / |r_k|^2 \\ r_{k+1} = r_k - \alpha_k \cdot A \cdot p_k, p_{k+1} = (-1/4) \cdot r_{k+1} + \beta_k \cdot p_k \end{cases} \qquad (2.6)$$

改进后的 CG 迭代流程如图 2.1 所示。

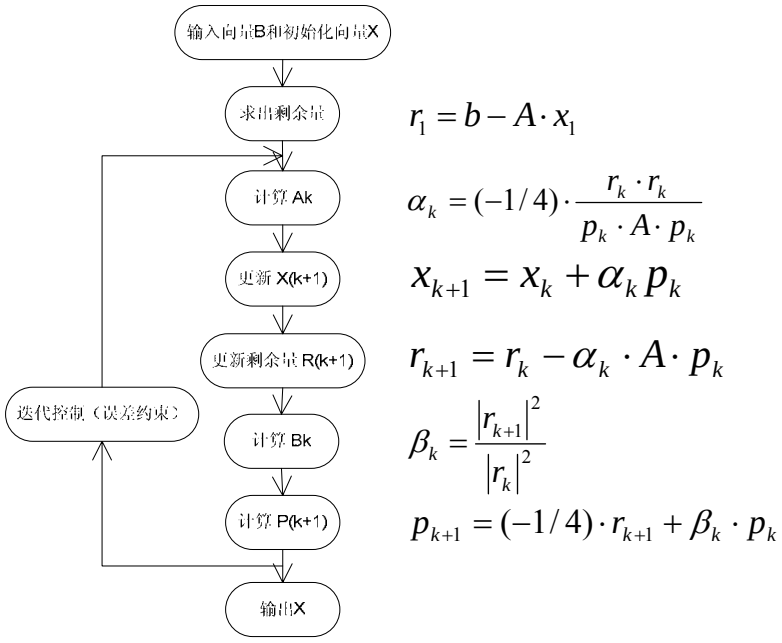

图 2.1 改进的 CG 迭代过程

由以上分析，整个算法涉及到向量点乘、向量线性组合、矩阵 A 与向量的乘法三种基本线性运算。考察矩阵 A 与向量的乘法，由于 A 中除去对角线元素为-4 外，每行最多有 4 个非零元素，且值均为 1，并且都分布在对角线元素两旁固定位置。针对矩阵 A 的这种结构，可以将 A 与向量的乘法运算转换为计算 5 个移位后的向量的线性组合，这不但避免了矩阵 A 的存储，同时将使整个求解过程不涉及任何基本的矩阵运算。这种方式将极大地加速运算过程。

对于 CG 迭代过程的初始解，由于输出结果为在对数域的亮度图像，其图像细节与源图像相似，所以我们采用直接将源亮度图像在对数域的值作为 CG 方程的初始解输入。

下图是经过上述针对泊松方程优化后的 CG 算法与通用 CG 算法以及 FMG 算法在 CPU 上运行的比较结果。

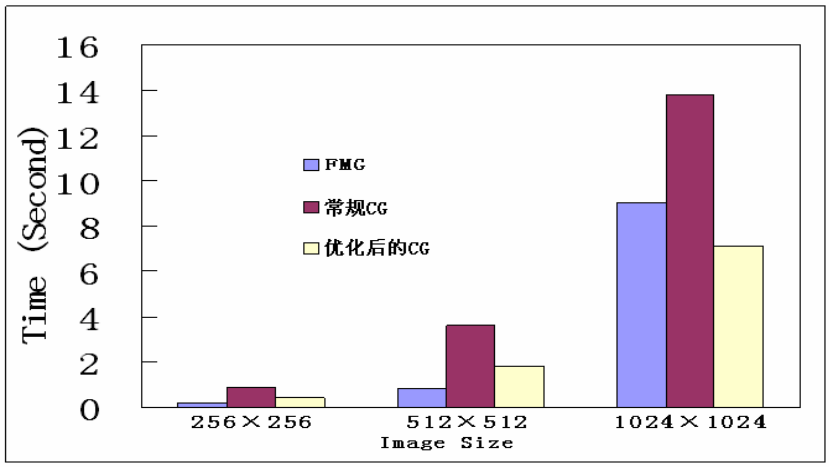

图 2.2 CG常规算法与优化之后性能比较[†]

（α=0.1, β=0.88, 在 CPU 上迭代 100 次结果）

可以看到，经过改进以后的 CG 算法，性能比通用算法已经有了显著提高，接近了 FMG 的性能，甚至在大尺寸图像的处理速度上已经超过了 FMG 算法。

[†]注：本文所有试验数据均在P4 2.8G(512M),ATI 9800 XT(256M)平台上获得

3 基于可编程图形硬件的实现

当今，即便是普通的图形加速卡，其中的图形处理单元（GPU）都已经发展成为具备强劲性能并且灵活易用的处理器。这些GPU不但提供了巨大的存储带宽和计算能力，它们所具备的可编程顶点以及象素处理单元也把对向量的运算支持提升到了浮点运算的精度。另外，高级语言的出现也对这些顶点及象素处理管线提供了良好的可编程性支持[16]。由于这些特性，可以将现代的GPU看作是一种通用的流式信息处理器（stream processor），它完全适用于对任何流信息模型进行处理。

由于 GPU 强大的并行处理能力，使它在流式信息计算性能上远远超过同时代的 CPU，将其应用于传统图形加速以外的数值计算上，架构新一代高性能计算体系，已经成为计算机科学的一个新兴领域。越来越多的基础数值算法被成功移植到 GPU 上，随之也产生了各种实时交互的模拟应用，比如对流体、光线物理行为的交互模拟，而不久前，这种计算密集型应用在普通计算设备上还是不能够甚至不可能现实地完成的。

我们基于改进的图像动态范围快速压缩算法（以下称 GHDRC），针对其流式信息处理特征，将其整个过程在 GPU 上进行了实现。以下三节分别阐述了在 GPU 上实现 GHDRC 算法的关键问题，即数据结构、基本运算、系统架构及优化。

3．1 数据结构

对于向量V，在GPU上的运算中由纹理图像来表示，如图 3.1 所示。我们使用长宽相等的正方形二维纹理来代表向量，并且使其宽度等于 2^n。这样做一方面由于GPU对方形纹理的处理支持最完善，另一方面针对GPU对纹理的处理方式，长宽取 2 的幂将有助于实现向量的基本运算。

对于 GHDRC 算法，信息的输入和输出本身就是一幅图像，可以直接由纹理图来表示和显示，这也避免了由于数据的输入输出带来的附加拷贝和转换操作。

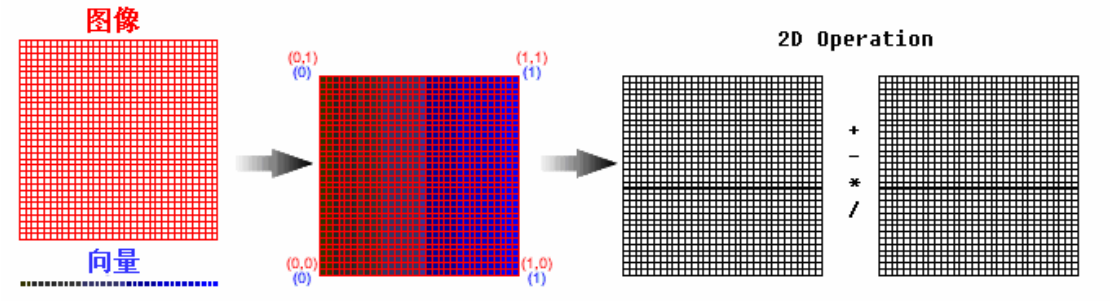

图 3.1 使用 2D 纹理表示向量

3．2 基本运算

根据对整个 GHDRC 算法的分析，除了在纹理上直接进行的各种简单图像处理操作外，大量运算都集中在求解式 2.2 的 CG 迭代过程中，这里涉及了向量的三种基本代数运算：

1. 向量与向量点积 $\vec{V} = \vec{V}_1 \bullet \vec{V}_2$
2. 向量的线性组合 $\vec{V} = s_1 \cdot \vec{V}_1 + s_2 \cdot \vec{V}_2$
3. 矩阵 A 与向量相乘 $\vec{V}_{out} = A \cdot \vec{V}$

下面分别对这三种运算进行讨论。

3．2．1 向量与向量点积

由于 GPU 的并行处理方式，我们将向量的点积过程分为“向量逐元素相乘（Vector Multiply）”与“向量元素求和”两个步骤。

“向量逐元素相乘”可以直接通过渲染器（Shader）逐纹理元素值相乘来完成。

对于向量元素求和这种需要对向量所有元素进行合并的运算（Reduce），则不能直接在GPU 上通过单个渲染周期对一个纹理完成处理。我们采用在源向量纹理的基础上创建多层纹理，每层纹理长宽分别是上层纹理的 1/2，这样最小一层纹理将只有一个纹理元素（Texel）。以最初层次的纹理为输入，由渲染器对纹理（渲染目标）进行逐层渲染，将前一层输入纹理每 4 个 Texel 的值之和写入下一层输出纹理对应的一个中心 Texel 上。这样直到最小一层纹理输出，它仅有的一个纹理元素值即是最初向量所有元素的和。整个过程可以通过下图表示：

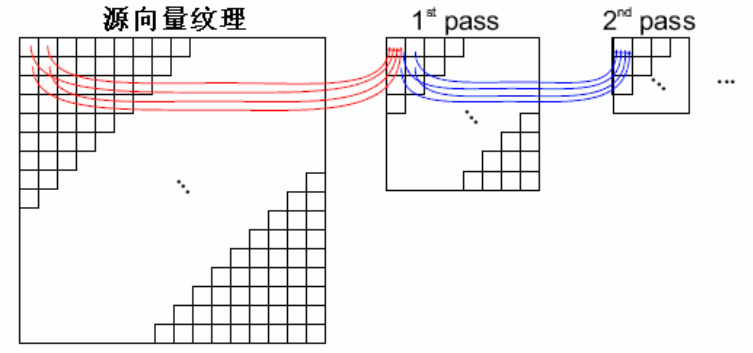

图 3.2　向量元素的结合运算过程

3. 2. 2 向量的线性组合

计算向量的线性组合$\vec{V} = s_1 \cdot \vec{V}_1 + s_2 \cdot \vec{V}_2$时，将代表向量$V_1$，$V_2$的纹理作为输入，通过渲染器对纹理元素逐值执行线性组合的操作，渲染输出的目标纹理即表示了结果向量，如下图所示。

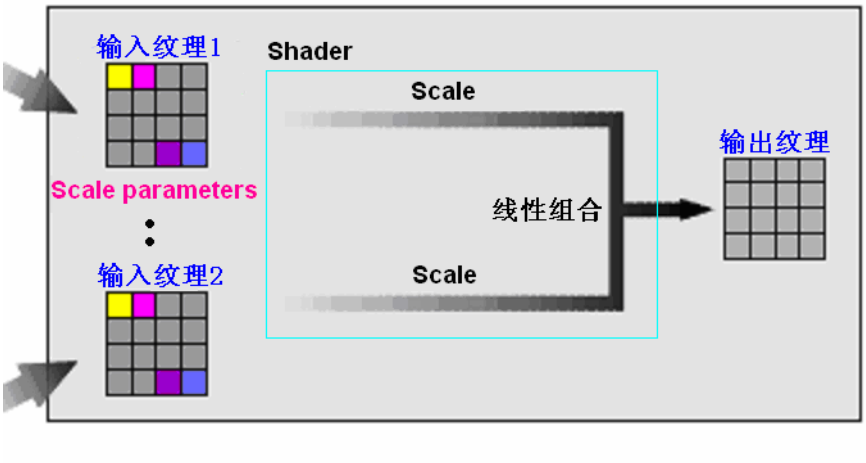

图 3.3 向量的线性组合

实现上，由于要在加入边界条件的情况下求解，所以对于式 2.6 中的部分标量与向量的乘积，将进行边缘调整。这在矩阵 A 与向量的相乘中也会发生，下面将对这点进行详细讨论。

3. 2. 3 矩阵 A 与向量的乘积

根据 A 的特殊情况，可用下式表示与向量相乘的过程：

$$A \cdot \vec{V} = (-4) \cdot \vec{V} + \overrightarrow{V_{+1}} + \overrightarrow{V_{-1}} + \overrightarrow{V_{+N}} + \overrightarrow{V_{-N}}$$

其中向量下标表示向量元素的移位，在边界处，第一项的系数为-3 或者-2（四角顶点）。这样，A*V 的运算简化为 5 个向量的线性组合。而由于这些向量均是源向量移位后的

结果，所以可以通过同一个渲染器在一个渲染周期之内一次性求得 A*V。

同样，对于边界情况，通过在边界处加入相邻边缘向量的线性组合，来调整最后的乘积结果使之满足边界条件。根据既定条件，边界的调整向量被预先设定，这样，结果调整将在 4 个快速的渲染周期之内完成。试验证明，附加的处理对整体性能影响不大，而且，由于满足边界条件使得 CG 迭代能够较为快速地收敛，则整个 CG 求解过程就可以在更少的时间内完成，从而系统性能得到提升。

3．3 系统架构及 GPU 优化

经过以上对向量基本运算的 GPU 化，我们将整个 CG 迭代过程直接移植到 GPU 上。如果在未针对 GPU 做任何优化的情况下直接移植 CG 过程，处理速度仅有小幅度提升，为了使 GPU 发挥其流式数据并行处理的优势，我们针对其独特的处理模式构造系统架构并进一步改进和优化数据结构。

对应于 GHDRC 过程的各个步骤，我们在 GPU 上创建了相应的处理模块，整个数据流程可用下图表示：

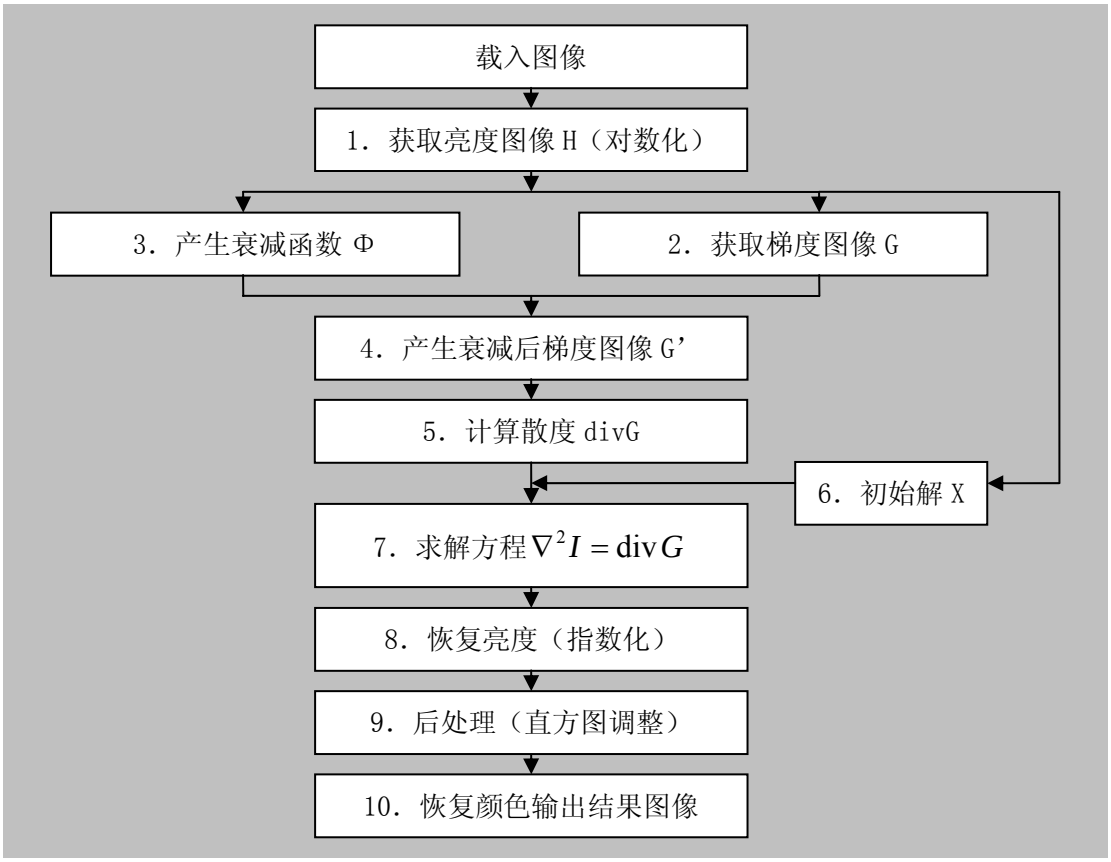

图 3.4 GHDRC 过程框图

我们将上图所示的整个 GHDRC 算法在 GPU 上实现（GPU GHDRC），表 3.1 是在输入图像大小 1024×1024，α=0.1, β=0.85, 迭代次数为 100 次的情况下各模块处理时间的统计数据，该表显示了 GHDRC 系统中各个单元模块的相对处理时间对比。可以看出，在整个 GHDRC 处理过程中，由于图像的对数化、指数化、求梯度、求散度等模块均可采用渲染器在单个 Pass（纹理处理周期）内完成，所以时间占整个处理过程的极小部分，而采用 CG 算法求解方程 $\nabla^2 I = \mathrm{div}\, G$ 的模块则占用了超过 90%的处理时间。所以，针对 GPU 独特的处理模式，进一步对求解式 2.2 的 CG 模块进行优化，表 3.1 中括号内数据表示了优化后各模

块的对比性能。

单元模块	处理时间（秒）	占总时间百分比
1. 获取亮度图像 H（对数化）	6.0065806e-5	0.0013%（0.0039%）
2. 获取梯度图像 G	6.936118e-5	0.0015%（0.0045%）
3. 产生衰减函数 Φ	0.048174005	1.067%（3.122%）
4. 产生衰减后梯度图像 G’	7.4476004e-5	0.0016%（0.0048%）
5. 计算散度 divG	6.8873167e-5	0.0015%（0.0045%）
6. 初始解 X	0.02973491	0.6586%（1.927%）
7. 求解 Possion 方程	4.07(优化后 1.0981162)	90.152%（71.182%）
8. 恢复亮度（指数化）	6.5326691e-5	0.0014%（0.0042%）
9. 后处理（直方图调整）	0.366261751	8.112%（23.742%）
10. 恢复颜色输出结果图像	6.9320202e-5	0.0015%（0.0045%）
总共处理时间	4.51457808905（优化后 1.54269428905）	

表 3.1 GPU GHDRC 各模块时间对比

首先，针对目前 GPU 支持的四通道（IEEE A32B32G32R32F）浮点纹理格式，将最初在单通道（IEEE R32F）上直接表示的数据采用 4 通道分量来表示。这样，对于表示图像数据的纹理来说，不仅其纹理元素（象素）总数可以减少到原来的 1/4，减少对 GPU 资源的占用率，而且 GPU 在每个纹理渲染周期将同时并行处理 4 个通道数据。试验证明，4 通道优化后的 CG 模块处理速度能够提高到原单通道处理的 3 倍以上，这符合预期的理论优化结果。

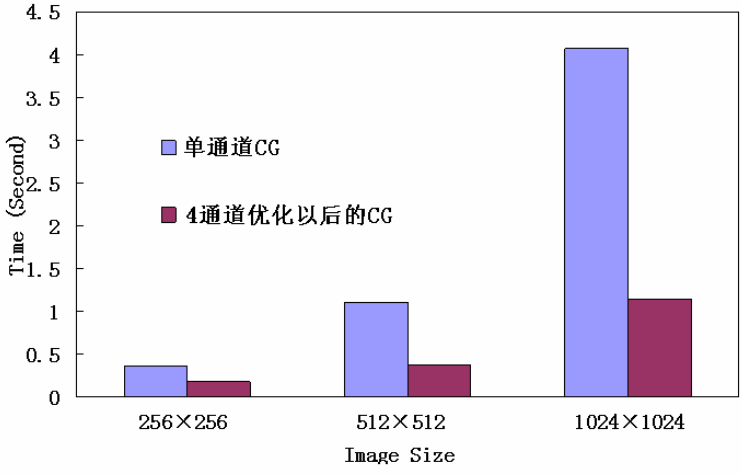

图 3.5 在 GPU 上实现时 CG 单通道与 4 通道性能对比

（α=0.1, β=0.88, 迭代次数为 100 次）

从上图可以看出，RGBA4 通道的全部利用使得 CG 迭代过程的性能有了很显著的提升，我们会在以后的改进工作中将这种优化方式应用到整个 GHDRC 过程中去，这必将进一步提高整体性能。

其次，认识到当前 GPU 可以以异步方式与 CPU 同时进行数据处理，为了充分发挥 GPU 的异步和并行性能，我们对 CG 算法再次简化。经过对多个图像样本的试验，在具备良好输出图像质量的情况下，我们确定了一个能满足通常情况的 CG 迭代次数 iTerMax，而不必在每次迭代之后都计算一次误差并与指定误差比较。这样，不仅减少了迭代中的运算单元，还可以减少对 GPU 运算结果的判断操作，尽可能消除 CPU 与 GPU 的数据交换，使两者异步运行。同时，我们对向量基本运算单元进行改进，将它们的输出或者输入的浮点数值参数直

接用 1×1 大小的纹理来表示，这样避免了 GPU 帧缓存中的数据与 CPU 不断进行交换，使整个 CG 模块都异步于 CPU 运行，从而使处理速度进一步得到了提高。

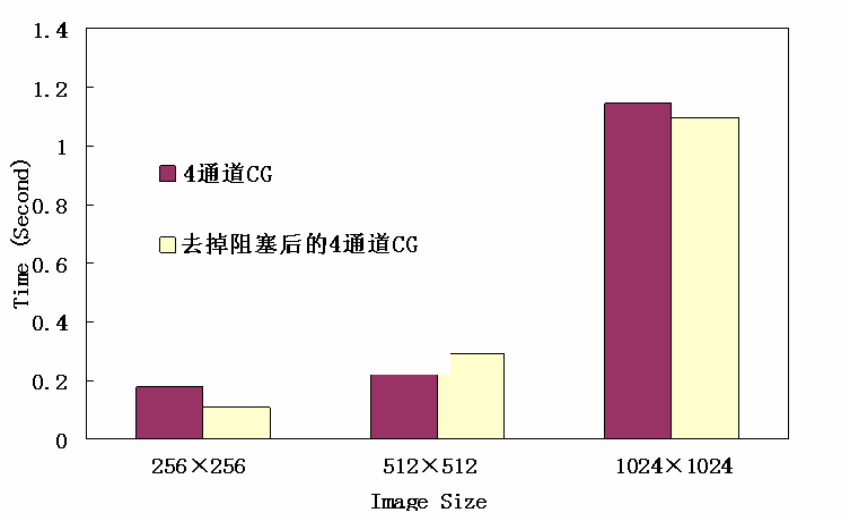

图 3.6 在 GPU 上实现时 CG 阻塞模式与异步模式性能对比
（α=0.1, β=0.88, 迭代次数为 100 次）

通过将向量的各种基本运算单元用 GPU 实现，我们实现了整个泊松方程求解过程的 GPU 化和异步化。为应用在高动态范围图像的实时处理和显示上，我们建立起一个借助 GPU 加速的 HDR 纹理图像实时渲染框架，如下图所示：

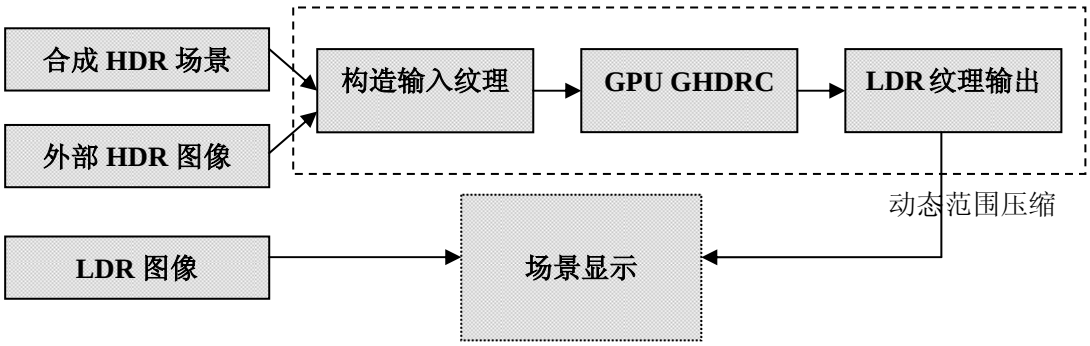

图 3.7 HDR 场景实时显示框架

上图中，整个系统架构左部分表示各种需要显示的场景图像，这些图像可以是计算机合成的模拟场景也可以是摄像机拍摄的实际场景。对于 HDR 输入的情况，将对整幅场景图像的动态范围进行压缩（上图中虚线框部分），最后系统输出代表场景图像的 LDR 纹理以供显示。

对于固定场景，动态范围压缩后产生的 LDR 结果图像可以被长期保存直接进行显示。而对于大多数交互应用，场景往往是动态变化的，这就需要即时更新场景的显示图像，这样，每一帧都将对场景源图像进行动态范围压缩，更新当前显示的 LDR 场景图像。

4 结论和思考

本文通过在梯度域上对图像动态范围进行压缩调整，使得高动态范围的图像能够在常规显示设备上得以良好地显示，使图像在观察时与真实场景一致。由于整个过程几乎全部在 GPU 上实现以及对主要单元的优化，试验中我们在得到令人满意的输出图像的同时也使系

统获得了非常好的性能。从下面的性能曲线图上可以看出，CG 算法以及整个 GHDRC 处理的时间是与图像的象素数目近似成正比的。

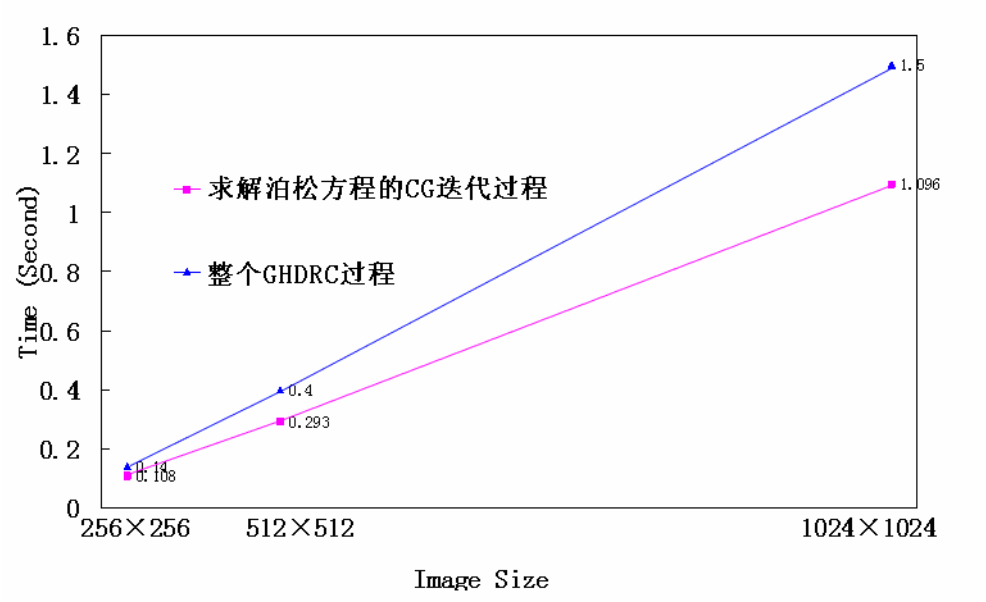

图 4.1 GPU 上实现时 CG 及整个 GHDRC 性能曲线

（α=0.1, β=0.88, 迭代次数为 100 次）

虽然由于硬件的限制，基本的向量运算均是在浮点精度上进行，运算结果在精度上较之双精度的 CPU 运算有所损失，但在一定的误差约束下，人眼已经无法分辨出浮点精度和双精度的质量差别了。

对于 CG 解法需要设定初始值的问题，本文采用直接代入原始图像作为初值的方法，该方式仅能比不设初值的情况减少迭代次数 3 到 4 次，性能提高不是很明显，这是由 CG 迭代过程的收敛特性所决定的。

我们的试验均在配置 P4 2.8G，512M 内存,ATI 9800 XT 图形加速卡的平台上进行，由于 ATI 9800 XT 图形加速卡对浮点运算的支持使得我们能够应用整个 GPU GHDRC 方案。从对以下 HDR 图像进行处理的过程和结果来看（图 4.2、图 4.3），通过对迭代次数的适当选择，该方案不仅适用于诸如数字摄影、电影艺术、科学图像增强等要求高质量静态图像输出的情况，随着图形处理单元（GPUs）的硬件技术发展，该方案也能适用于动态场景渲染系统，使其在输出满意质量图像的同时获得越来越高的帧速率，满足交互式图形应用中高动态范围场景显示的实时性需求。

图 4.2a　　图 4.2b　　图 4.2c

图 4.2a 欠曝光的 HDR 场景图像 ，图 4.2b 过曝光的 HDR 场景图像

图 4.2c　GPUGHDRC 输出图像（512*512，α=0.01，β=0.88，300 次迭代，0.98 秒）

图 4.3a　　图 4.3b　　图 4.3c

图 4.3a 欠曝光的 HDR 场景图像 ，图 4.3b 过曝光的 HDR 场景图像

图 4.3c GPUGHDRC 输出图像（512*512，α=0.01，β=0.88，50 次迭代，0.26 秒）

参 考 文 献

[1] Debevec, Ward, and Lemmon, HDRI and Image-Based Lighting, SIGGRAPH 2003 Course #19

[2] James A. etc. A model of visual adaptation for realistic image synthesis. In Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series, pages 249–258, New Orleans, Louisiana, August 1996. ACM SIGGRAPH / AddisonWesley. ISBN 0-201-94800-1.

[3] 何 烽，徐之海等，一种基于数字图像合成的扩展动态范围方法，光电工程，2003 年 20 月 第 30 卷 第 5 期 66－68 页

[4] Greg Ward. A contrast-based scalefactor for luminance display. In Graphics Gems IV, pages 415–421. Academic Press, Boston, 1994. ISBN 0-12-336155-9.

[5] Gregory Ward Larson, ect. A visibility matching tone reproduction operator for high dynamic range scenes. IEEE Transactions on Visualization and Computer Graphics, 3(4):291–306, October – December 1997. ISSN 1077-2626.

[6] Jack Tumblin, etc. Two methods for display of high contrast images. ACM Transactions on Graphics, 18(1):56–94, January 1999. ISSN 0730-0301.

[7] A. Oppenheim, etc. Nonlinear filtering of multiplied and convolved signals. In Proceedings of the IEEE, volume 56, pages 1264–1291, August 1968.

[8] Sumanta N. Pattanaik. etc. A multiscale model of adaptation and spatial vision for realistic image display. In Proceedings of SIGGRAPH 98, Computer Graphics Proceedings, Annual Conference Series, pages 287–298, Orlando, Florida, July 1998. ACM SIGGRAPH / AddisonWesley. ISBN 0-89791-999-8.

[9] D. J. Jobson, etc. A multiscale retinex for bridging the gap between color images and the human observation of scenes. IEEE Transactions on Image Processing, 6(7):965–976, July 1997.

[10] Jack Tumblin and Greg Turk. Lcis: A boundary hierarchy for detail-preserving contrast reduction. In Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series, pages 83–90, Los Angeles, California, August 1999. ACM SIGGRAPH / Addison Wesley Longman. ISBN 0-20148-560-5.

[11] Fr´edo Durand and Julie Dorsey. Fast bilateral filtering for the display of high dynamic range image. In John Hughes, editor, SIGGRAPH 2002 Conference Graphics Proceedings, Annual Conference Series, pages 257–265. ACM Press/ACM SIGGRAPH, 2002.

[12] Raanan Fattal, Dani Lischinski, and Micheal Werman. Gradient domain high dynamic range compression. In Proceedings of ACM SIGGRAPH 2002, Computer Graphics Proceedings, Annual Conference Series. ACM Press / ACM SIGGRAPH, July 2002.

[13] Erik Reinhard, Michael Stark, Peter Shirley, and Jim Ferwerda. Photographic tone reproduction for digital images. In Proceedings of ACM SIGGRAPH 2002, Computer Graphics Proceedings, Annual Conference Series. ACM Press / ACM SIGGRAPH, July 2002.

[14] Nolan Goodnight1, etc. A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware. Graphics Hardware 2003, pp. 1–11

[15] W. H., TEUKOLSKY, etc. 1992. Numerical Recipes in C: The Art of Scientific Computing, 2nd ed. Cambridge University Press.

[16] Kekoa Proudfoot, etc. A real time procedural shading system for programmable graphics hardware. In Proceedings of SIGGRAPH 2001, pages 159–170, August 2001.

High Dynamic Range Compression on Programmable Graphics Hardware

PENG Tao[1,2] HU Yaohua[2] LI Zaiming[1]

1. Communication and Information Engineering Institute of UESTC, ChengDu 610054
2. Microsoft Research Asia, BeiJing 100080

Abstract We present a system for compressing high dynamic range images to fit conventional display devices that are only capable of outputting a low dynamic range. In addition to manipulating the gradient field of luminance image by attenuating the large gradients‘ magnitudes, it can preserve fine details, resulting in an image which provokes the same responses as someone would have viewing the scene in the real world. Specifically, with the whole process built on programmable graphics hardware, we present an efficient algorithm based on GPU acceleration and provide a fast dynamic range compression technique. Furthermore, we describe a framework for rendering high dynamic range images in real time. It not only can be used in image based tone mapping but also is able to be a core technique in interactive graphics applications.

Key words Image Processing, High Dynamic Range Compression, Poisson Equation, Programmable Graphics Hardware

研究背景

本课题是在微软亚洲研究院高动态范围场景 3D 游戏项目的需求下提出的，旨在有效地服务于目前交互式图形应用，成为实现虚拟实景、交互 3D 应用的基础之一。