Journal of Image and Signal Processing 图像与信号处理, 2022, 11(1), 1-9
Published Online January 2022 in Hans. http://www.hanspub.org/journal/jisp
https://doi.org/10.12677/jisp.2022.111001

Hans汉斯

基于多进程的焦点堆栈图像融合方法

尹少齐[1,2]

[1]海克斯康制造智能技术(青岛)有限公司，山东 青岛
[2]七海测量技术(深圳)有限公司，广东 深圳

收稿日期：2021年12月14日；录用日期：2021年12月28日；发布日期：2022年1月13日

摘 要

传统上为了拍摄前景与后景均清晰的图像，相机光圈应当尽可能的小。但是受到衍射极限的影响，过小的光圈会造成图像发生畸变。然而通过将多张相同构图但不同焦距的图像合成，可克服上述困难。本方法包含硬件设备和算法设计，重点在算法设计上，包括高斯滤波、拉普拉斯算子、均值滤波以及探测点的极大值映射，最终实现图像的合成。除此而外，算法中还加入了多进程的思想，使得算法在获取同样锐度图像的条件下，程序运行时间极大缩短，这通过仿真实验得以论证。本论文提出的算法适用于多核CPU的处理，具备普适性。

关键词

焦点堆栈，多进程，图像合成，多核CPU

Method of Focus Stacks' Fusion Based on Multi-Processing

Shaoqi Yin[1,2]

[1]Hexagon Manufacturing Intelligence (Qingdao) Co., Ltd., Qingdao Shandong
[2]Seven Ocean Metrology (Shenzhen) Co., Ltd., Shenzhen Guangdong

Received: Dec. 14th, 2021; accepted: Dec. 28th, 2021; published: Jan. 13th, 2022

Abstract

Traditionally, the camera's aperture should be as small as possible in order to make the foreground and the background clear, respectively. However, affected by the diffraction limit, tiny apertures can cause the image distortion. The above difficulties can be overcome by merging multiple images with the same compositions but with different focus. The thesis includes hardware

文章引用: 尹少齐. 基于多进程的焦点堆栈图像融合方法[J]. 图像与信号处理, 2022, 11(1): 1-9.
DOI: 10.12677/jisp.2022.111001

and algorithm design, emphasizing on algorithm design. The image can be eventually fused including in Gaussian filtering, Laplacian operator, mean filtering and maximum mapping of detection points. Besides, in order to greatly shorten the running time of program, the multi-process is added to the algorithm without deprived of any sharpness, demonstrated by simulation. The algorithm is suitable for the multi-core CPU, universally.

Keywords

Focus Stack, Multiprocessing, Image Fusion, Multi-Core CPU

1. 引言

在工业检测领域和摄影领域，当为了获得图像中所有细节部分均清晰的效果，传统方法是选用小光圈的相机来进行拍摄。然而当光圈过小时，光通过光圈时就会发生衍射效应。根据瑞利判据，光学系统存在衍射极限，即艾里斑的半径 x [1]：

$$x = \frac{1.22\lambda}{d} \tag{1}$$

其中 λ 为光的波长，d 为光圈的直径，f 为相机镜头的焦距。当艾里斑尺寸大于像素尺寸时，便会使得图像发生畸变。

为了克服上述困难，考虑采用数字图像处理的技术。通过将同一构图进行多焦距的采样，提取有效信息并将所有图像融合，可最终得到大景深的图像[2] [3]。正是因为焦点堆栈所具备的优势，因此其应用得到了拓展，斯坦福大学的 Marc Levoy 等人在显微摄影中成功利用该技术对昆虫的腿部进行成像[4]，突破了传统显微镜的空间分辨率和角分辨率的衍射极限；Samuel W. Hasinoff 等人利用该技术提出了计算 3D 形状的新方法[5]，解决了三维重构时对于如毛发等细微物体复杂三维排列的问题[6] [7] [8]。然而，现今的图像融合算法存在程序运行时间长的问题，这会导致工业检测时间过长，检测效率低的问题；或者是摄影过程中合成时间过长，无法及时地查看图像合成效果，导致图像拍摄过程中存在有部分漏拍的情况时，摄影师无法及时地对拍摄有很好的把控。现对焦点堆栈算法进行算法设计和多进程处理，能够有效的提升图像显示效果和程序运行时间，为工业检测和微距摄影的研究提供了有效的应用拓展。本文的主要思想在于对每一张图像均提取高频信息，通过在每一个像素点位置处选取高频信息的最大值进行图像融合，从而得到合成图像。

2. 项目需求

本项目主要是用来对不同焦距的图像进行快速合成，最终实现在一张图片中所有细节均呈现清晰的效果，即超景深合成图像。运用卷积方法如 Sobel 算子、Prewitt 算子、Roberts 算子、Laplacian 算子以及 Canny 算子进行梯度边缘检测，虽然图像检测的精度高，但是程序的运行时间长；而若对图像直接进行简单处理，类似于对图像进行算数平均或者是小波变换，虽然程序运行时间短，但是图像的合成效果不佳。因此本项目考虑在图像检测精度得到保证的情况下，尽可能地压缩程序运行时间。

3. 光路设计

光路设计如图 1 所示，LD 光源发出的光经过待测物，最终被 CMOS 相机(MV-CA050-20GM)所接收。CMOS 相机安置在减震平台的支架上，可做从左往右的平移运动；减震平台的支架可做从上至下的垂直运动以及从前至后的平移运动。CMOS 相机先对待测物的最左端聚焦后拍摄(绿色虚线部分)，之后每间隔 1 s，通过将减震平台的支架做从下至上的垂直运动，让 CMOS 相机的焦点往右平移一段距离。此时进行聚焦拍摄，直到 CMOS 相机运行到待测物最右端，进行最后一次聚焦并拍摄，CMOS 相机便停止运行，图像采集完毕。每一次 CMOS 相机采集到的图像数据均传递给计算机，为焦点堆栈图像的合成做下一步准备。这样在不同焦平面上的物体信息，便通过 CMOS 记录了下来。

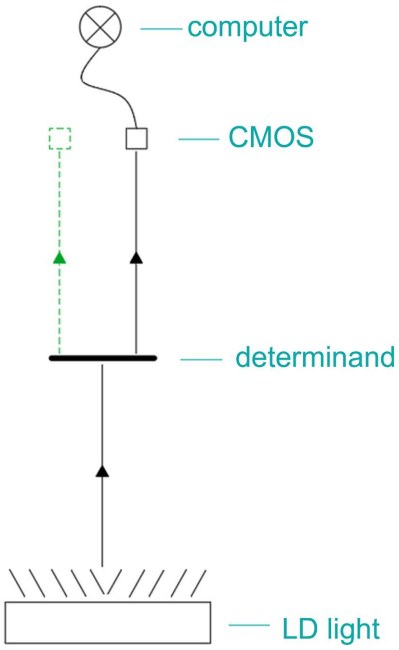


Figure 1. The beam path of schematic setup for focus stacking based on multi-processing
图 1. 基于多进程的焦点堆栈系统光路图

4. 软件设计

本系统待处理的图像是通过工业级 CMOS 相机(MV-CA050-20GM)进行拍摄的，与其配置的镜头(MT05-110-HR-D1.5)为 0.5 倍，工作距离 110 mm。后续图像处理采用 python 编程语言，因为这能够开源地实现该项目。

4.1. 流程处理

算法是本部分的核心，其流程图如图 2 所示，包括了图像导入部分、图像处理部分和图像融合部分。在经过图 1 所示的光路系统对物体信息采集后，计算机中便保存了 N 张不同焦面，但构图一致的图像。将图像利用多进程进行导入后，便可通过滤波处理以及边缘梯度检测获取到每一张图像的边缘信息，最后通过选取最大梯度信息进行图像融合，便可得到一幅超景深的合成图像。其详细步骤如下。

首先，利用 CMOS 相机对同一物体从左往右地进行聚焦拍摄，其光路图如图 1 所示，一共拍摄了 N 张图片。导入图片时，我们采用 opencv 的 imread 函数来对待处理的图像数据进行导入，为后续进行图像处理做准备。

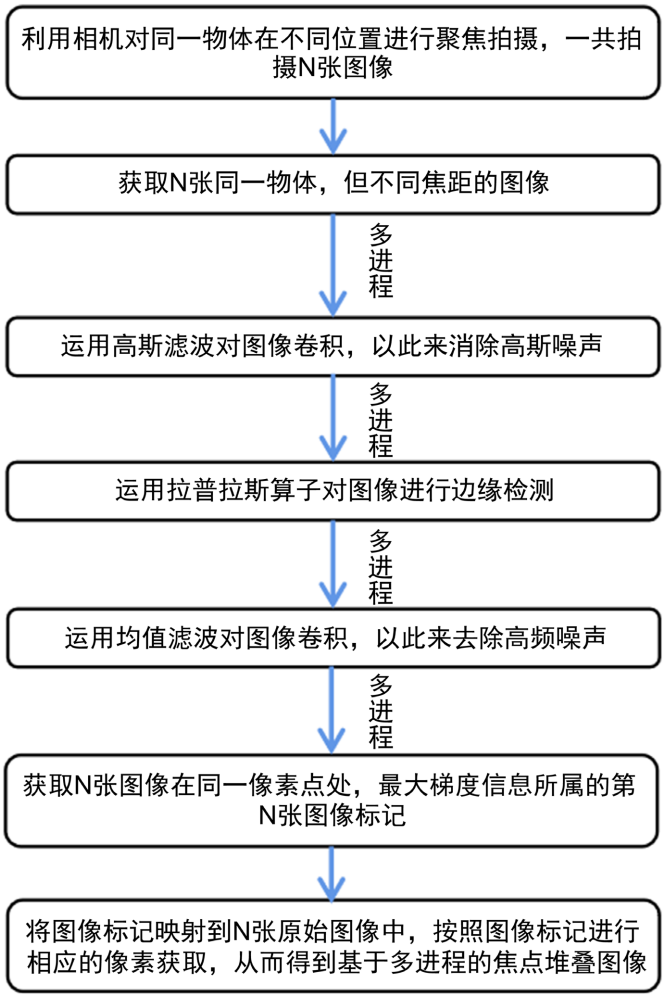


Figure 2. Flow chat of the system for focus stacking based on multi-processing
图 2. 基于多进程的焦点堆栈系统流程图

为了避免当光照不均，或高温引起的传感器噪声而带来的高斯噪声影响，我们可通过高斯滤波来减小其对图像处理的影响。若此时我们正准备处理第 m 张图像 $I_m(x,y)$，其中 $m\in[1,N]$，N 为图像的总张数，通过卷积操作，我们可得到处理过后的第 m 张图像 $O_m(x,y)$，其公式为[9]

$$O_m(x,y)=I_m(x,y)*K_m(x,y,\sigma) \tag{2}$$

式中 x 为输入图像的横向坐标，y 为输入图像的纵向坐标，$K_m(x,y,\sigma)$ 为第 m 张图像的卷积核，$*$ 为卷积运算符。此时的 $K_m(x,y,\sigma)$ 为第 m 张图像的高斯卷积核，公式如下：

$$K_m(x,y,\sigma)=\frac{1}{2\pi\sigma^2}\exp\left[-\frac{\left(x-\frac{W-1}{2}\right)^2+\left(y-\frac{H-1}{2}\right)^2}{2\sigma^2}\right] \tag{3}$$

其中 σ 为标准差，W 为输入图像 $I_m(x,y)$ 的宽，H 为输入图像 $I_m(x,y)$ 的高。之后我们可对图像进行梯度边缘检测，通过确定二阶微分为 0 的点，从而来判断图像的极值点，而这些极值点便为图像中的轮廓，即边缘部分。这里我们采用的是拉普拉斯算子来进行处理，公式如下：

$$\begin{aligned}\nabla^2 O_m &= \frac{\partial^2 O_m(x,y)}{\partial x^2} + \frac{\partial^2 O_m(x,y)}{\partial y^2} \\ &\approx O_m(x+1,y) + O_m(x-1,y) + O_m(x,y-1) + O_m(x,y+1) - 4O_m(x,y)\end{aligned} \tag{4}$$

然而由于拉普拉斯算子对噪声并不具备过滤的功能，因此运用该算子来对图像进行处理会造成边缘点的误判，最终导致图像边缘定位不够精准。考虑到后续我们会对所有图像的极大值信息进行比较，而通过拉普拉斯算子检测出来图像的极大值点，这属于高频信息，那么所需处理的噪声便转变为了高频噪声。为了解决上述问题，我们又采用均值滤波来对拉普拉斯算子检测后的图像进行噪点剔除工作，因为均值滤波适合消除图像的高频噪声，具备实现图像模糊、图像平滑的功能。公式如下：

$$F_m(x,y) = O_m(x,y) * K_m(x,y) \tag{5}$$

其中 $F_m(x,y)$ 代表了第 m 张图像经过均值滤波后输出的图像，$O_m(x,y)$ 代表了第 m 张图像经过拉普拉斯算子处理过后的图像，$*$ 为卷积运算符。$K_m(x,y)$ 代表了第 m 张图像的卷积核，此时的卷积核为均值滤波卷积核，公式如下：

$$K_m(x,y) = \frac{1}{H \times W}[1]_{H \times W} \tag{6}$$

其中 W 为经过拉普拉斯算子处理过后图像 $O_m(x,y)$ 的宽，H 为经过拉普拉斯算子处理过后图像 $O_m(x,y)$ 的高。另外均值滤波卷积核是可分离算子，通过卷积核的分离性和卷积的结合律，可减小程序的运算量。可分离的均值滤波卷积核公式为：

$$K_m(x,y) = \frac{1}{1 \times W}[1]_{1 \times W} * \frac{1}{H \times 1}[1]_{H \times 1} = \frac{1}{H \times 1}[1]_{H \times 1} * \frac{1}{1 \times W}[1]_{1 \times W} \tag{7}$$

其中 $*$ 为卷积运算符。经过滤波处理和拉普拉斯算子对边缘信息提取后，我们便进入图像融合部分。我们通过把均值滤波处理后的 N 张图像按照像素点的位置 x_i 和 y_j，进行逐一比较，其中 $1 \le x_i \le W$，$1 \le y_j \le H$，$i, j \in [1, N]$，从而选取每一个像素点位置所处的最大值点所属的图像标记 k，$k \in [1, N]$。公式如下：

$$k = \arg\max\left(F_m(x,y)\right)_{H \times W} \tag{8}$$

之后我们将得到的图像标记 k 映射到原始图像堆 $I(x,y)$ 中，按照图像标记 k 对原始图像堆 $I(x,y)$ 进行像素点提取，直到提取完整个 $H \times W$ 个像素点。公式如下：

$$M(x,y)_{H \times W} = I_k(x,y)_{H \times W} \tag{9}$$

其中 $M(x,y)$ 为基于多进程的焦点堆栈合成的图像，$I_k(x,y)$ 为在像素点 (x,y) 处第 k 张图像的像素值。这样我们便可实现基于多进程的焦点堆栈图像的合成，合成图像中的每一个像素位置均拥有通过图像处理后的最大像素值，至于多进程的部分我们接下来会谈到。

4.2. 多进程处理

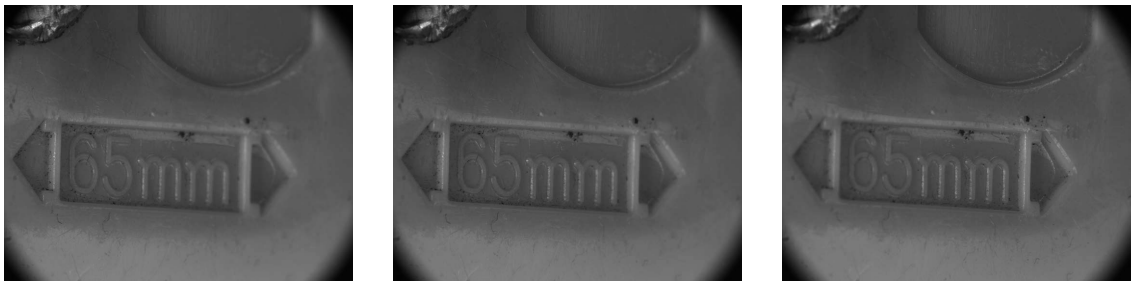
为了加快程序运行速度，我们采用了多进程的方式。如图 2 所示，多进程方式涵盖了从获取 N 张原始图像到对图像进行均值滤波处理，基本上涵盖了整个流程的大部分。

线程是操作系统进行运算调度的最小单位，是进程中的实际运行单位。但是由于 python 有 GIL (Global Interpreter Lock)的存在，这会对多线程处理造成很多限制，因此这里不考虑多线程处理。进程指的是可以独立运行的程序单位，当存在多核 CPU 时，通过多进程处理，就相对于并行处理程序，这可极大提升程序运算效率。

在多进程处理中，进程池的池化处理有着调度的作用。池化处理是指当进程数小于池中规定的最大值时，用户发出的请求可被当作一个新的进程进行并行处理；而当进程数大于或等于池中规定的最大值时，用户发出的请求需被等待，直到某一个进程结束，用户发出的请求方可作为一个新的进程进行并行处理。池化处理的优势在于有效的避免了计算机资源消耗过大，最终导致程序运行效率不高的不利情况。因此，我们采用 multiprocessing 安装包中的 Pool 函数来对程序进行调度，从而实现焦点堆栈图像的多进程处理。

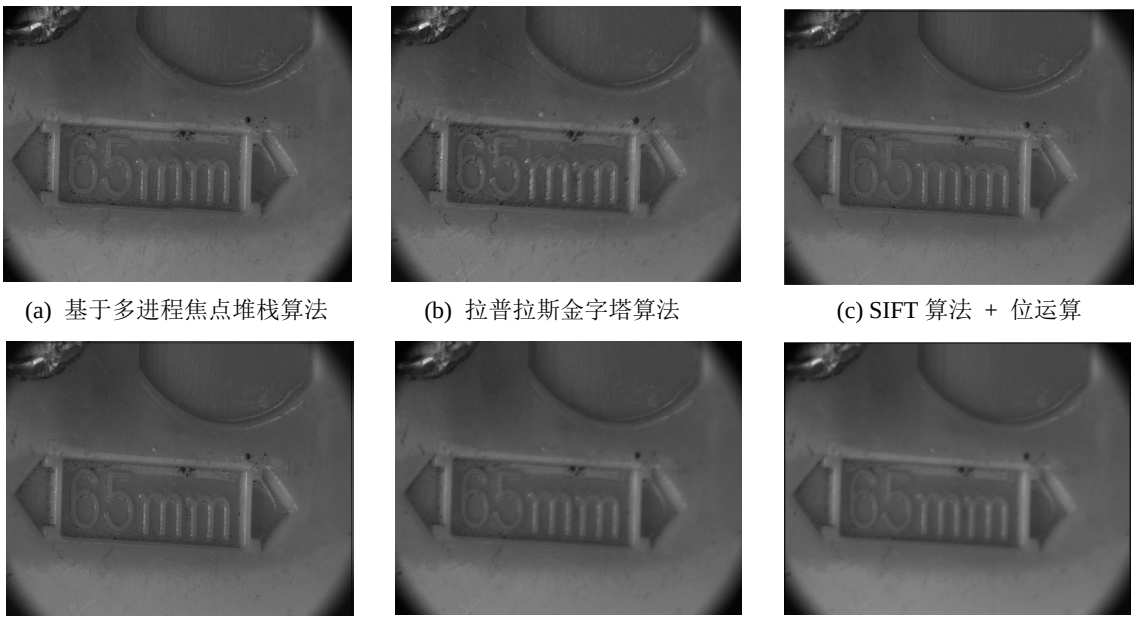
5. 实验结果

经过如图 1 所示的实验装置对待测物拍摄后，最终我们一共获得 4 张焦距位置不同的图像(2592 pixel × 2048 pixel)。如图 3 所示我们展示了其中三张图像，可以发现图像中的清晰部分从最左端最终到达最右端，代表了 CMOS 相机(MV-CA050-20GM)焦点在从左往右的移动。我们采用 12 核 CPU 来对图像进行并行处理，CPU 型号为 Intel 酷睿 i710750H，GPU 型号为 GeForce RTX 3060。为了与串行算法相对比，我们采用了另外五种不同算法来作为参照，其效果如图 4 所示。



(a) 焦点在左方的图像　(b) 焦点在中间的图像　(c) 焦点在右方的图像

Figure 3. Samples
图 3. 采集样图



(a) 基于多进程焦点堆栈算法　(b) 拉普拉斯金字塔算法　(c) SIFT 算法 + 位运算

(d) SIFT 算法 + 最大值映射　(e) 算数平均算法　(f) 小波变换算法

Figure 4. Displays of merging images by different algorithms
图 4. 不同算法的合成图像

图 4 中的图 4(a)图是通过本文的算法处理出来的图像；图 4(b)图是通过拉普拉斯金字塔算法处理出来的图像，主要思想是先通过对每一张图逐级下采样从而获得高斯金字塔，记录完高斯金字塔每一级下采样后，再上采样与下采样前的差异最终可重构图像；图 4(c)图是通过 SIFT 算法以及位运算来获得的图像，主要思想是在不同的尺度空间上查找关键点，并计算出关键点的方向，在得到每一张图像关键点后，通过把图像关键点的比较用布尔值表示，最终按照布尔值叠加得到重构图像；图 4(d)图是通过 SIFT 算法以及最大值映射来获得的图像，主要思想是获得每一张的关键点后，在每一个像素点的位置确定最大关键点所属的图像编号，最后将图像编号映射到原图像中得到重构图像；图 4(e)图是通过算数平均方法来获得图像，主要思想是将所有图像叠加后，除以总张数便可得到重构图像；图 4(f)图是通过小波变换来获得图像，主要是通过选择合适小波基来获得图像的高频信息，将处理后的图像叠加后得到重构图像。

为了衡量各算法在程序运行时间、CPU 利用率和内存使用率的情况，我们通过表 1 展示了相应的参数，其中表 1 展示的算法所处理的图像与图 4 编号相对应。为了显示不同算法之间程序运行时间的差异，我们通过运行时长耗比 C 来进行表示，公式如下：

$$C = \frac{T_i}{T_a} \tag{10}$$

其中 T_i 为图 4 中编号为 i 所属的算法程序运行时间，$i = \mathrm{b}, \mathrm{c}, \cdots, \mathrm{f}$；$T_a$ 为基于多进程的焦点堆栈算法程序运行时间。

Table 1. Running parameters of different algorithms
表 1. 各算法的运行参数

算法名称	对应图 4 的编号	运行时间/s	CPU 利用率/%	内存使用率/%	运行时长耗比
基于多进程的焦点堆栈	a	0.58	14.9	44.7	
拉普拉斯金字塔算法	b	51.88	10.4	44.1	89.4
SIFT 算法 + 位运算	c	1.63	7.2	42.4	2.8
SIFT 算法 + 最大值映射	d	20.70	5.3	42.5	35.7
算数平均算法	e	0.39	10.7	43.4	0.7
小波变换算法	f	0.93	5.4	43.5	1.6

将表 1 和图 4 综合来看，我们发现本论文提出的基于多进程的焦点堆栈图像算法在运行时间和图像效果上取得了不错的表现，从程序运行时长耗比来看，只有算数平均算法的程序运行时间比基于多进程的焦点堆栈图像算法的程序运行时间要短，其余算法均超过了基于多进程的焦点堆栈图像算法的程序运行时间，甚至拉普拉斯金字塔算法的程序运行时长耗比是基于多进程的焦点堆栈图像算法的 89.4 倍。然而从图像效果来看，尽管算数平均算法的程序运行时间比基于多进程的焦点堆栈图像算法的程序运行时间要短，但是最终合成的图像却不够锐利，字迹、图像均有不同程度的模糊。小波变换算法得出的图像也和算数平均算法得出的图像显示效果类似，处理效果不佳。不过，拉普拉斯金字塔算法和以 SIFT 为主的两种算法均与基于多进程的焦点堆栈图像算法的显示效果差不多，图像足够锐利，易于很好辨认细节。另外，之所以图 4 中所有合成图像有一定重影或缺失，是因为拍摄图像的过程中 CMOS 相机有一定振动，

导致每一张图像有无法避免的错位。

为了凸显出多进程的优势，我们以本论文的焦点堆栈图像算法来比较多进程与串行的效率，通过利用阿姆达尔定律[10]，我们可得出并行处理前执行速度与并行处理后执行速度的加速比数值 S_p，以此来阐明并行化后的效率提升情况，公式如下[11]：

$$S_p = \frac{T_1}{T_p} \tag{11}$$

其中下标 p 代表并行处理的 CPU 核心数，T_1 代表并行处理前程序所耗费的时间，T_p 代表并行处理后程序所耗费的时间。为了衡量并行处理条件下，多进程的效率数值 E_p，我们有如下公式[12]：

$$E_p = \frac{S_p}{p} = \frac{T_1}{T_p p} \tag{12}$$

其中 p 为并行处理的 CPU 核心数。多进程处理的图像与串行处理的图像如图 5 所示，串行处理与并行处理的图像对比参数如表 2 所示。

(a) 基于串行的焦点堆栈算法　　(b) 基于多进程的焦点堆栈算法

Figure 5. Merged image by serial algorithm and parallel algorithm, respectively
图 5. 串行与并行算法合成的图像

Table 2. Running parameters with serial algorithm and parallel algorithm
表 2. 串行算法与并行算法的运行参数

算法名称	运行时间/s	CPU 利用率/%	内存使用率/%	并行加速比	并行效率/%
基于串行的焦点堆栈	4.49	8.6	50.5		
基于多进程的焦点堆栈	0.58	14.9	44.7	7.74	64.5

从图 5 的图像显示效果来看，基于多进程的焦点堆栈算法和基于串行的焦点堆栈算法得到的图像锐度一致。从表 2 看出当使用串行的焦点堆栈算法时，CPU 的利用率会大幅度下降到只有 8.6%，而内存使用率会有少许上升至 50.5%。从并行加速比来看，串行程序所消耗的时间是并行程序所消耗时间的 7.74 倍，平均每一个 CPU 核的运行效率为 64.5%。这意味着，通过多进程的改进，尽管 CPU 利用率会有一定程度的上升，但是程序运行时间会被极大减少，并且最终处理出来的图像与串行程序处理出来的图像无异，因此这是极为具备优势的，尤其是在 CPU 核心数越多的情况下。

6. 结论

本文提出的焦点堆栈图像算法是为了便于在工业领域对于瑕疵品进行检测或在摄影技术上有所提

升，通过将焦距不同的图像进行合成，可最终得到超景深的图像，这便于我们从图像中提取有效且大量的信息。本文根据多核 CPU 的思路提出了基于多进程的焦点堆栈图像融合方法，通过将不同的算法进行对比，如拉普拉斯金字塔算法、基于 SIFT 的算法、图像算数平均算法和小波变换算法，若将图像显示效果和程序运行时间综合来看，本文提出的算法具备很大优势。为了探究多进程方法对程序运行时间的提升情况，我们又对多进程算法与串行算法进行对比，发现相比于串行算法，多进程算法的 CPU 利用率虽然有一定程度的上升，但是在程序运行时间上的确缩短了不少，串行程序运行时间是并行程序运行时间的 7.74 倍。随着计算机 CPU 核心数的增加，并行算法与串行算法的运行时间应当还会进一步拉开，这对于处理张数越多的图像更具备优势。然而，本论文提出的算法对合成有一定位移的图像却稍显不足，图像容错性不够，这在未来是应考虑改进的方向。

参考文献

[1] 玻恩, 沃尔夫. 光学原理[M]. 沈常宇, 金尚忠, 译. 北京: 电子工业出版社, 2006.

[2] Jacobs, D.E., Baek, J. and Levoy, M. (2012) Focal Stack Compositing for Depth of Field Control. Stanford Computer Graphics Laboratory Technical Report, 2012-1.

[3] Kutulakos, K. and Hasinoff, S.W. (2009) Focal Stack Photography: High-Performance Photography with a Conventional Camera. *The Eleventh IAPR Conference on Machine Vision Application*s, Yokohama, 20-22 May 2009, 332-337.

[4] Levoy, M., Ng, R., Adams, A., *et al*. (2006) Light Field Microscopy. *SIGGRAPH*06: *Special Interest Group on Computer Graphics and Interactive Techniques Conference*, Boston, 30 July 2006-3 August 2006, 924-934. https://doi.org/10.1145/1179352.1141976

[5] Hasinoff, S.W. and Kutulakos, K.N. (2009) Confocal Stereo. *International Journal of Computer Vision*, **81**, 82-104. https://doi.org/10.1007/s11263-008-0164-2

[6] Zitnick, C.L., Kang, S.B., Uyttendaele, M., *et al*. (2004) High-Quality Video View Interpolation Using a Layered Representation. *ACM Transactions on Graphics* (*TOG*), **23**, 600-608. https://doi.org/10.1145/1015706.1015766

[7] Fitzgibbon, A., Wexler, Y. and Zisserman, A. (2005) Image-Based Rendering Using Image-Based Priors. *International Journal of Computer Vision*, **63**, 141-151. https://doi.org/10.1007/s11263-005-6643-9

[8] Hertzmann, A. and Seitz, S.M. (2005) Example-Based Photometric Stereo: Shape Reconstruction with General, Varying BRDFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**, 1254-1264. https://doi.org/10.1109/TPAMI.2005.158

[9] Gonzales, R.C. and Woods, R.E. (2002) Digital Image Processing. Prentice-Hall Inc., Upper Saddle River.

[10] Amdahl, G.M. (1967) Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *Proceedings of the AFIPS* '67 *Spring Joint Computer Conference*, Atlantic City, 18-20 April 1967, 483-485. https://doi.org/10.1145/1465482.1465560

[11] Seshadri, G., Jain, R. and Mittal, A. (2010) Parallelization of Principal Component Analysis (Using Eigen Value Decomposition) on Scalable Multi-Core Architecture. 2010 *IEEE* 2*nd International Advance Computing Conference* (*IACC*), Patiala, 19-20 February 2010, 44-49. https://doi.org/10.1109/IADCC.2010.5423039

[12] Benzi, J. and Damodaran, M. (2009) Parallel Three Dimensional Direct Simulation Monte Carlo for Simulating Micro Flows. In: *Parallel Computational Fluid Dynamics* 2007, Springer, Berlin, 91-98. https://doi.org/10.1007/978-3-540-92744-0_11