

# 高精度壁画图像的实时浏览技术

葛 蓉 许端清 杨 鑫 刘建明

(浙江大学 人工智能研究所, 浙江 杭州 310027)

**内容摘要:** 数字化壁画图像可以广泛应用于保护修复、学术研究和旅游指导等方面,敦煌壁画数字化技术的引入将是永久无损保存敦煌艺术的最佳手段。作为壁画数字化技术的一个重要组成部分,高精度壁画的实时浏览给计算机交互显示带来了巨大的挑战。本文利用CUDA强大的并行计算能力,设计了一个基于动态分辨率的高精度壁画实时浏览算法。该算法不需要对大图像进行预处理,可以让用户很流畅地浏览不同分辨率下的高精度大图像。

**关键词:** 壁画浏览; 实时; 并行计算; JPEG压缩

**中图分类号:** J429.9 **文献标识码:** A **文章编号:** 1000-4106(2010)06-00098-06

## Real-time browsing of high-precision frescos

GE Rong XU Duanqing YANG Xin, LIU Jianming

(1. Artificial Intelligence Institute Zhejiang University, Hangzhou Zhejiang 310027;

2. Key Scientific Research Base of Comservation for Ancient Wall—paintings

State Admin Stration for Culture Heritage, Dunhuang Gansu 736200)

**Abstract:** Digitalized frescos are widely used in many areas, such as image impainting, art research, tourist guiding and so on. The introduction of digital image technology into Dunhuang frescos will bring the best way to permanently conserve Dunhuang art without destruction. As an important part of the technology, Real-time browsing of high-precision frescos brings us tremendous challenges. Taking advantages of parallel computing capabilities of CUDA, this paper presents an interactively image browsing algorithm based on dynamic resolution method. This algorithm avoids of burdensome pre-processing operations, while brings to users smooth viewing experience of large high-precision images.

**Keywords:** Fresco viewing; Real-time; Parallel computing; JPEG compression

### 1 引言

古代壁画艺术作品作为主要的文化遗产之

一,蕴含着大量的历史、文化、艺术信息,形象地记载了各个民族各个时代的社会风貌,具有重要的历史、科学和艺术价值。但是,由于古代壁画遗址往往地处荒漠,环境恶劣,经受着风、雨、雷、电、

收稿日期:2010-07-12

基金项目:古代壁画保护国家文物局重点科研基地开放课题资助项目—古代壁画数字化表达及表现技术研究(200803);新世纪优秀人才支持计划资助项目(NCET-04-0535)

作者简介:葛蓉(1986—),女,浙江省东阳市人,浙江大学计算机学院计算机应用技术专业硕士生。

火、地震、虫害、霉菌等多种因素的自然破坏,以及战争、偷盗等人为损坏,目前普遍存在严重的病害,亟待保护。传统的壁画保护手段虽然能够大幅度延缓壁画的退化,却无法阻止这个过程。壁画的数字化可以记录当前时刻壁画的状态,使之得以永久真实地保存,同时为壁画的研究提供准确详细的信息资料;并可制作虚拟洞窟供游客欣赏参观,为缓解石窟开放的压力、保护壁画提供技术保障,因此受到广泛的应用。然而,数字化获取的壁画图像精度越来越高,现有的图像浏览工具受限于内存,难以实现高精度壁画的快速、方便的浏览。以敦煌壁画为例,一副高精度壁画图像达到几GB,几十GB,甚至是几TB的大小,远远超出了PC机的内存容量和网络的带宽限制。另外为了保护版权,不允许使用者把整副图像全部下载到本地来浏览。由于以上两个原因,按需传输图像的部分区域<sup>[1]</sup>就显得非常有必要。当用户在客户端拖动、缩放图片时,服务器端实时传输相应分辨率下当前显示区域的部分图像。然而即便是每次只传输显示区域的部分图像,也要传输好几兆的数据,使得用户在大图像浏览时没法获得流畅的体验。为了解决这个问题,我们可以对图像进行压缩传输。JPEG是一种很好的图像压缩算法,它利用了人的视角系统的特性,使用量化和无损压缩编码相结合来去掉视角的冗余信息和数据本身的冗余信息<sup>[2]</sup>。JPEG压缩中的DCT变换和量化有很高的代码并行性,但是在CPU运算中没法有效地利用起来,导致运行速度慢,影响大图像的实时浏览。

随着计算机硬件技术的不断发展,尤其是多核技术的出现,使数据的并行处理能力有了巨大的提高。以往的大图像浏览系统都在单核CPU中进行运算,图像的预处理和数据传输的延迟严重影响了实时浏览的效果。而GPU强大的并行计算能力恰好给图像的实时浏览带来了可能。在过去的几年时间里,GPU的运算能力有了飞速的提高。2006年底,Ge80每秒钟的浮点运算数为520G,而2008年GT200的速度已经达到933G,差不多翻了一番。GPU的发展速度远远超过了CPU,2008年,NVIDIA的GT200的运算速度差不多是Intel Harpertown的6倍<sup>[3]</sup>。而且,GPU的应用也已经从原来单一的图形处理发展到了通用并行计算上。

本文中,我们给出了一个基于动态分辨率的大图像并行实时浏览系统。在此系统中,服务器端

维持当前显示区域的五个分辨率下的图像,当客户端用户缩放图像时,从该五个分辨率的图像中选择相应的一副传输到客户端。同时,重新调整生成缩放后的五个分辨率下的图像。另外,传输前先把图像压缩成JPEG格式,以使有限的带宽传输更多的数据。动态分辨率图像的生成和JPEG压缩中的DCT变化和量化都在GPU上并行实现以加快速度达到实时要求。

本文给出的算法可以有效地解决大图像实时浏览的问题。客户端用户可以很方便、流畅地对大图像进行缩放、拖动操作,从而欣赏不同分辨率下的图像。本文给出的算法避免了图像的预处理,加快了图像的压缩速度,减少了图像数据传输的延迟,从而达到了大图像实时浏览的目的。

## 2 相关工作

Tanner, C.C.等人提出的Clipmap<sup>[4]</sup>将底层硬件和高层系统软件结合,通过高速缓存机制调度当前显示区域的图像,有效地解决了在有限内存里的实时图形应用。但是这个系统是为单一图形工作站设计的,没法解决远程大图像浏览中与用户交互以及有效获取远程数据的问题。

Zhongding Jiang等人提出了一个面向多投影显示墙的大图像浏览实时交互系统<sup>[5]</sup>。该系统解决了用户远程数据获取和实时交互的问题,使得远程用户能够很方便地浏览大图像,包括图像的拖动和缩放操作。由于这个系统需要预先对大图像进行压缩和分块,实现图像的金字塔存储,所以要消耗大量的预处理时间和存储空间。

以前的大图像浏览系统都没有解决实时浏览的问题。这些大图像浏览系统都需要进行大量的预处理工作,耗费大量的时间和空间。

我们的大图像原始数据都是RGB格式的,这种格式的数据占用的空间很大,不适合网络传输。举个例子,在远程可视化中对于1000×1000分辨率的图像,为了达到25FPS需要100M/s的带宽<sup>[6-8]</sup>。即使是千兆位以太网在实际应用中也只有80M/s的速度,所以在传输前需要对图像进行压缩。JPEG是国际标准化组织(ISO)和CCITT联合制定的静态图像的压缩编码标准<sup>[9]</sup>,和相同图象质量的其它常用文件格式(如GIF,TIFF,PCX)相比,是目前静态图像中压缩比最高的。正是由于JPEG的高压缩比,

使得它广泛地应用于多媒体和网络程序中。本文中我们就采用JPEG格式对图像进行压缩。

CUDA是用于GPU计算的开发环境,它是一个全新的软硬件架构,可以将GPU视为一个并行数据计算的设备,对所进行的计算进行分配和管理。CUDA提供C语言编程接口,对程序启动数十上百个线程进行运算,相比CPU运算在速度上有很明显的优势。Pranit Patel等人使用CUDA实现JPEG压缩算法,发现与在CPU中做相同的运算相比可以提高61%的速度。文中我们将用CUDA来实现多分辨率图像的生成和JPEG压缩。

### 3 算法实现

本实验设定客户端的显示区域大小为 $1024 \times 768$ ,依据这个大小选取 $512 \times 512$ 的图像作为一个图像块(记为BLOCK),对原始图像以图像块为单位进行操作。

算法总体框架如下:先从原始图像中预读入 $20 \times 20$ 个BLOCK(图像块)到GPU的全局存储器(global memory),把这 $20 \times 20$ 个BLOCK(图像块)记为orgMap。orgMap包含当前显示的几个图像块以及其周围的一些图像块,它们是接下来介绍的多分辨率图像的数据来源。接着在服务器端建立五个分辨率下的图像,包括当前分辨率的图像(记为 $g_0$ )以及与之相邻的上下各两个分辨率的图像(分别记为 $g_{-2}$ , $g_{-1}$ 和 $g_1$ , $g_2$ ),相邻分辨率的图像长宽比都为 $2:1$ 。当客户端进行缩放或者拖动操作时,我们从该五个分辨率的图像中选择相应分辨率的一副图像,压缩成JPEG格式传输到客户端,同时服务器端重新调整生成这五个分辨率下的图像(详见3.4)。由于客户端采用网页形式浏览大图像,所以不需要解压缩,直接采用服务器端传输过来的JPEG格式的图像进行显示即可。下面我们将描述具体的实现细节。

#### 3.1 初始化

本实验设定客户端初始显示的图像为原始图像中间区域的图像。首先,从原始图像文件中读入中间的 $20 \times 20$ 个BLOCK(图像块),即 $10240 \times 10240$ 像素,到GPU的全局存储器(global memory),如果图像的宽度或高度小于10240,则读入整个的宽度或高度的图像。把这 $20 \times 20$ 个图像块记为orgMap。一方面要尽量一次性多读入数据来减少频繁读入

造成的大量的时间开销,另一方面读入的大小要受GPU中global memory(全局存储器)大小的限制,权衡之后我们选取读入 $20 \times 20$ 个BLOCK(图像块)。

从orgMap中读出中间的若干个BLOCK(图像块)生成初始显示的五个分辨率下的图像(多分辨率图像生成算法见3.4)。

#### 3.2 客户端交互操作过程

当客户端请求图像数据时,服务器端通过CUDA并行编程技术将 $g_0$ (当前分辨率的图像)快速压缩成JPEG格式(压缩算法见3.3),压缩后的图像记为 $g'$ ,通过TCP协议传输到客户端。由于该大图像浏览系统是基于网页形式的,JPEG正好符合网页图像的格式要求,所以客户端无需进行解压缩。为了更清楚地描述本方法,我们给出了交互过程中的一些细节处理:

拖动过程:如果拖动后客户端图像的显示区域在 $g'$ 范围内,则直接用 $g'$ 显示,只需客户端改变显示坐标位置属性即可,服务器端不需要任何操作;否则,服务器端需要重新生成多分辨率的图像(见3.4)。

放大(缩小)过程:首先压缩 $g_1(g_{-1})$ 图像(见3.3),压缩后的图像记为 $g'$ , $g'$ 通过TCP协议传输到客户端显示,就同时重新生成五个多分辨率的图像。

#### 3.3 基于CUDA的图像压缩

图像压缩的时候,JPEG编码器先将一幅原始图像转化为自己的色彩系统,按照人眼特点对其中各个色彩分量作不同的取样,经由DCT转换从时域变为频域,接着将变换后的数据量化以丢弃无用信息,然后用哈夫曼编码进行编码都到压缩数据,最后将色彩分量信息、量化表、编码表和各个色彩分量的压缩数据等混合成一个整体数据流,即形成JPEG文件。由于DCT变换和量化过程有很高的并行性,我们可以用CUDA并行编程实现。具体实现过程如下:

1) 颜色模式转换及采样。把RGB色彩系统转换为YCbCr色彩系统并采用YUV411方式采样。

2) 对采样后的各个色彩系统分别进行DCT变换。把图像分成多个 $8 \times 8$ 大小的图像块,对每个图像块执行2-D DCT变化。我们把2-D DCT转换成两个1-D DCT变换<sup>[8]</sup>(如图1),即先对所有列执行DCT变换然后对结果矩阵的所有行执行DCT变换。1-D

DCT公式如下：

$$X_k = \sum_{n=0}^7 x_n \cos \left[ \frac{\pi}{8} \left( n + \frac{1}{2} \right) k \right] \quad k=0, 1, 2, \dots, 7$$

为了加快运算速度,需要事先计算好每个cos值,保存为矩阵C。从公式中可以看出,1-D DCT变化实际上就是图像数据和C之间的矩阵乘法。矩阵乘法可以启用GPU的多线程进行并行运算。每个线程块分配64个线程处理一个 $8 \times 8$ 的图像块,每个线程负责计算结果矩阵中的一个元素。每个线程先从C矩阵的一行中读入一个数据到共享存储器(shared memory),然后从图像数据矩阵的一列中读入一个数据到共享存储器(shared memory),把这两个数相乘。循环8次,最后把得到的8个值相加。

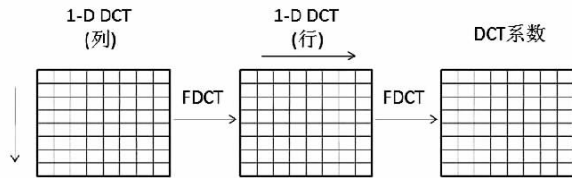


图1 2-D DCT 计算模型

3) 对图像进行量化。经过DCT变换之后得到的DCT系数矩阵最左上角的是直流成分,其他的都是交流成分,而且越到右下角频率越高,为了实现压缩我们需要用量化将高频的部分去掉。图2是对亮度信息进行量化的示意图,色度量化的表改成色度量化的表即可。GPU中线程的分配

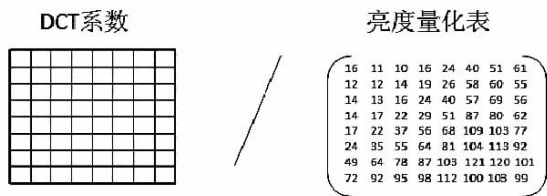


图2 亮度量化

同步步骤b,在步骤b的基础上每个线程读入量化表中相应位置的量化值,用b所得到的值除以该量化值,最后把得到的结果写会全局存储器(global memory)。

4) 对图像进行哈夫曼编码(Huffman Coding)。由于这部分代码的并行度不高,所以我们在CPU中进行该部分的运算。

### 3.4 基于CUDA的多分辨率图像的生成

当客户端用户进行缩放操作的时候,服务器

端需要传输不同分辨率下的图像到客户端进行显示,从而每次都需要从原始图像数据中读出相应的图像块进行缩小操作,严重影响了大图像的实时交互性。动态多分辨率图像的生成正好可以解决这个问题。我们在服务器端维持五个分辨率下的图像,当客户端用户进行缩放操作的时候,服务器端可以选择相应分辨率下的图像进行压缩后传输到客户端进行显示。

orgMap是预存在全局存储器(global memory)中的图像数据。在初始化过程中已经从原始图像中载入了部分图像数据,此后每次用户拖动的过程中,如果客户端显示的图像区域超出了orgMap的范围,则从原始图像数据中读入部分新的数据更新orgMap,每次让其读入远超过超出范围的数据,以减少读入次数。

我们在服务器端维持五个分辨率下的图像:  $g_0$ (当前分辨率的图像),  $g_2$ (比 $g_0$ 大两个分辨率的图像),  $g_1$ (比 $g_0$ 大一个分辨率的图像),  $g_{-1}$ (比 $g_0$ 小一个分辨率的图像)和  $g_{-2}$ (比 $g_0$ 小两个分辨率的图像)。设定各分辨率下的图像均稍大于客户端显示区域。本实验中由于设定了客户端显示区域为 $1024 \times 768$ ,所以可以选择各个分辨率下的图像大小为 $3 \times 2$ 个BLOCK(图像块)。初始化时,我们从orgMap中直接读取 $3 \times 2$ 个BLOCK得到 $g_2$ ,读取 $6 \times 4$ 个BLOCK缩小一半得到 $g_1$ ,依次类推得到 $g_0$ 、 $g_{-1}$ 和 $g_{-2}$ 。缩小操作时,我们将现在的 $g_1$ 、 $g_0$ 、 $g_{-1}$ 、 $g_{-2}$ 分别赋给新的 $g_2$ 、 $g_1$ 、 $g_0$ 、 $g_{-1}$ ,然后从orgMap中读取数据按相应分辨率缩小生成 $g_{-2}$ ;放大操作时,我们将现在的 $g_2$ 、 $g_1$ 、 $g_0$ 、 $g_{-1}$ 分别赋给新的 $g_1$ 、 $g_0$ 、 $g_{-1}$ 、 $g_{-2}$ ,然后从orgMap中读取数据按相应分辨率缩小生成 $g_2$ ;拖动操作时全部从orgMap中读取新的数据生成 $g_2$ 、 $g_1$ 、 $g_0$ 、 $g_{-1}$ 、 $g_{-2}$ 。假设orgMap和 $g_2$ (对于 $g_1$ 、 $g_0$ 、 $g_{-1}$ 和 $g_{-2}$ 也采用同样的操作)相差 $n$ 个分辨率,则我们从orgMap(全局存储器中预存的图像数据)中读取包含当前显示区域的 $(3 \times n) \times (2 \times n)$ 个BLOCK的图像(即图中orgMap中间的矩形区域),长度和宽度分别缩小 $n$ 倍得到 $g_2$ 。缩小过程用CUDA多线程并行编程实现。假设缩小后的图像分辨率为 $w \times h$ (这里 $w \times h$ 是BLOCK的整数倍),则我们分配维度分别为 $w/16$ 、 $h/16$ 的数目的线程块。我们给每个线程块分配256( $16 \times 16$ )个线程,每个线程处理缩小后图像的一个像素数据的值。线程块中的每个线程从orgMap中读取窗口大小为 $n \times n$ 的

Global memory 预存图像数据 orgMap

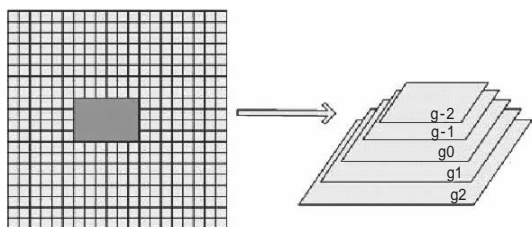


图3 动态多分辨率图像生成算法

图像数据, 计算这 $n \times n$ 个数据的平均值得到结果像素的数据值。

## 4 实验结果

我们将一台配有一颗 Intel Core 2 Duo T6600 2.2GHZ 的 CPU 和一颗 Nvidia GTX285 (1G 显存) 的 PC 机作为服务器进行实验。首先对 CPU 和 GPU

1024*1024	CPU(ms)	CUDA(ms)
DCT	12.03	0.44
Quant	18.17	0.15
Total	30.20	0.59
Speedup	51.19	

图4 CPU 和 GPU 实现 JPEG 压缩  
1024 × 1024 大小的图像时间

2048*2048	CPU(ms)	CUDA(ms)
DCT	58.24	2.12
Quant	65.22	0.21
Total	123.46	2.33
Speedup	52.98	

图5 CPU 和 GPU 实现 JPEG 压缩  
2048 × 2048 大小的图像时间

实现图像 JPEG 压缩算法的效率进行对比, 然后实现基于 CUDA 的高精度壁画实时浏览算法。

对于 CPU 和 GPU 实现的 JPEG 压缩, 我们仅对 DCT 和量化两个阶段做对比。我们用 1024 × 1024 和 2048 × 2048 分辨率的两幅图像做实验, 得到的结果数据如图 4 和图 5 所示。

对于大图像浏览, 我们选取敦煌莫高窟 66 号洞窟拍摄的五台山图作为实验图像, 该图像分辨率为 45000 × 25000, 图 9 所示为中间某一分辨率下的浏览图。实验设定浏览操作的步长为 50 个像素, 缩放

图像浏览拖动操作

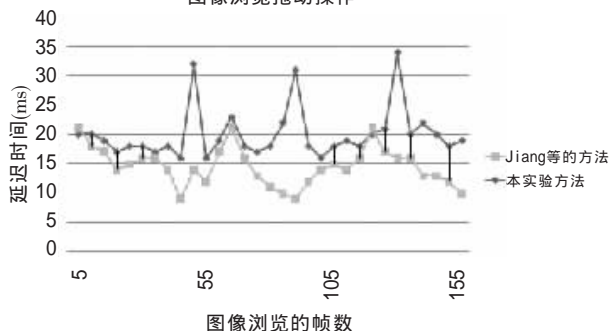


图6 图像拖动操作的时间延迟

图像浏览放大缩小操作

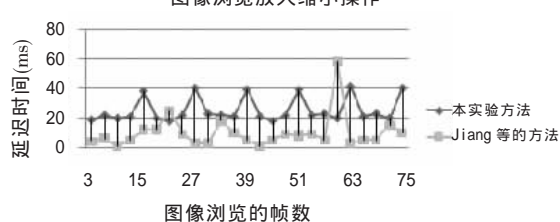


图7 图像缩放操作的时间延迟

图像大小 (BLOCK)	IO 时间 (秒)
16	0.48
32	0.96
64	1.93

图8 图像从 CPU 传输到 GPU 的 IO 时间



图9 五台山大图像浏览

层数为 50。我们与 Jiang 等的算法进行比较, 结果如图 6 和图 7 所示, 我们在图像拖动和缩放操作上略逊于他们的算法。在 CPU 和 GPU 的数据传输方面也有点耗时, 如图 8 所示。但是与他们的算法相比, 我们少了预处理过程, 可以少去很多预处理的时间和存储空间开销。我们的大图像浏览不仅少了预处理同时也获得了很好的浏览效果。

## 5 结论与展望

本文,我们给出了一个大图像实时浏览的系统模型,可以让网络客户端用户很流畅地浏览不同分辨率下的大图像。依赖于CUDA的并行运算能力,我们设计的算法不需要大量的预处理,又可以达到很好的实时浏览效果。这对于古代壁画的保护、欣赏和研究都具有很重要的意义。在以后的工作中,我们还需要对并行算法做进一步的优化,以便更好地提升用户体验。另外,我们将继续优化多分辨率图像的存储结构,改进动态调度不同分辨率图像的方法。当前我们的方法消耗了过多的带宽,由于Nvidia公司最新推出的CUDA 2.2版本中支持在GPU里直接操纵CPU内的数据<sup>[10]</sup>,因此我们可以省去数据从CPU到GPU的导入过程,这就大大加快了算法的处理效率。同时,我们也希望我们的算法能在Inter公司所设计的Larrabee处理器<sup>[11]</sup>中获得出色的性能。

---

### 参考文献:

- [1] Krishnaprasad, N., Vishwanath, V., Venkataraman, S., Rao, A., Renambot, L., Leigh, J., Johnson, A.: Juxta View – A Tool for Interactive Visualization of Large Imagery on Scalable Tiled Display. In: Proceedings of IEEE Cluster 2004:411-420(2004).
- [2] 张春田, 苏育挺, 张静. 数字图像压缩编码 [M]. 北京: 清华大学出版社, 2006.
- [3] NVIDIA Corporation, CUDA Programming Guide 2.0, <http://www.nvidia.com>, 2008.
- [4] Christopher C.Tanner, Christopher J.Migdal, Michael T. Jones. The Climap: A Virtual Mipmap [A]. Michael Cohen, Proceeding of SIGGRAPH 98[C]. ACM SIGGRAPH. Addison Wesley, 1998:151.
- [5] Zhongding Jiang, Xuan Luo, Yandong Mao, Binyu Zang, Hai Lin, and Hujun Bao. Interactive Browsing of Large Images on Multi-projector Display Wall System, Beijing, China, 2007:827-836.
- [6] Stefan Lietsch, Oliver Marquardt. A CUDA-Supported Approach to Remote Rendering. ISVC 2007, Part I, LNCS4811: 724-733, 2007.
- [7] Pranit Patel, Jeff Wong, Manisha Tatikonda, and Jarek Marczewski. JPEG Compression Algorithm Using CUDA. [R] Course Project for ECE 1724. Department of Computer Engineering University of Toronto, (2009)
- [8] Leiwen Wu, Mark Storus, David Cross. CUDA Compression Project for CS315A. Find Project Stanford University, 2009.
- [9] Gregory K. Wallace. The JPEG Still Picture Compression Standard [J]. IEEE Transactions on Consumer Electronics, 1992, Vol.38. No.1.
- [10] <http://forums.nvidia.com/index.phpshowtopic=97333> [EB], Nvidia.
- [11] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugarman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A many-core x86 architecture for visual computing [A]. ACM Transactions on Graphics. ACM Transactions on Graphics [J], 2008: Volume 27, No.3, Article 18.