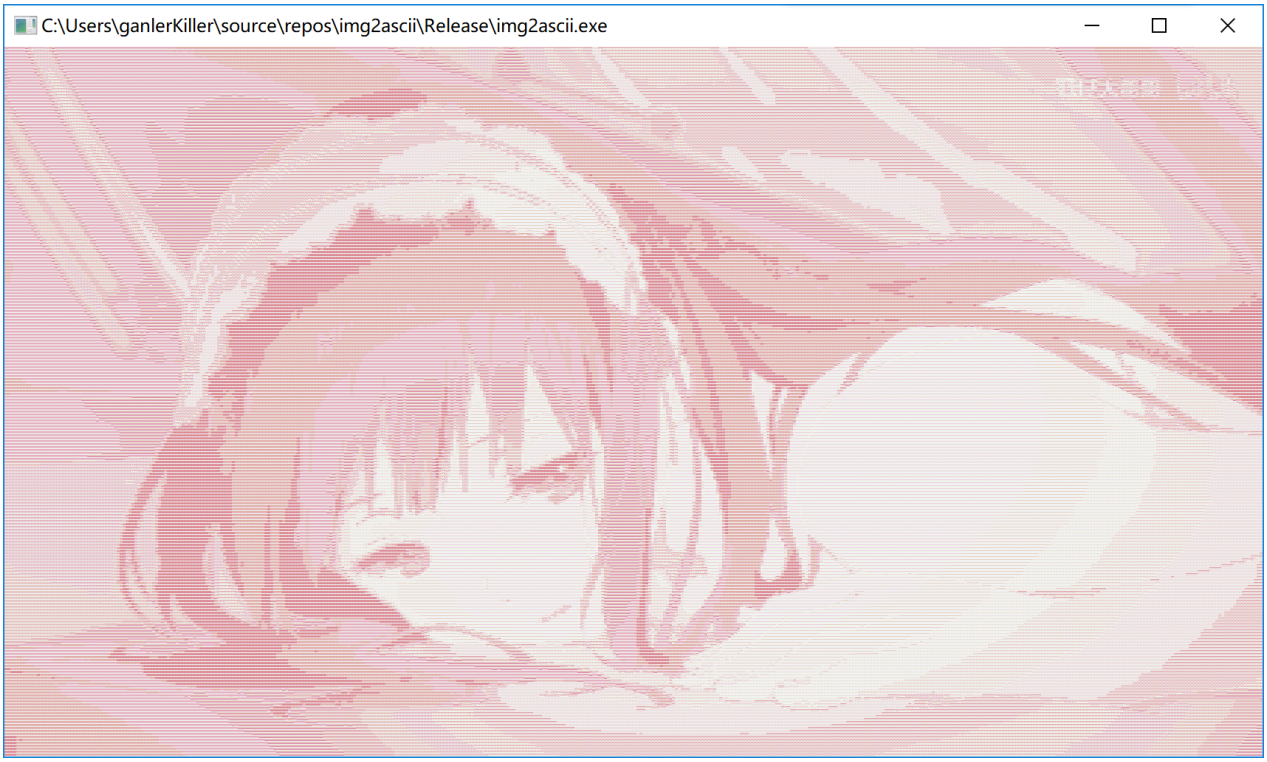
高程大作业：IMG2ASCII



专业/班级：计算机科学与技术 - 计算机三班

学号：1753070

姓名：刘佳伟

完成日期：2019/6/2

特点速览

> **注意：**
>
> 限于文件大小，最后在网站上上提交的只是一个简单的demo，最终成品建议参考下面链接的视频。

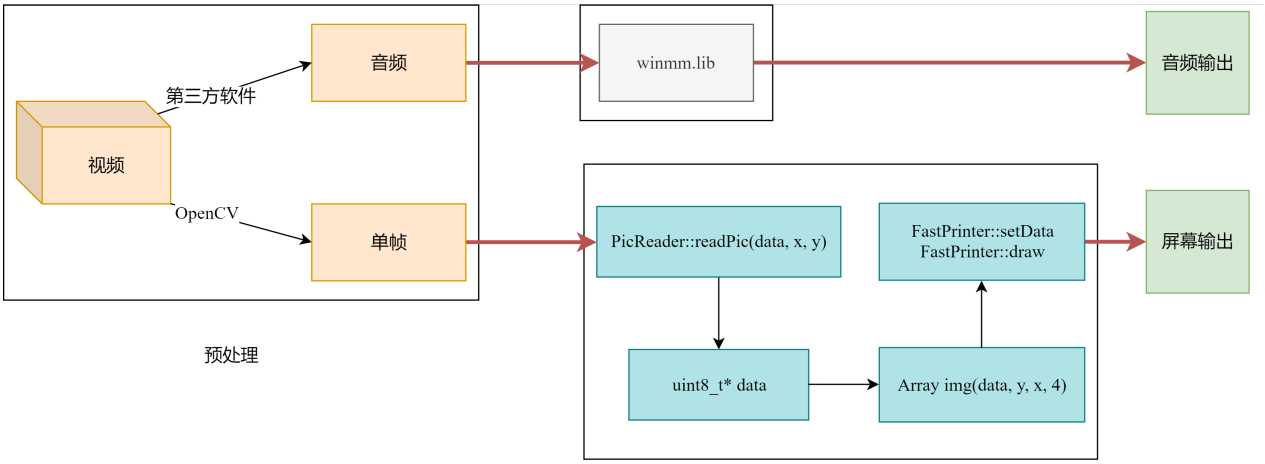
完整视频的bilibili链接

- 高性能实现：
  - Array类实现了移动语义，直接移动右值而不是拷贝；
  - 考虑图像的遍历需要较多时间，直接在一次遍历中先转灰度图转ascii码；
  - 关闭异常功能；
  - 高性能rgb转灰度：
    - 内存复用；
    - 全称只有整数加法、乘法和位移（整数除法和浮点数运算比这些运算消耗更多的指令周期）；
    - 使用SIMD指令支持（为了兼容性使用了SSE），同时使用16字节内存对其以增加数据吞吐量；
- 彩色字符切换；
- Music ! ! !
- 精简API，从粗粒度到细粒度均有设计，核心调用一句话 $img.to\_ascii()$。

设计思路和功能描述

设计思路

一图胜过千言。



Kurt Guntheroth在他写的*Optimized C++* 中的第一章多次重复“内存是很慢的，尤其是动态内存”，动态内存的申请要有可能要调用系统函数（调用内核方法是很耗时的），多次这样的操作会导致性能降低。所以不应该重复的new和delete，应该一次申请干净，之后都使用该块内存其中笔者设计中的data对应的内存只在最开始处（由我设计的array类在构造函数的时候申请）动态申请一次，之后所有操作都只复用该块内存，直到array对象被析构的时候才被释放，没错这就是c++之父提倡的伟大而简单的RAII（Resource Acquisition Is Initialization）技术，即在构造的时候获取所需要的资源，并在析构的时候将其释放；

其中用OpenCV分解视频很简单：

```
#include <opencv2/opencv.hpp>
#include <string>

using namespace std;
using namespace cv;

int main()
{
    VideoCapture cap("my_video.flv");
    Mat frame;
    int cnt = 0;
    while(cap.isOpened())
    {
        cap >> frame;
        if(frame.empty())
            break;
        imwrite("imsrc/"+to_string(cnt)+".png", frame);
    }
}
```

array类的设计与功能描述

设计思想与使用的技术

- RAII；
- 五法则(因为用户定义析构函数、复制构造函数或复制赋值运算符的存在阻止移动构造函数和移动赋值运算符的隐式定义，所以任何想要移动语义的类必须声明全部五个特殊成员函数)；
- 实现了无异常(`noexcept`)的移动语义，能够消除对于右值（可简单的理解为“临时产生，即将丢弃的变量”）的拷贝，同时无异常的设计可以让其在强类型安全的数据结构（如`std::vector`）的在进行类似于`push_back`这样的操作的时候激活移动语义，极大的提高性能。（当然这样的设计在本项目上可能帮助难以看出，但是放置到长远使用可以极大的提高效率）；
- 接口上模仿了STL Container的常见接口，让用户使用更简单；
- 模板设计，可适用于多种需求的数据结构（对于本次大作业使用的uint8_t结构，笔者将`array<uint_8>`设置了一个`Array`的别名，即`using Array = array<uint8_t>`）；
- 多种构造函数，满足多种用户需求；

接口声明与功能简介

> 我给我的代码做了一下美颜，嘿嘿嘿。

- 最多三维索引（可当1~3维数组使用）；
- 支持加减乘除操作（对于长长的表达式可以用移动语义消灭临时变量的拷贝）；
- 支持操纵原始指针；
- 考虑了常对象的实现；
- reshape操作；
- 直接用`to_ascii`转字符（同时内存复用）；

```cpp
#pragma once

#include <iostream>
#include <cstdint>

template <typename _Tp = int> class array
{
private:
    using value_type       = _Tp;
    value_type* m_data     = nullptr;
    std::size_t m_cols     = 0;
    std::size_t m_rows     = 1;
    std::size_t m_channels = 1;
private:
    void clear() noexcept;
public:
    array(value_type*, std::size_t, std::size_t, std::size_t = 1);
    array(std::size_t, std::size_t, std::size_t = 1);
    array(const array&);
    array(array&&)          noexcept;
    array& operator=(array) noexcept;
    ~array() noexcept;
public:
    value_type*&        data();
    const  value_type*& data()     const;
    inline value_type&  at     (std::size_t, std::size_t, std::size_t = 1);
    inline value_type   at     (std::size_t, std::size_t, std::size_t = 1) const;
    inline void         reshape(std::size_t, std::size_t, std::size_t = 1);
    inline std::size_t  size()     const noexcept;
    inline std::size_t  rows()     const noexcept;
    inline std::size_t  cols()     const noexcept;
    inline std::size_t  channels() const noexcept;
public:
    void   to_ascii()               noexcept;
    array  operator + (array) const noexcept;
    array  operator - (array) const noexcept;
    array  operator * (array) const noexcept;
    array  operator / (array) const noexcept;
    friend std::ostream& operator << (std::ostream&, const array&);
    friend std::istream& operator >> (std::istream&, array&);
};

using Array = array<uint8_t>;
```

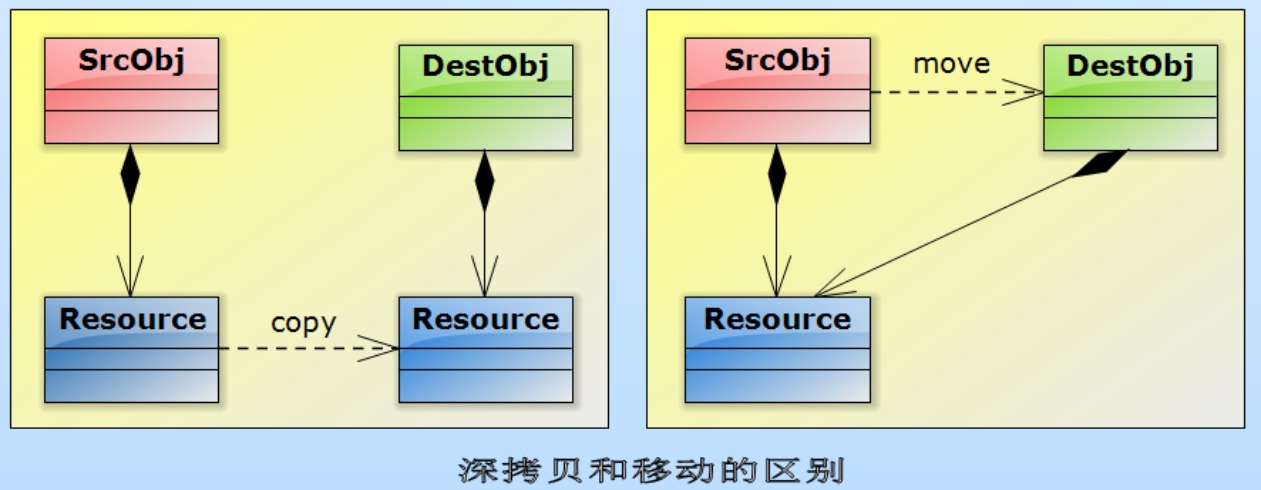
> 接下来讲一下核心设计思路，因为笔者的长处是x86优化，所以也会聊聊如何提高x86的程序性能。

移动语义与五法则构建类的核心函数

C++ 11 引入了两个新的特殊成员函数：移动构造函数和移动赋值运算符。由于你想要遵循 C++ 03 中的 Rule of Three 的所有相同的原因，你通常想要遵循 C++ 11 中的 Five of Rule：如果一个类需要五个特殊成员函数中的一个，并且如果移动语义如果需要，那么它很可能需要五个。

但是，请注意，未遵循五法则通常不会被视为错误，而是错过优化机会，只要仍然遵循三法则。如果编译器通常不使用移动构造函数或移动赋值运算符，则它将在可能的情况下使用复制语义，从而导致由于不必要的复制操作而导致操作效率降低。如果类不需要移动语义，则不需要声明移动构造函数或赋值运算符。

移动语义的思想很简单，比如对于我的array类我做了一个计算：`auto out = arr1 * arr2 * arr3`，如果不用移动语义，那么就是先计算产生个`arr_tmp1 = arr2 * arr3`的临时变量，然后再计算`out = arr1 * arr_tmp1`，不用移动语义的后果就是这里的`arr_tmp1`成为了一个未被利用的临时变量，从而进行了一次没有意义的拷贝，如果我们可以把`arr_tmp1`对象直接拿来作计算，而不拷贝的话，那么久可以节省拷贝操作，特别对于array这种大型对象，充分利用临时变量可以减少大量有关动态内存的操作，以提高巨大的性能。



> 简单来说，移动语义直接转移临时变量的资源，而不是愚蠢的拷贝一遍。对于大型对象有关的操作，可以大幅度提升性能。比如我们计算$out = \sum_{i=0}^{n} arr_i$这个式子，其中有n个临时变量，使用移动语义可以节约n次拷贝。

RAII思想与实现

RAII是Resource Acquisition Is Initialization（wiki上面翻译成“资源获取就是初始化”）的简称，是C++语言的一种管理资源、避免泄漏的惯用法。利用的就是C++构造的对象最终会被销毁的原则。RAII的做法是使用一个对象，在其构造时获取对应的资源，在对象生命期内控制对资源的访问，使之始终保持有效，最后在对象析构的时候，释放构造时获取的资源。

以我们的array类为例子，我们构造一个array类对象的时候，在构造函数中申请好我们要的资源（申请动态内存），在析构函数清理这些资源（释放动态内存）。这种技术可以帮助有效的组织类的逻辑结构。也能避免内存泄漏。

具体到我的实现：

```
// array.hpp - 构造函数

array(std::size_t r, std::size_t c, std::size_t cn = 1)
  : m_cols(c), m_rows(r), m_channels(cn), m_data(new value_type[c*r*cn]) {}
// 构造时new value_type[c*r*cn];


// array.hpp - 析构函数
```

```
void clear() noexcept
{
  if (m_data != nullptr)
    delete[] m_data;
}
~array() noexcept { clear(); }
// 析构时delete[] m_data;
```

img2ascii优化实践

灰度转化公式：

$$GRAY = 0.299 * R + 0.587 * G + 0.114 * B$$

1. **优化1：单指令优化：**

浮点数计算很慢的，而且很耗电...就算是高程网站的那个公式，整数计算的除法也慢：

> 参考一下CSAPP中提供的Intel Core i7 Haswell的架构。

5.7.2 功能单元的性能

图 5-12 提供了 Intel Core i7 Haswell 参考机的一些算术运算的性能，有的是测量出来的，有的是引用 Intel 的文献[49]。这些时间对于其他处理器来说也是具有代表性的。每个运算都是由以下这些数值来刻画的：一个是延迟(latency)，它表示完成运算所需要的总时间；另一个是发射时间(issue time)，它表示两个连续的同类型的运算之间需要的最小时钟周期数；还有一个是容量(capacity)，它表示能够执行该运算的功能单元的数量。

运算	整数			浮点数		
	延迟	发射	容量	延迟	发射	容量
加法	1	1	4	3	1	1
乘法	3	1	1	5	1	2
除法	3~30	3~30	1	3~15	3~15	1

图 5-12 参考机的操作的延迟、发射时间和容量特性。延迟表明执行实际运算所需要的时钟周期总数，而发射时间表明两次运算之间间隔的最小周期数。容量表明同时能发射多少个这样的操作。除法需要的时间依赖于数据值

那么我希望只使用整数运算(快，并且省电)，而且不用到整数除法。因为上面的数字小数位数不多，所以我们可以将其转为大整数：

```
constexpr uint16_t SCALE_SZ = 15;
constexpr uint16_t r_ = static_cast<uint16_t>(0.299 * (1 << SCALE_SZ) + 0.5);
constexpr uint16_t g_ = static_cast<uint16_t>(0.587 * (1 << SCALE_SZ) + 0.5);
constexpr uint16_t b_ = static_cast<uint16_t>(0.114 * (1 << SCALE_SZ) + 0.5);
```

`constexpr`代表的是常量表达式，这些结果都是编译期间算好的。这样我们得到了一个对应原来浮点数的整数参数。浮点数计算的规约是四舍五入，而整数是向下取整，所以我加了一个0.5让其进行“四舍五入”。

然后在进行运算的时候：

```
auto answer = static_cast<uint8_t>(
  (r_ * src[base] + g_ * src[base + 1] + b_ * src[base + 2] + (1 << (SCALE_SZ
- 1))
) >> SCALE_SZ);
```

这样就算完了，一般的操作是：

- 整数 to 浮点数
- 浮点乘法
- 浮点加法
- 浮点数 to 整数

我的操作是：

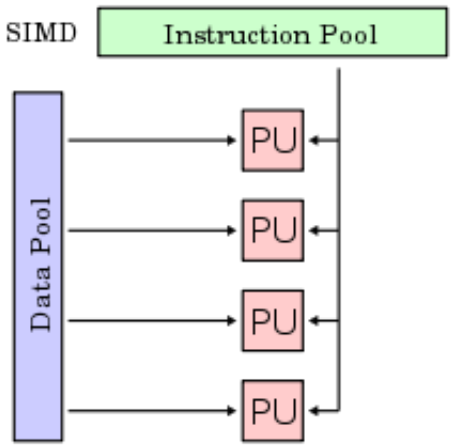
- 整数乘法
- 整数加法
- 整数位移

操作更少且更高效。

2. **SIMD单指令多数据流：**

这里的操作都是一样的，且位置不同的元素相互独立，那么我们可以用SIMD指令进行并行运算，因为这里的操作非常简单，只要加上(在gcc和clang这种编译器)`-ftree-vectorize -march=native`编译条件即可自动向量化，而在vs中也简单，只要手动在工程设置处添加对应指令集即可(为了兼容性，我在我提交的项目中用了SSE 128位并行，更厉害的指令有AVX何AVX512等，但不是每个机器都有)。

我们一个byte就8位，如果我们用128位的寄存器并行的话，那么我们就可以做到一次算128/8=16个数字。



除此之外，使用128位的寄存器还需要使用16字节对齐，以保证cache line能被一次读入。

3. **内存复用：**

内存复用的原理很简单，就是对于一块内存，如果它暂时不会被使用，我就先拿过来用着。

这里我们在程序刚开始申请了一块$size = rows \times cols \times channels$的动态内存，我在读入图片的时候用这块内存，我转灰度的时候，依旧用这块内存。这样只有一次动态内存申请，一次动态内存释放，当然会比重复申请内存的情况要好。

4. **关闭异常功能：**

由于我们的这次作业用不到异常，所以我们可以直接把异常关了。关掉异常是能提速的，启动异常需要额外的栈空间。具体实现是在函数声明处加上`noexcept`关键词。在vs中还可以设置项目选项以关闭异常。

在实验中遇到的问题与解决方法

字符不是正方形

最开始在输出字符的时候，发现输出字符图像的形状很畸形(长宽比很起怪，偏瘦)，所以我在转ascii图像的时候，将宽度给直接翻倍了，这样后就可以显示正常了。

心得体会

本次实验中我基于c++14的标准，使用了RAII等技术书写了array类。同时对一些关键函数进行了重度的人工优化：

- Array类实现了移动语义，直接移动右值而不是拷贝；
- 考虑图像的遍历需要较多时间，直接在一次遍历中先转灰度图转ascii码；
- 关闭异常功能；
- 高性能rgb转灰度：
    - 内存复用；
    - 全称只有整数加法、乘法和位移（整数除法和浮点数运算比这些运算消耗更多的指令周期）；
    - 使用SIMD指令支持（为了兼容性使用了SSE），同时使用16字节内存对其以增加数据吞吐量；

同时在API的设计上设计的非常精简，在`demo.cpp`上，我主要就用到了一个`to_ascii()`函数完成了所需要的功能。

附录1：源代码

```
#include "PicReader.hpp"
#include "FastPrinter.hpp"
#include <stdio.h>
#include <iostream>
#include <string>
#include "array.hpp"

#include<mmsystem.h>
```

```
#pragma comment(lib,"winmm.lib")


void imshow(const char* picname, int pix_sz = 1)
{
  PicReader imread;
  BYTE *data = nullptr;
  UINT x, y;

  // Pure black
  imread.readPic(picname);
  imread.testReader(data, x, y);
  pix_sz = (x <= 75) ? 2 : pix_sz;

  Array img(data, y, x, 4);
  constexpr size_t tms = 2;
  img.to_ascii();

  WORD* colorBuffer = new WORD[img.size() / 2];
  FastPrinter printer(2 * img.cols(), img.rows(), pix_sz);
  SMALL_RECT drawArea{ 0, 0, img.cols() * 2, img.rows() };

  for (int i = 0; i < tms * img.size() / 4; i++)
    colorBuffer[i] = fp_color::f_black | fp_color::b_l_white;

  printer.cleanSrceen();
  printer.setData((char*)img.data(), colorBuffer, drawArea);
  printer.draw(true);

  delete[] colorBuffer;
  printf("Press enter to continue...");
  getchar();
}

void play_demo()
{

  PicReader imread;
  BYTE *data = nullptr;
  UINT x, y;

  // Pure black
  imread.readPic("imsrc\\1.jpg");
  imread.testReader(data, x, y);

  Array img(data, y, x, 4);
  constexpr size_t tms = 2;
  img.to_ascii();
```

```
  WORD* colorBuffer = new WORD[img.size() / 2];
  FastPrinter printer(2 * img.cols(), img.rows(), 2);
  SMALL_RECT drawArea{ 0, 0, img.cols() * 2, img.rows() };
  Sleep(500);
  mciSendString("open demo.mp3 alias aa", NULL, 0, NULL);//alias后面为设备名称
  mciSendString(TEXT("play aa"), NULL, 0, NULL);

  constexpr uint8_t colors[] = { fp_color::f_red, fp_color::f_blue,
fp_color::f_black, fp_color::f_purple, fp_color::f_gray, fp_color::f_green };
  uint8_t fc = 0;
  for (int i = 1, cnt = 0; i <= 155; i++) {

    imread.readPic(("imsrc\\" + std::to_string(i) + ".jpg").c_str());
    imread.testReader(std::ref(img.data()), x, y);
    img.to_ascii();

    if (++cnt % 100 == 0)
      fc = colors[(cnt / 100) % 6];

    for (int i = 0; i < tms * img.size() / 4; i++)
      colorBuffer[i] = fc | fp_color::b_l_white;
    Sleep(35);
    printer.cleanSrceen();
    printer.setData((char*)img.data(), colorBuffer, drawArea);
    printer.draw(true);
  }

  delete[] colorBuffer;
  mciSendString("play aa wait", NULL, 0, NULL);
  mciSendString("close aa", NULL, 0, NULL);
  printf("Press enter to continue...");
  getchar();
}

int main() {

  {// Images test.
    imshow("classic_picture/airplane.jpg");
    imshow("classic_picture/baboon.jpg");
    imshow("classic_picture/barbara.jpg");
    imshow("classic_picture/cameraman.jpg");
    imshow("classic_picture/compa.png");
    imshow("classic_picture/lena.jpg");
    imshow("classic_picture/lena1.jpg");
    imshow("classic_picture/milkdrop.jpg");
    imshow("classic_picture/peppers.jpg");
    imshow("classic_picture/woman.jpg");
  }
```

```
  {// Video test.
    play_demo();
  }
}
```

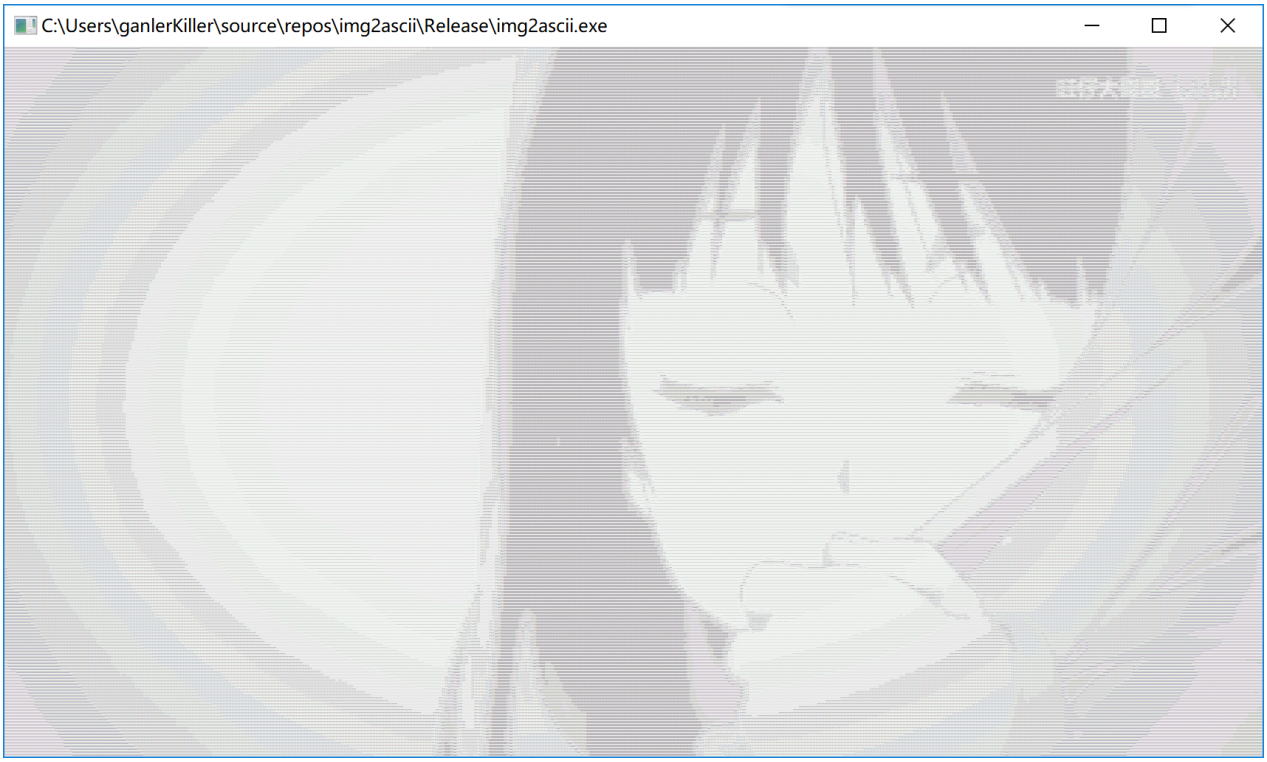
```
// ARRAY.HPP

#pragma once

#include <iostream>
#include <cstdint>

template <typename _Tp = int> class array
{
private:
  using value_type = _Tp;
  value_type* m_data = nullptr; // [m_channels] -> m_rows -> m_cols
  std::size_t m_cols = 0;
  std::size_t m_rows = 1;
  std::size_t m_channels = 1;
private:
  void clear() noexcept
  {
    if (m_data != nullptr)
      delete[] m_data;
  }
public:
  array(value_type* src, std::size_t r, std::size_t c, std::size_t cn = 1)
    : m_data(src), m_cols(c), m_rows(r), m_channels(cn) {}
  array(std::size_t r, std::size_t c, std::size_t cn = 1)
    : m_cols(c), m_rows(r), m_channels(cn), m_data(new value_type[c*r*cn]) {}
  ~array() noexcept { clear(); }
  array(const array& arr)
  {
    clear();
    if (arr.m_data != nullptr)
    {
      m_data = new value_type[arr.size()];
      m_cols = arr.m_cols;
      m_rows = arr.m_rows;
      m_channels = arr.m_channels;
      std::memcpy(m_data, arr.m_data, sizeof(value_type) * arr.size());
    }
  }
  array(array&& arr) noexcept
  {
    clear();
    if (arr.m_data != nullptr)
    {
```

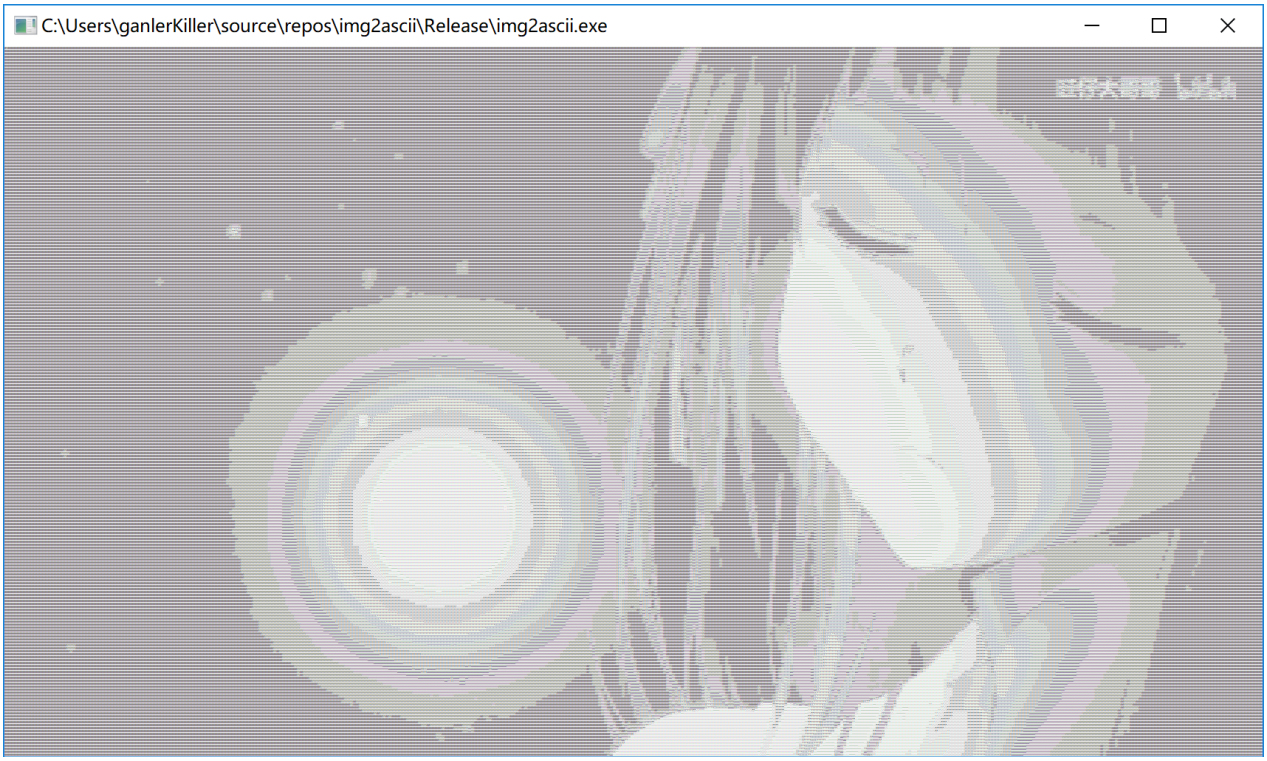
```
      std::swap(arr.m_data, m_data);
      m_cols = arr.m_cols;
      m_rows = arr.m_rows;
      m_channels = arr.m_channels;
    }
  }
  array& operator=(array arr) noexcept
  {
    std::swap(m_data, arr.m_data);
    m_cols = arr.m_cols;
    m_rows = arr.m_rows;
    m_channels = arr.m_channels;
    return *this;
  }
public:
  inline value_type& at(std::size_t r, std::size_t c, std::size_t cn = 1)
  {
    return m_data[m_channels*m_cols*r + m_channels * c + cn];
  }
  inline value_type at(std::size_t r, std::size_t c, std::size_t cn = 1) const
  {
    return m_data[m_channels*m_cols*r + m_channels * c + cn];
  }
  inline std::size_t size() const noexcept
  {
    return m_cols * m_rows*m_channels;
  }
  inline std::size_t rows() const noexcept
  {
    return m_rows;
  }
  inline std::size_t cols() const noexcept
  {
    return m_cols;
  }
  inline std::size_t channels() const noexcept
  {
    return m_channels;
  }
  void reshape(std::size_t r, std::size_t c, std::size_t cn = 1)
  {
    m_rows = r;
    m_cols = c;
    m_channels = cn;
  }
public:
  array operator + (array rhs) const noexcept
  {
    for (std::size_t i = 0; i < size(); ++i)
```

```
      rhs.m_data[i] += m_data[i];
    return rhs;
  }
  array operator - (array rhs) const noexcept
  {
    for (std::size_t i = 0; i < size(); ++i)
      rhs.m_data[i] -= m_data[i];
    return rhs;
  }
  array operator * (array rhs) const noexcept
  {
    for (std::size_t i = 0; i < size(); ++i)
      rhs.m_data[i] *= m_data[i];
    return rhs;
  }
  array operator / (array rhs) const noexcept
  {
    for (std::size_t i = 0; i < size(); ++i)
      rhs.m_data[i] /= m_data[i];
    return rhs;
  }
  value_type*& data()
  {
    return m_data;
  }
  const value_type*& data() const
  {
    return m_data;
  }
  decltype(auto) to_ascii() noexcept
  {
    constexpr size_t tms = 2;
    constexpr uint16_t SCALE_SZ = 15;
    constexpr uint16_t r_ = static_cast<uint16_t>(0.299 * (1 << SCALE_SZ) +
0.5);
    constexpr uint16_t g_ = static_cast<uint16_t>(0.587 * (1 << SCALE_SZ) +
0.5);
    constexpr uint16_t b_ = static_cast<uint16_t>(0.114 * (1 << SCALE_SZ) +
0.5);
    constexpr int8_t ascii_code[] =
{'#','M','$','N','$','%','?','>','+','=','!',';',':','-',',', '.'};
    auto src = m_data;

    for (int i = 0; i < m_cols * m_rows; ++i)
    {
      std::size_t base = i * channels();
      src[i*tms] = ascii_code[static_cast<uint8_t>(
        (r_ * src[base] + g_ * src[base + 1] + b_ * src[base + 2] + (1 <<
(SCALE_SZ - 1))) >> SCALE_SZ
```

```
        ) >> 4];
      src[i*tms + 1] = src[i*tms];
    }
  }
  friend std::ostream& operator << (std::ostream& os, const array& arr)
  {
    const auto line_width = arr.m_cols * arr.m_channels;
    for (int i = 0; i < arr.size(); ++i)
    {
      if (i > 0 && i % line_width == 0)
        os << '\n';
      os << arr.data()[i] << ' ';
    }
    return os << '\n';
  }
  friend std::istream& operator >> (std::istream& is, array& arr)
  {
    for (int i = 0; i < arr.size(); ++i)
      is >> arr.data()[i];
    return is;
  }
};

using Array = array<uint8_t>;
```

附录2：效果展示




C:\Users\ganlerKiller\source\repos\img2ascii\Release\img2ascii.exe