内容回顾

- **递归函数：**

  函数在其函数体中直接或间接地调用了自己。

- **直接递归**

```
void f()
{   .......
    ... f() ...
    .......
}
```

- **间接递归**

```
extern void g();
void f()
{   .......
    ... g() ...
    .......
}
void g()
{   ......
    ... f() ...
    ......
}
```

递归条件和结束条件

- 在定义递归函数时，一定要对两种情况给出描述：
  - 递归条件：指出何时进行递归调用，它描述了问题求解的一般情况，包括：分解和综合过程。
  - 结束条件：指出何时不需递归调用，它描述了问题求解的特殊情况或基本情况。

```
int f(int n)
{
    if (n == 0)
        return 1;
    else
        return n*f(n-1);
}
```

例：汉诺塔问题 -

内容回顾

关键在于分析当前规模和简化后规模的关系

```
#include <iostream>
using namespace std;
void hanoi(char x,char y,char z,int n) //把n个圆盘从x表示的
                                       //柱子移至y所表示的柱子。
{  if (n == 1)
        cout << "1: " << x << "→" << y << endl; //把第1个
                                //盘子从x表示的柱子移至y所表示的柱子。
   else
   {    hanoi(x,z,y,n-1); //把n-1个圆盘从x表示的柱子移至
                          //z所表示的柱子。
        cout << n << ": " << x << "→" << y << endl;
                //把第n个圆盘从x表示的柱子移至y所表示的柱子。
        hanoi(z,y,x,n-1); //把n-1个圆盘从z表示的柱子移至
                          //y所表示的柱子。
   }
}
```

例：母牛生小牛问题 -

内容回顾

关键在于分析当前规模和简化后规模的数值关系

```
myGetCowR(int year, int m)   // 一般的情况
{
        if ( year < m )
                return 1;
        else
                return myGetCowR(year-1) + myGetCowR(year-m+1);
}
```

宏定义

内容回顾

- 在C++中，利用一种编译预处理命令：宏定义，用它可以实现类似函数的功能：
  - #define <宏名>(<参数表>) <文字串>

  例如：
  - #define max(a,b) (((a)>(b))?(a):(b))

- 在编译之前，将对宏的使用进行文字替换！

  例如：编译前将把
  - cout << max(x,y);

  替换成：
  - cout << (((x)>(y))?(x):(y));

内联函数

- 内联函数是指在定义函数定义时，在函数返回类型之前加上一个关键词inline，例如：

```
inline int max(int a, int b)
{
    return a>b?a:b;
}
```

- 内联函数的作用是建议编译程序把该函数的函数体展开到调用点，以提高函数调用的效率。
- 内联函数形式上属于函数，它遵循函数的一些规定，如：参数类型检查与转换。
- 使用内联函数时应注意以下几点：
  - 编译程序对内联函数的限制
  - 内联函数名具有文件作用域

函数名重载

内容回顾

- 对于一些功能相同、参数类型或个数不同的函数，有时给它们取相同的名字会带来使用上的方便。例如，把下面的函数：

```
void print_int(int i) { ..... }
void print_double(double d) { ..... }
void print_char(char c) { ..... }
void print_A(A a) { ..... } //A为自定义类型
```

定义为：

```
void print(int i) { ..... }
void print(double d) { ..... }
void print(char c) { ..... }
void print(A a) { ..... }
```

- 上述的函数定义形式称为**函数名重载**。

例：用函数实现求小于n的所有素数(回顾)

```
#include <iostream>
#include <cmath>
using namespace std;
bool is_prime(int n);                    //函数声明
void print_prime(int n, int count);  //函数声明
int main()
{    int i,n,count=1;
     cout << "请输入一个正整数："
     cin >> n; //从键盘输入一个正整数
     if (n < 2)    return -1;
          cout << 2 << ",";   //输出第一个素数
     for (i=3; i<n; i+=2)
          if (is_prime(i))
          …… ……
     cout << endl;
     return 0;
}
bool is_prime(int n) //函数定义
{
     int i,j,k=sqrt(double(n));
     for (i=2, j=k; i<=j; i++)
               if (n%i == 0)
          return false;
     return true;
}
```

如何使用这些库函数？

还有那样的一些库函数？

4.6 C/C++库函数和条件编译

郭 延 文

2019级计算机科学与技术系

C++标准库函数

- 为了方便程序设计，C++语言提供了标准库，其中定义了一些语言本身没有提供的功能：
  - 常用的数学函数
  - 字符串处理函数
  - 输入/输出
  - …
- 在标准库中，根据功能对定义的程序实体进行了分类，把每一类程序实体的声明分别放在一个头文件中
- 在C++中，把从C语言保留下来的库函数重新定义在名空间std中；对相应的头文件进了重新命名：*.h -> c*

```
#include <stdio.h>
#include <cstdio>
```

一些标准数学函数（cmath或math.h）

- int abs(int *n*);　　//int型的绝对值
- long labs(long *n*);　　//long int型的绝对值
- double fabs(double *x*);　//double型的绝对值
- double sin(double *x*);　//正弦函数
- double cos(double *x*);　//余弦函数
- double tan(double *x*);　//正切函数
- double asin(double *x*);　//反正弦函数
- double acos(double *x*);　//反余弦函数
- double atan(double *x*);　//反正切函数
- double ceil(double *x*);　//不小于x的最小整数（返回值为以
  // double表示的整型数）
- double floor(double *x*);　//不大于x的最大整数（返回值为以
  // double表示的整型数）
- double log(double *x*);　　//自然对数
- double log10(double *x*);　//以10为底的对数
- double sqrt(double *x*);　　//平方根
- double pow(double *x*, double *y*);　//x的y次幂
- ……

cmath和math.h的用法区别

- C++把从C语言保留下来的库函数，重新定义在名空间std中；对相应的头文件进了重新命名：*.h -> c*

- cmath: 标准c++库文件

  #include <cmath>

  using namespace std;

- math.h：c语言头文件，兼容c风格的库文件

  #include “math.h”

C++模块调用(回顾)

- 一个C++模块一般包含接口(.h)和实现(.cpp)两个部分
  - .h：本模块中定义的、提供给其它模块使用的一些程序实体（如：函数、全局变量等）的声明
  - .cpp: 模块中的程序实体的定义

- 在模块A中要用到模块B中定义的程序实体时，可以在A的.cpp文件中用文件包含命令（#include）把B的.h文件包含进来，格式如下：

  #include <文件名> 或 #include "文件名"

编译预处理命令

- C++程序中可以写一些供编译程序使用的命令：**编译预处理命令**。
- 编译预处理命令不是C++程序所要完成的功能，而是用于对编译过程给出指导，其功能由编译预处理系统来完成。
- 编译预处理命令主要有：
  - 文件包含命令（#include）
  - 宏定义（#define）命令
  - 条件编译命令？

Adobe Photoshop

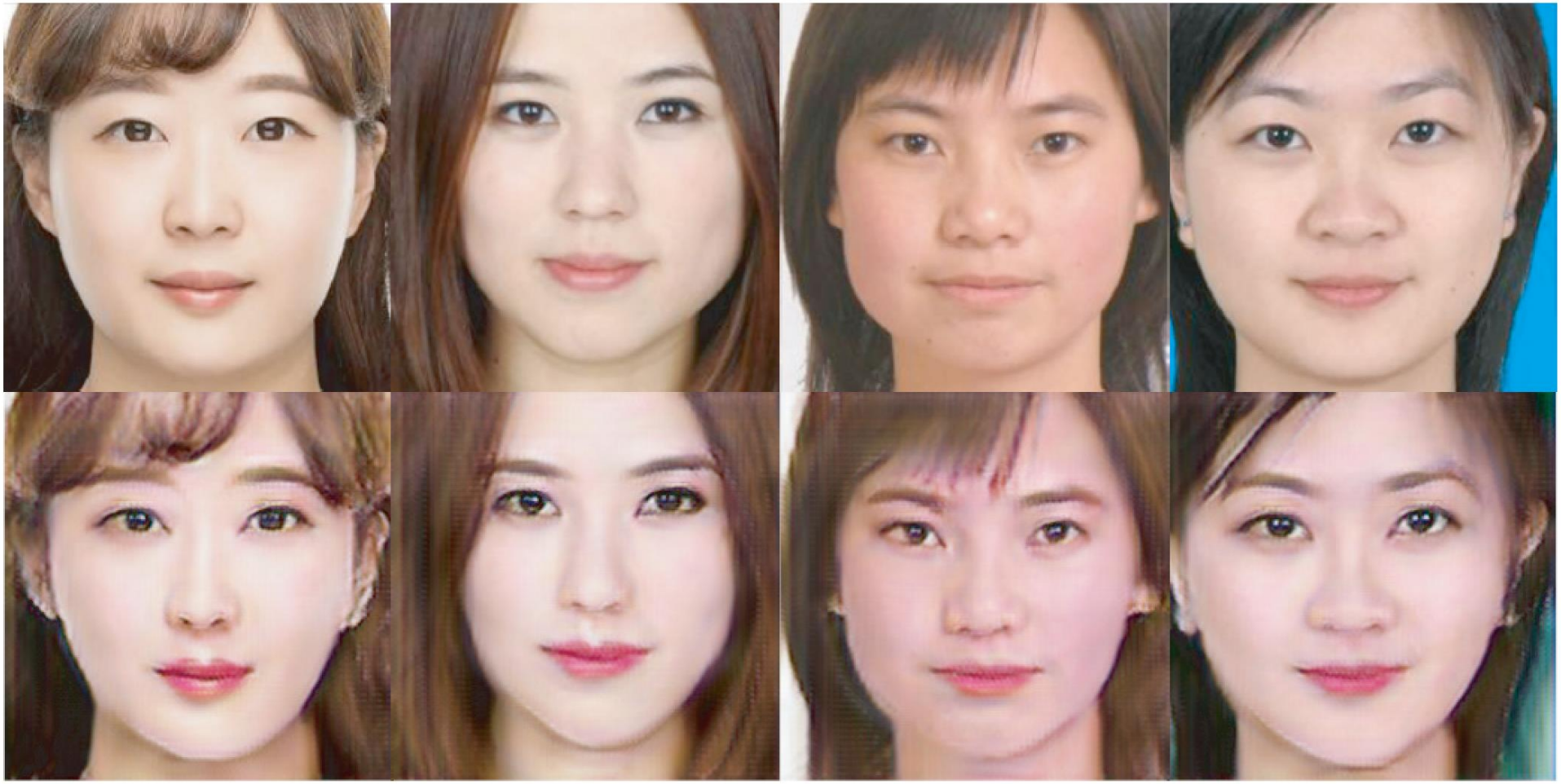
手机图像增强和增强现实

需要Android和IOS版本！

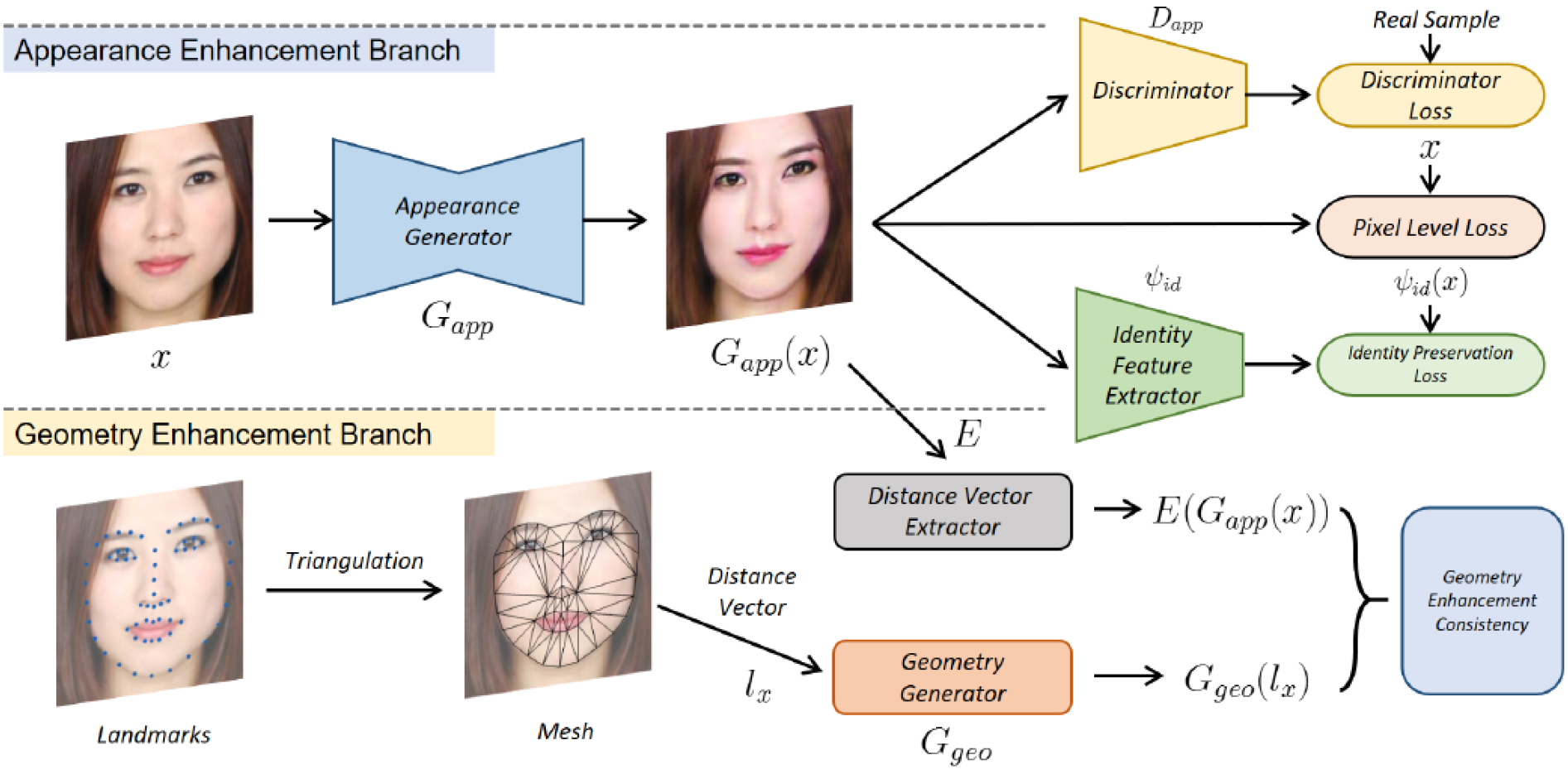
核心算法一致，

但...

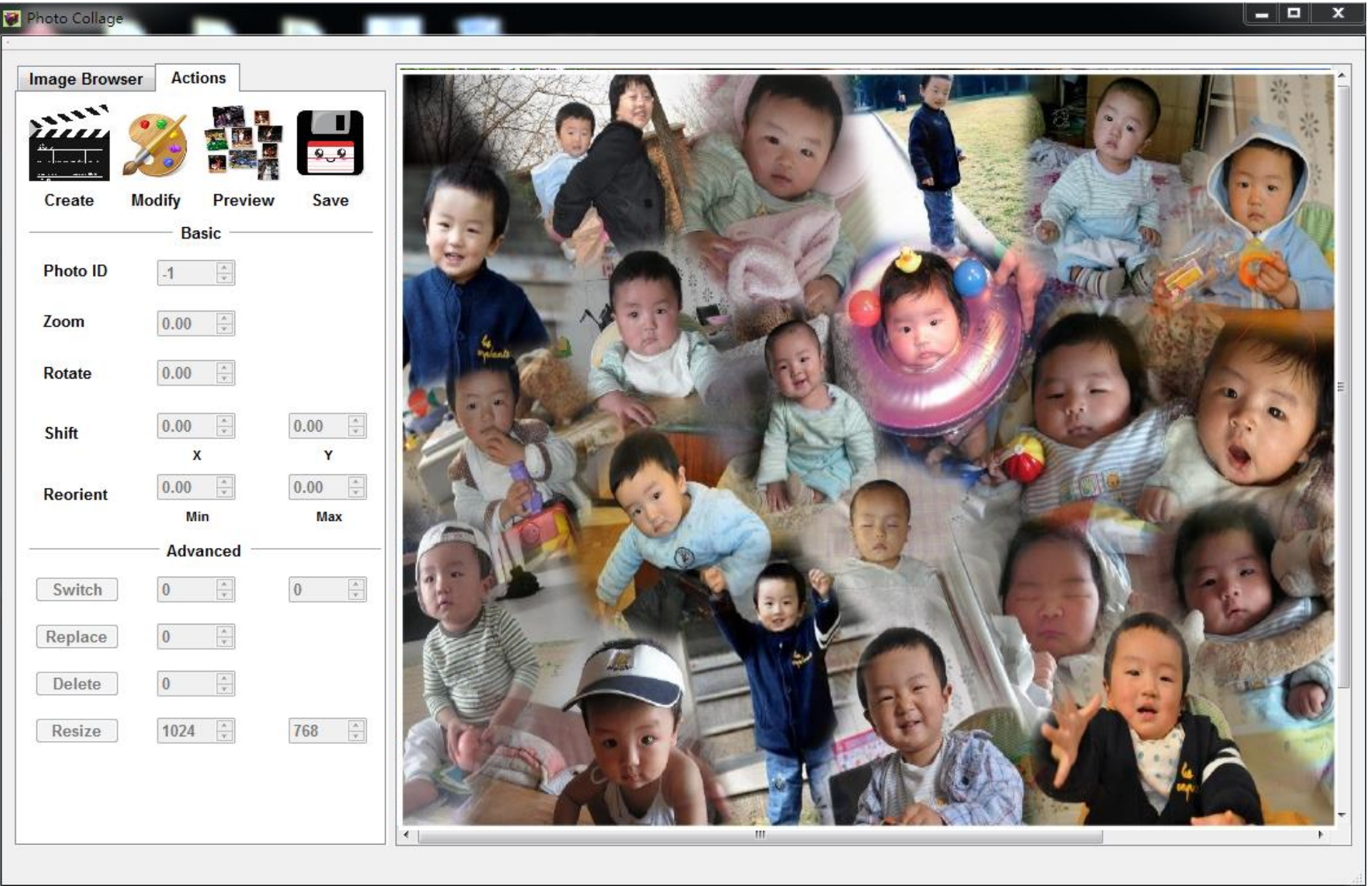
Our Group: [本组研究成果]
基于深度学习-对抗生成网络的人脸美化



Our Group: [本组研究成果]
基于深度学习-对抗生成网络的人脸美化



人脸检测的应用：照片Collage[本组研究成果]



条件编译

- 编译程序根据不同的情况来选择需编译的程序代码。例如，编译程序将根据宏名ABC是否被定义，来选择需要编译的代码：

```
<代码1> //必须编译的代码
#ifdef ABC
        <代码2> //如果宏名ABC有定义，编译之
#else
        <代码3> //如果宏名ABC无定义，编译之
#endif
<代码4> //必须编译的代码
```

- 用于条件编译的宏名ABC在哪里定义？

- 方式1：在程序中定义宏

```
<代码1> //必须编译的代码
#define ABC …… 
#ifdef ABC
    <代码2> //如果宏名ABC有定义，编译之
#else
    <代码3> //如果宏名ABC无定义，编译之
#endif
<代码4> //必须编译的代码
```

- 方式2：在编译环境中定义宏
  - VS 2008及以上或C++ 6.0的集成开发环境中，选择“Project|Settings”菜单，在“Project Settings”对话框的“C/C++”选项卡中，选择“Category”中的“Preprocessor”，然后在“Preprocessor definitions”中添加要定义的宏名：ABC。
  - 命令行

    cl <源文件1> <源文件2> … -D ABC …

总结：条件编译(常用)

- 编译程序根据不同的情况来选择需编译的程序代码。例如，编译程序将根据宏名ABC是否被定义，来选择需要编译的代码：

```
<代码1> //必须编译的代码
#ifdef ABC      // #ifndef <宏名>
      <代码2> //如果宏名ABC有定义，编译之
#else
      <代码3> //如果宏名ABC无定义，编译之
#endif
<代码4> //必须编译的代码
```

- <宏名>可以在程序中用#define定义，也可以在编译器的选项中给出

条件编译命令的另一种格式

```
#if <常量表达式1> / #ifdef <宏名> / #ifndef <宏名>
   <程序段1>
#elif <常量表达式2>
   <程序段2>
……
#elif <常量表达式n>
   <程序段n>
[#else
   <程序段n+1>]
#endif
```

- 上述条件编译命令的含义是：如果<常量表达式1>的值为非零（#if）或<宏名>有定义（#ifdef）或<宏名>无定义（#ifndef），则编译<程序段1>，否则，如果<常量表达式2>为非零值，则编译<程序段2>，…，否则如果有#else，则编译<程序段n+1>，否则什么都不编译。
- <常量表达式>中只能包含字面常量或用#define定义的常量。

基于多环境的程序编制

```
#ifdef UNIX
   ..... //适合于UNIX环境的代码
#elif WINDOWS
   ..... //适合于WINDOWS环境的代码
#else
   ..... //适合于其它环境的代码
#endif
..... //适合于各种环境的公共代码
```

条件编译的作用

- 条件编译的作用：
  - 基于多环境的程序编制
  - 程序调试
  - ……

程序调试

- 加入调试信息

  #ifdef DEBUG

  …… //调试信息，主要由输出操作构成

  #endif

- 问题：写起来比较麻烦！

- 利用标准库中定义的宏：**assert**

```
#include <cassert> //或<assert.h>
.....
assert(x == 1); //断言
```

- assert的定义大致如下：

```
.....
#ifdef  NDEBUG
#define assert(exp)  ((void)0)
#else
#define assert(exp)  ((exp)?(void)0:<输出诊断信息并调用库函数abort>)
#endif
.....
```

应用举例

```
function fo(){
  $fp = fopen("c:/test.php",'w');
  fwrite($fp,"123");
  fclose($fp);
  return true;
}
assert("fo()");
// 下面是关于fp的些操作
// 上面assert“确保”了只有以上函数正常运行时才对fp操作！
…
```

应用举例

```
#include <iostream>
using namespace std;
#define DEBUG
```

```
int main()
{
        int x = 999;
        cout<<"第一次执行ASSERT（）："<<endl;
         ASSERT(x==999);
         cout<<"第二次执行ASSERT（）："<<endl;
         ASSERT(x!=999);
         cout<<"程序结束"<<endl;
         return 0;
}
```

Q & A