Application - Virtual Blue Screen: Most folks are familiar with blue screen video effects formally known a chroma key http://en.wikipedia.org/wiki/Bluescreen. This is where actors are filmed in front of illuminated green surfaces. Later, computer techniques identify the background region and replace it with an alternate filmed or computer generated background. Blue screen sets must be constructed, often at great expense. So how hard would it be to produce a "virtual blue screen" set where background objects are automatically identified and replaced by alternate frames? Let's find out.

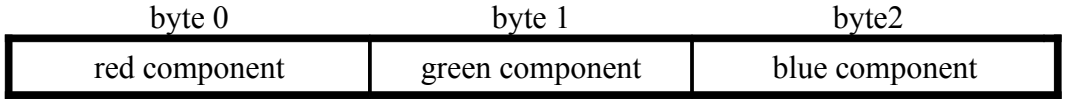
We'll use outdoor image sequences which contain dynamic background, changing lighting, and plenty of shadows; this is the most difficult environment for foreground extraction. To help bound the computational burden, we'll consider a short, low frame rate (3-4 frames per second) sequence and we'll use a single, invariant background image.

Walk in the Park: Who wants to be headed to class on a beautiful Spring day? A sequence of 180 images contains a cropped strip of Tech students walking to exams. Through the magic of technology, we'll teleport them to an open field where they can fully enjoy the day. The primary task is to cut out the moving students in each frame and then paste them over a fixed position in the background image. The resulting images will then be stored to form an "improved" video sequence of happy folks walking in a park. Along the way, we'll also produce a summary of intermediate processing results.

This project requires effective *integration* of available software packages and libraries, along with your own code, to complete a quality implementation. Programming in the corporate world rarely allows a computer engineer to write *all* the software. Instead, we employ available software, writing connecting code so that components work together. Debugging, testing, and improving robustness and performance are all significant tasks in software design. As in previous projects, submissions demand both accuracy and performance.

JPEGs and Frame Buffer: To simplify frame processing, sequences are distributed as a collection of numbered jpeg images. Each frame is loaded, one at a time, into a frame buffer where pixel data can be easily processed.

Most digital imaging devices represent color pixel data as a triple of typically eight bit integers representing the intensity of its red, green, and blue components (RGB format).



A frame buffer contains a byte array of RGB bytes. Its size (in bytes) is the product of the image height and width, multiplied by three for RGB values of each pixel. Needless to say, there is no word alignment … just a bunch of bytes.

But JPEG images use compression techniques to reduce their storage size. Specifically, it reduces high spatial frequency detail in the image, that humans do not see well anyway. JPEG also codes the pixel data in a decidedly non-RGB format. What's the easiest way to decode this format?

Over the last two decades, JPEG has presented plenty of technological (and legal) challenges. Fortunately, there are good software libraries to encode and decode JPEG images. The Independent JPEG Group runtime library is widely used and does a good job.

An image utility library, `utils.c`, simplifies and extends this functionality, and introduces a frame buffer object and accompanying access functions. Here's the key object: a frame buffer.

```
typedef struct FrmBuf {
   unsigned char       *Frm;
   int                 Height;
   int                 Width;
   struct FrmBuf       *Next;
} FrmBuf;
```

A frame buffer contains decoded image information as an array of unsigned bytes, plus the image dimensions. The image utility library also explicitly manages frame buffer objects. All function descriptions, prototypes, and source code is available in the code listing.

It is important to reuse frame buffers, they are rather large (269K bytes). Once a proper sized frame buffer is created, many frames can be loaded into it. Note also that the `Copy_Image` function can copy one frame buffer *into* another.

Foreground Extraction: Effective extraction of foreground image data is not a simple task. Image subtraction, subtracting the current image from a reference frame, depends on uniform lighting, stable background, and accurate image capture, all of which are rare in real world scenes. A good technique captures multiple values, or *modes*, at each pixel. These RGB triples

are then compared against an input pixel. If it matches each component with a specified epsilon (maximum component difference), the pixel is declared background. Otherwise the pixel is foreground.

The Multi-Modal Mean libraries includes a efficient algorithm for pixel level multi-modal capture using a ratiometric representation of modes. Image removal of background depends on a good model. While this model does not require a "blank" reference frame containing no foreground, it does need a preamble period (a few frames) to recognize background pixels.

The library code, in `mmm.c`, contains an explanation of the technique, a description of key functions, and an example usage, plus the source code. An important function, `Process Frame Foreground`, adds a new frame to the background model. It returns the frame with background pixels blacken (set to zero).

Density Analysis and Blob Detection (Rollers): Even the best foreground extraction method contains lots of noise. The goal is to get as much of the foreground as possible while capturing as little of scene background as possible. In the parlance of the trade, this is called segmentation that minimizes false negatives (foreground is blackened), and false positives (background that is not blackened).

Since false negatives tend to be near large segments of the foreground, and false positives are often scattered in small clusters, a density filter can help improved analysis of the foreground extracted image. Read up on the density filters to learn how.

A blob annotated image builds on the density map created by the density filter. It clusters adjacent positions in the image that are above an established density threshold. Once blobs are found, they can be displayed using the density map thresholding function. Annotations include marking each blob's center of mass, bounding box, and blob map. Good news: these functions are already coded for your use.

An efficient spatial density analysis and blob detection package (`rollers.c`) allows the non-zero foreground to be grouped, thresholded, and blobified. Different methods are available to compute the linear and area density of non-zero pixels. The result is placed in a preallocated density map. Once the density map is computed, it can be used to render the frame as a colorized, grayscale, or thresholded map. It also includes extensive support for blob identification. Key input parameters are mask size and blob threshold. Documentation is provided in the source code. The returned blob list contains blob structures that include bounding box, area, and center of mass. A blob map can also be generated.

Debug Mode: It is convenient to add debugging information (i.e., print statements) during execution. Yet these debugging statements must be turned off to achieve the highest performance. Rather than removing the debug statements, it is better to include it as a condition to a defined value DEBUG. For example:

```
if (DEBUG)
   printf("   loading first image %s ...\n", Path);
```

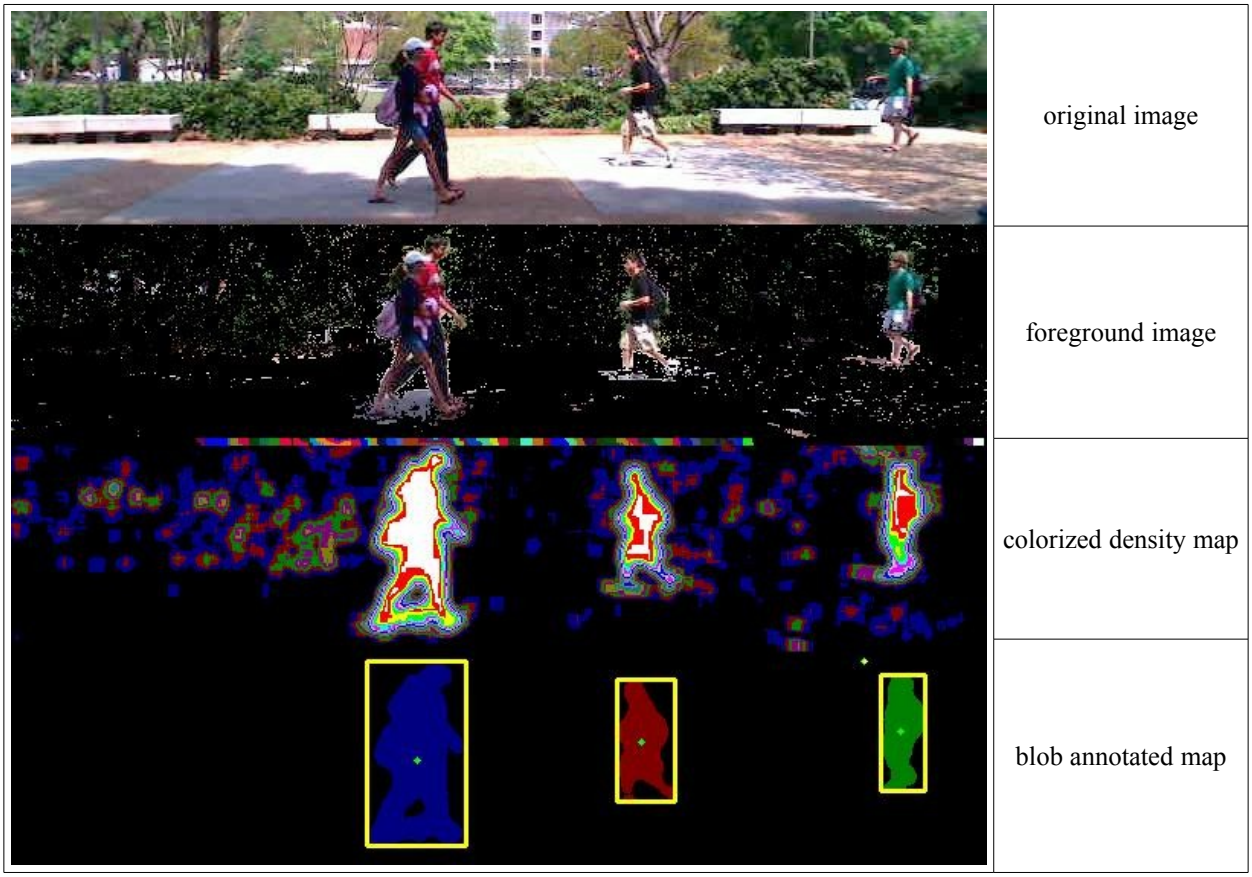
This DEBUG define is in header file `utils.h`. You can turn it on or off as needed. *Do not include any debugging information without this conditional*. Your project performance and grading depends on it!

Results: The sequence frames are 640 pixels wide and 140 pixels high and they are number `00001.jpg` to `00180.jpg`. They should be store in the sequence directory "`./seqs`" in a subdirectory "`./seqs/InSeq/`". The backdrop image is named `park.jpg`. The background

image is the same width as the input sequence, but it is 480 pixels high. So the transfer foreground pixels must be offset to realistic position within the background field. People don't walk in the sky. It is highly recommended that the input frames and the backdrop image be read exactly once.

For each frame, your submission should produce an output image containing the park background image with all appropriate overlaid foreground. It should be named `out00xxx.jpg` and it should be stored in the "`trials`" directory.

Results Stack: The progress of your submission depends in part on the quality of the intermediate processing results, including the following:



Your code should produce an image contain these results for each frame of the input sequence. The file should be named `rs00xxx.jpg`. These result stack images should be created in the same directory as the field composite images.

Academic Honesty: This project should be completed independently by each student. Algorithms and code should not be shared at any time. No code should be used except for (a) code in the included libraries, and (b) code written by you alone.

P3-1 Results Stack and Park Output Images: When have completed the assignment, submit the single file `P3-1.c` to the class website. Do not submit data files. Do not make changes to library routines as they will not be turned in. They will be used in their original form in the grading process. Although it is good practice to employ a header file (e.g., `P3-1.h`) for declarations, external variables, etc., in this project you should just include this information at the beginning of your submitted program file.

In order for your solution to be properly graded, there are a few requirements. Not following these requirements will result in lost points.

1. The file must be named `P3-1.c`. ***You should only turn in this file***.
2. Your submitted file should compile and execute using the provided makefile, support programs, and JPEG encoding/decoding package. The command line parameters should not be changed. This submission should produce 180 results stack images in the trials directory displaying the original image, extracted foreground image, colorized density map, and annotated blob map. This submission should also create 180 output images that include the background scene plus the appropriate extracted foreground from the input sequence.
3. Your solution must be properly uploaded to the submission site before the scheduled due date, **9:00pm** on **Friday, 22 April 2011.**

Grading Metric: Projects will be scored based on accuracy and performance using sequences that are similar to the examples provided. The project grade will be determined as follows:

part	*description*	*due date 9:00pm*	*percent*
P3-1	Walk in the Park	Fri, 22 April 2011	
	results stack: correct operation and accuracy		30
	park output: correct operation and accuracy		50
	performance		10
	style		10
	total		100