Project Overview

The primary objective for this project will be to develop an algorithm that can be used in a Kodak Picture Kiosk that will take a source image and split it into 2 or more image panes horizontally or vertically. The algorithm should support 2up, 3up, and 4up layout styles. Ideally, the algorithm will result in artistically pleasing splits (i.e. Not splitting a person’s face across two panes), but initially a goal of this ideal split being achieved 20% of the time has been set. Additionally, the end user will have the ability to “nudge”, crop, resize, and zoom any of the split images around on the page layout to customize it to their liking. Another feature which may be included if time allows is the integration of our algorithm with the previously created Photorganizer automatic layout tool. Settings for the algorithm and the accompanying application (described below) must be easily configurable.

We also need to create an application to demonstrate the algorithm as it would function on a Kodak kiosk, and this application will also serve as a testing harness during development. The application must have a Windows XP installer, but must also run on Windows Vista, and will run in kiosk mode in Internet Explorer. To give the application a pleasing look and feel, we will be utilizing WPF (Windows Presentation Foundation), a standard in Visual Studio .NET 2008, using C#. WPF will allow us to provide an almost Apple-esque experience in the user layout, with the workflow being analogous to picking images from a shelf and laying them out on a table, while maintaining the look of the
current Kodak Picture Kiosks. This kiosk application will give users a more intuitive experience, while not overloading them with options and features, by limiting the available choices per screen.

**Basic Requirements**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Algorithm splits images 2 panes vertically        | Frame is drawn with a bar down the middle vertically, giving the appearance of two separate images.  
Input : Desired Image  
Output : Image split down the middle vertically |
| Algorithm splits images 2 panes horizontally      | Frame is drawn with a bar across the middle horizontally, giving the appearance of two separate images.  
Input : Desired Image  
Output : Image split across the middle horizontally |
| Algorithm splits images 3 panes vertically        | Frame is drawn with two vertical bars. The first bar should be at 1/3 width and the second at 2/3 the width. Giving the appearance of 3 separate images.  
Input : Desired Image  
Output : Image split down into 3 parts vertically |
| Algorithm splits images 3 panes horizontally      | Frame is drawn with two horizontal bars across. The first bar should be at 1/3 height and the second at 2/3 the height. Giving the appearance of 3 separate images.  
Input : Desired Image  
Output : Image split across into 3 parts horizontally |
<p>| Algorithm splits                                  | Frame is drawn with a bar down the middle vertically, and a bar across the middle horizontally. Giving the |</p>
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
</table>
| images 4 panes              | appearance of four separate images.  
Input: Desired Image  
Output: Image split down the middle vertically and across horizontally |
| Algorithm can automatically determine a split | Algorithm will determine what the best split of the photograph is. If the algorithm determines the image is un-split able, notify the user  
Input: Desired Image  
Output: Image will have any one of the possible splits  
Output 2: Image will remained unchanged if split not possible |
| Ability to Nudge            | User should be able to nudge the picture around if the split provided doesn’t suit needs.  
Input: Desired Image and User Input  
Output: Image is adjusted in frame to users liking. |
| Ability to Zoom             | User should be able to zoom in and out of desired subject matter  
Input: Desired Image and User Input  
Output: Image is zoomed in or out to users liking. |
| Configurable Settings       | All algorithm settings need to be configurable by an administrator |
| Windows Installer          | The application needs to have an installer, the installer needs to work on the specified platform at the very least |
| 3d Effects                  | The application needs to be developed with WPF. The application should showcase some of the 3d effects that are native to WPF |
Constraints

We faced many constraints when creating our solution for Kodak. At the base level, we were restricted to coming up with a viable, effective solution in 20 weeks. This was compounded by having only our 5 team members to draw experience from and to delegate work to, and having no funding to purchase certain heavily suggested development technologies, such as Microsoft’s Expression Blend. Additionally, to avoid having to sign a nondisclosure agreement with Kodak, our ability to use pre-developed modules and code from Kodak was limited.

On a more technical level, our solution was to run on Windows XP embedded, and be Windows Vista capable, on a machine with a 3 Ghz processor and 1 GB of memory. Since we were developing for a Kodak Picture Kiosk, we also needed to develop a system that would work on a 15” touchscreen monitor at a resolution of 1024X768. It was also strongly recommended that we develop in C# and Windows Presentation Foundation (WPF), using Microsoft Visual Studio 2008. If any part of our project used something other than C#, those components had to be accessible from C#. Our program had to be fast, splitting images in a short amount of time, and powerful, able to load many images at once, to appeal to consumers.

Other constraints to our development of the project were due to the nature of the project’s goal. The application we created was to “aesthetically split” images, which was mostly foreign territory to the group. A few of us had taken computer vision courses in the past, but not to the depth that this solution would require. Since aesthetics are subjective, we had no perfect measure to determine if our work was splitting images correctly, nor was there an established starting point for us to work from. Additionally, we needed to come up with a computer vision library that would work for us, and our limited funding, time, and manpower made finding a pre-created library our best bet.
Development Process

We felt the spiral methodology would be appropriate for our project. First of all we recognized that there were a lot of risks involved in this project. The team had a lot of research to do in order to learn new technologies and comprehend image splitting technology. The spiral model is a good choice for high risk projects since the model is a risk oriented model. The spiral model is also good for providing the customer with multiple iterations. We felt that providing the sponsor with progress updates through multiple releases would not only help them see where we’re going, but also provide direction for us. We had hoped that it allowed them to provide feedback for what we’re doing and we could adjust our development efforts accordingly. The spiral model also contains four main phases in each iteration that include determining objectives, identifying and resolving risks, development and test, and planning for the next iteration. Our sponsors agreed that this methodology would be most appropriate as well as our coach.

The spiral methodology also ensured that documents were up to date and detailed. We have our requirements document, our system architecture, our interface specifications, software test plan, well commented source code, a component detailed design document, and our risk analysis document as well as supporting documents.

Project Schedule: Planned and Actual

The project schedule was developed with the major requirements in mind. Originally, we knew we wanted to get a firm base that we could easily build off of. With this in mind, the project schedule was built around our spiral model. We decided to have a number of iterations to gain valuable feedback from potential users and our sponsors. One of the key activities was to get the face detection working and being able to at least split images into 2 panes. We also identified that getting the carousel to function with animations and other UI effects were important to get in early so we could modify it later. We also felt the functions for being able to edit images on the screen. The ability to move the image around and zoom in and out was key activities for the project. The other
key activities were adding in the edge detection for the algorithm and color segmentation as well.

The actual schedule was solid right up until the end of the project. However, we planned for slack time at the end of our project, so it didn’t affect us negatively. Overall, we felt the actual schedule was built well and did need some minor tweaks here and there. There were as in any project unexpected hurdles that caused us to modify our schedule slightly, but with our risk-oriented model we were able to mitigate them very well. We adapted well to changes and our schedule was on the ball in most cases. Our schedule worked well as we developed it with the long haul in mind. We also knew realistically we wouldn’t be able to get a ton of work done over break or during exam weeks, so our schedule adhered to this.

**System Design**

Image Splitter system architecture.
Above is our system architecture of the Image Splitting program. The 3 most important parts of this system is the User Interface module, the Image Module, and the CIA. We’ll first discuss the User Interface Module. We were tasked to using WPF in our application and showcase some of its 3d effects. The user interface module is where all of this occurs.

The UI design was dictated by the way WPF works and our understanding of WPF. We used templates to help make our components and make them reusable. These components were stored in a resource dictionary as xaml code. This would be loaded dynamically by code as events were sent through the UI. Our animations were made using storyboards. The storyboards allowed us to create a set flow of actions and changes that occur with UI elements on the screen based upon specific time intervals.
The next section we’ll discuss is the image module. This module consists of three parts. The first part is the ImageManager. This is the class that allows access to all the functionality of the algorithm or other image manipulation tools. Below this are smaller modules for tools and the algorithm. The tools are where we do any type of image manipulation: Nudge, Zooming, Resizing, etc. The algorithm sub module does all the computation on the image to determine if it is split able or not. The main reason we created these sub modules was to make the project extendible to Kodak. When designing our system, we assumed that Kodak already had commonly used functions or libraries for these kinds of operations. We wanted to give Kodak the option of taking out our tools and including their own tool set. The same goes with the algorithm.

The algorithm module has access to three types of algorithms: Face Detection, Edge Detection, and Color Segmentation. We developed this function with the assistance of an image library called openCV (computer vision) developed by Intel some years ago. By using this library we did not have to spend a large amount of time developing detection algorithms from scratch.

Our initial goal was to have all our algorithms working together to give us the best result. However we quickly realized that this created a performance issue. For our algorithms to communicate effectively and give us a favorable result, we would have to run our detection algorithms on our image multiple times, this is because we might have to adjust the image based off one result and have to rescan to make sure the new result is favorable. This quickly broke our performance requirement. It was because of this we decided not to have our detection algorithms communicating with each other. What we decided to do was run all three concurrently. When all three had finished do their calculations we chose our results based on preference. The preference we decided was Face Detection -> Edge Detection -> Color Detection.

If at the end of our image processing, face detection returned favorable results we would return those results and not worry about what edge and color returned. However, if face were to fail we would look at the results of edge detection. After we implemented this approach we were pleased with the results. Most pictures with people in them was appropriately handled by the face detection algorithm. Images without faces were
subsequently handled better by edge and color detection. Our results were favorable and our computation time was very manageable for the average picture. Large pictures such as panoramic would still take quite a long time, but this expected.

The CIA is our main controller. It facilitates communication between all the modules in the system. The CIA has no functionality of its own; it only calls on the appropriate function from the appropriate module.

The storage module supports our ability to load in images from different locations and devices. It currently only supports the ability to load from a folder. Our sponsor said focusing on different type of devices was not a focus and any other formats would be a stretch goal. We designed it so that adding new types of devices would be simple and slide into the program with relative ease.

The Settings Manager module was never implemented. This was our attempt to make sure that our settings were configurable. Our sponsor expressed that as long as our settings were documented and easily changeable in a file or dialog it would be acceptable. This was taken out by request of our sponsor. It was their desire that we focus on the UI and the algorithms development. It was then regarded as fluff and never followed.

The memento module was also never implemented. We wanted to support being able to backtrack through states. We thought this would be useful if a user decided to change their mind and go back and redo some images. Our sponsor labeled it as a stretch goal and it was only to be implemented if time allowed. We never had the time to attend to this module.

**Process and Product Metrics**

To track the progress of development, we collected a number of measurements and created metrics from them.
First, we kept track of the number of tasks we completed. This measurement, compared over time, provides us with our first metric: cumulative completed tasks over time. We compared actual tasks completed over time to the number of tasks we estimated we will complete at that time. This metric was used to track how well we were adhering to our own schedule.

The results of the number of actual tasks completed to the estimated number of tasks completed showed that for the most part, we overestimated our number of tasks. This shows a lack of granularity because this doesn’t mean things didn’t get done, it means that things that were completed, often encompassed other tasks which makes the metrics look like we did not complete as much as we actually did. This metric also shows that we did not have a steady climb in a number of tasks. We did not become more granular as the project continued.

The next measurement we monitored was number of defects in the system. We observed open and closed defects, as well as their sum. These metrics were collected during the relevant section of the development process. The results of this metric show that we had a large number of defects show up later in our product. This is due to further testing and more integration testing that occurred later in the project. And, due to time constraints, we were unable to close all of our defects.

We also collected the amount of effort expended by the team as the project went along. Comparing actual effort to estimated effort showed, among other things, complexity of tasks. This metric was compiled and allowed transparency on the topic of schedule adherence. It shows that in the beginning of the project, our estimating of task time were not as accurate as they could be. Later in the project, our estimated hours matched our actual hours much more closely. Much like the number of tasks, the estimated hours and actual hours of tasks did not steadily increase throughout the life of the project. The metric shows that we put just as much effort into the beginning of the project as the end of the project.

These metrics show that our work ethic went very well. We were able to put a lot of effort into the beginning of the project as well as the middle and end of the project. The only dip in productivity is seen at the mid-project break (spring break). Most of our
estimates were slightly larger than our actual. As the project progressed, so did our estimation abilities. Part of this is becoming more granular and splitting tasks into easier to estimate pieces.

All of these metrics are available at
http://www.se.rit.edu/~imagesplitters/stuffOnWebSite/Tracking/Activity%20Tracker/ActivityTracker.xls

Product State at Time of Delivery

The Image Splitter is complete, though not completely bugless. A few errors proved too elusive to track down, especially in the UI. All of the features that we agreed to have in the project are included in the delivered product. See our SRS for a list of functional requirements (section 2.2). Some features were implemented differently than originally envisioned, such as reorienting splits, which originally was to happen automatically upon pushing the button, but, in implementation, the button must be pushed first, and then the split can be performed. Zooming also was implemented differently than originally described, as we use buttons to zoom in pre-determined increments, rather than allowing the user to create a region of interest box to zoom on. Furthermore, the 3D effects used in our UI were deemed “not 3D enough,” but it was too late in development to go back and change them. However, we managed to include some stretch goals, such as the final page with the carousel of split images. Additionally, we were able to deliver a working batch splitter to let users test the effects that different algorithm settings would have on the resulting splits. We were not able to implement some stretch goals, due to time constraints and a desire to eliminate as many bugs as we could.

Project Reflection

Some things that went right were that we got great advice from experts in the computer vision field, we avoided having to code our own face detection algorithm by using OpenCV, we kept in regular contact with our sponsors, and we were an optimal group to work on this project.

Our faculty coach had experience in the field of computer vision and image manipulation, and our sponsors put us in contact with Ray Ptucha, a professor of Imaging
Science here at RIT. Both men provided us with guidance and suggestions on how we could make our algorithm better, and pointed out things that might pose a hurdles to our work.

OpenCV provided us with ready-made methods to find faces in images. This functionality provided us with the basis of our image analysis algorithm. Since we did not have to create methods to perform these actions, we were free to focus on other aspects of the application.

Kodak’s representatives were very communicative with us, and provided valuable feedback on our documents and prototypes. Without this feedback, we would have likely created something that did not meet their needs and cost us a lot of development time. In fact, this occurred on a small scale in that we initially envisioned the splitting functionality to be quite different than what was finally implemented.

Furthermore, our group was well-suited to this project. A few of us had previous experience in Computer Vision, and we all were interested in the project and problem domain from the start. Equally important, we all got along, and were willing to do our fair share to make sure the work that needed to be completed got done.

Things that went wrong included OpenCV not being natively in C# and the wrapper class we found was not perfectly implemented, limited free time made meeting and getting group work done difficult, some initial requirements were misinterpreted and we lost some development time to fixing them, and we had a very difficult time being granular enough in reporting our activities and efforts to our sponsors and coach.

OpenCV was initially written in C, which would not help us, since we had to code in C#. We located a wrapper class which we could use to access the C methods. Unfortunately, as this wrapper class was human-coded, it was susceptible to human error, which we discovered while attempting to write a KMeans-based subalgorithm. A lot of time was spent attempting to work around these limitations, as well as others we found, which cost us some development time.
References

1. http://www.se.rit.edu/~imagesplitters/stuffOnWebSite/Tracking/Activity%20Tracker/ActivityTracker.xls