Scene Synthesis
CS 766: Computer Vision

David Koop
December 17, 2003

Abstract
With today’s digital cameras, one can acquire a wealth of image data with little effort. However, the use of these images is relatively limited: most images are used simply to capture a scene at a given point in time. In addition, scenes that do not represent an actual view of the environment are usually drawn or modeled from scratch. In this project, we investigate a cut-and-paste approach for generating novel scenes. Ideally, we would like to take a database of images and given a scene description, composite a subset of the existing images to generate an image of the defined scene. Notice that our technique lies somewhere between image processing and virtual environment generation. This project presents some results of the technique and addresses some of the inherent challenges.

1 Introduction
This project addresses a question which spans the areas of image processing and vision: given two images or selected regions of those images, how do you combine them in a meaningful way? Of course the answer to this question depends on a lot of characteristics of the images. If the images are similar enough, we can track the changes and mosaic the scenes together. If the images are taken with the exact same camera parameters, but at different times of the day, we could remove parts of the images based on the changes in the sequence. If we are given two unrelated images, we enter into a much more difficult realm. Because we can’t assume any coherency between the images, whatever knowledge about the relationship between them must either be derived using vision techniques or inputted by the user.

In this project, we assume the latter, but we hope to restrict the amount of user input required. Specifically, we assume that the user will select a region in one image that “corresponds” with a user-selected region in another region. Our use of correspondence here refers to the idea that one of the regions could be transformed to fit into the other with a matching boundary. In other words, if we could cut a piece out of one image, transform it, and place it on top of another image without noticeable artifacts, the two regions correspond.

We concentrate specifically on computing such a new image. This cut-and-paste technique is easy if the two regions are exactly the same or even if they are proportionate. However, the problem becomes more difficult when the regions are totally different shapes. We outline a process that accomplishes this task. Notice that while we require the user to input the regions, it is possible that given good enough segmentation and identification techniques, this technique could be used autonomously to add objects to a scene.
2 Previous Work

The idea of cutting out pieces of different photographs and pasting them together into a montage has existed for quite a while. Even in the Victorian era, there are examples of this work with regular photographs. Figure 2 shows both an early example of this work and a more modern version. After World War I, interest in photo-montages really took off, and interest has been fairly constant since then. Even today, this physical cut-and-paste technique continues both in the art world and scrapbook keepers.

On the computer, the cut-and-paste technique was first developed for text. The user was allowed to select text and move it to another part of the document. The use of a “clipboard” to store graphical information was developed at Apple Computer in the early 1980s with the development of Lisa and later, the Macintosh. This clipboard was multi-purpose but didn’t allow more than one item to be stored at once. Later developments included layers for editing and compositing images, popularized by Adobe’s Photoshop.

Despite plenty of advances in image editing technology, relatively little work has addressed mostly automated image composition. There are techniques for specific tasks included texture synthesis [3] and mosaicing [5], and techniques for single image analysis and information extraction (ie segmentation, feature detection). Of course, the basic cut-and-paste technique also exists as a simply copy of bits from one image to another. There are also improvements including feathering and anti-aliasing that reduce the hard edges most pastes would cause.

![Figure 1](image1.png)  
(a) German Postcard, Anonymous, 1902; (b) The Great Cliffs of Collage Green, Sean Hillen, 1997 [2]
3 Technique

Our technique flows from a very basic outline: first, we want to determine the regions to be cut and pasted into; second, we need to determine how to transform the clipping to fit the region; and finally, we need to composite the images in a pleasing manner. The first part is the only part for which we require user input; the next two parts run autonomously.

3.1 Selection

While this is a technique taken for granted in most programs, we wrote code to perform this fairly simple operation. To control the actual points that define the selected region, we caught all mouse events and placed these points into a structure associated with the image. Selection can be efficiently accomplished by simply counting the number of lines that you pass in raster scan order. An even number implies that the pixel is outside the selection while an odd number implies the pixel is part of the selection. It is important to note that the selections have no restrictions; a selection can be in any shape and may be concave. It is defined by a free-form curve.

3.2 Transformation

In a similar method to texture mapping and other warping techniques, we need to assume some planar transformation between the two selected regions. For this project, we assumed that such transformations could be approximated by planar transformations. This turned out to be a poor choice given the way that the homography was computed. The homography was computed via the following sequence:

1. Take the set of points that defines the curve that surrounds the region, and compute the convex hull of that region. From that hull, we can compute a quadrilateral that (a) encompasses the selected region and (b) minimizes the amount of area that falls outside the convex hull and inside the quadrilateral.

2. Given the quadrilateral, we have four points that define four “corners” of the region. (We call them corners because they will tend to be on opposite sides of the region to minimize area.) Recall that we have four “corresponding” points from the region to be pasted into. From these four correspondences, we can compute the planar affine transformation. However, before we compute the correspondences, we normalize the points so that their centers fall at (0, 0) to try to maintain accurate scaling.

3. With the homography, we can now go about computing the transformed image that will be composited with the base image. To do this, we do a forward mapping of all points to determine the bounds of the new image. After this, we can perform a backward mapping on the image to generate the transformed image.

Notice that while it would be inefficient to compute a homography based on all the points in the curve, we could add extra points if we could determine their correspondences. In fact, it would be much more robust if we added a couple more points from the convex hull to allow us to compute the full transformation without assuming that it is affine. This turns out to be a major flaw in the implementation.
Another way to get around the fact that the correspondences are unlikely to fit the affine assumption is to add in a rotation calculation independent of the homography calculation. For example, we can take the top two lines of both regions, compute their dot product which is also the angle between them, and rotate one image into another. Once the transformed image is calculated, we simply rotate the image back to get it to the desired state.

### 3.3 Composition

There are a variety of methods that can be used to actually composite the images. Some of these methods are independent of the images and are only take the relative geometry into consideration while others are based on the underlying image data. We began by implementing a simple 0-1 composition whereby if the pixel was in the pasted region, it always covered the base image. That left many jagged edges, so we turned to feathering technique instead. Recall that basic feathering computes the strength of the output pixel based on its distance from the center of the region. With two images, this becomes

\[ w_{t_{\text{clip}}} = \frac{(w - |x - w|) \cdot (h - |y - h|)}{wh} \quad \text{and} \quad w_{t_{\text{base}}} = 1 - w_{t_{\text{clip}}} \]

Notice that this is based on linear distance. It may be better if we use the square root of the distance instead, especially if the region is tightly selected. We can generalize this technique to allow for arbitrary curves as

\[ w_{t_{\text{clip}}} = \left( \frac{(w - |x - w|) \cdot (h - |y - h|)}{wh} \right)^{1/n} \]

\( n \) should be set higher as the boundary around the selected object is decreased.

In addition to feathering, we investigated the use of only the boundary pixels to determine some lower shifts and cuts. Recall that this can be based on the same ideas like texture synthesis. We want to find a good boundary between the base image and the pasted region. If we can slightly shift the image to get a better fit, we might as well do it. We created the boundary by counting a given number of pixels into the region from the outside.

Other ideas that were not investigated were possibly changing the brightness or hue to better fit the image the cut region is being pasted into. Again, this should probably be a localized search in the base image, but it could help to add to the seamlessness of the integration.

### 4 Implementation

In order to implement this idea, we needed a basic environment to perform cut and paste operations. We made use of the FLTK project [4] and its widgets to create a user interface. This interface consists of a series of tabs to keep track of all open images, buttons that control file management and the various operations, and event handlers to allow selection and placement. This section provides a description of relevant information about the actual program.

#### 4.1 File Management

Each file is loaded via the “Load” button. Only tga files are supported at the moment. When a file is loaded, it receives its own tab which displays the image. A file can be saved in ppm format
via the “Save” button. Files that are closed via the “Close” button are deleted from the display and are not saved.

4.2 Operations

In order to implement the cut-paste scheme, we need three operations. The copy button copies the current selection to a buffer so that it can be transferred to another buffer. This operation copies the full bounding box and sets the alpha values of the unselected values within the box to 0. The paste button pastes the contents of the most recently copied buffer into the current image. The pasted image floats above the current image and allows the user to change it’s position. The composite operation is the heart of the code. This operation controls the actual compositing of the two images. Once an image is composited, the composition is fixed and the user cannot change the relative positions of the images. The composition also requires that a region is selected in the target image before it can proceed. In addition, a clear button is provided to reset the image deleting any pasted images and selected regions for the current image

4.3 Event Handling

When an image is opened, the image is in selection mode. That means that a user can drag the mouse to select a sequence of points that defines the region to be copied. If the user pastes an image into another image, the program switches to paste mode so the user can position the image before composition.
5 Results

Results are shown below. Notice that the feathering technique is applied for various \( n \) depending on the user’s preference. Also notice that without any knowledge about rotation, the clips can be placed in really bizarre locations. This is again due to the affine assumption. Those clips that are simply skewed are easily transformed by this algorithm.

6 Improvements

This project is only a glimpse into the world of semi-automated image editing. The simple technique to do a “smart” cut-and-paste uses relatively little information about the underlying image. We assume very little, and there are vision techniques like segmentation that would greatly improve the accuracy of the algorithm. In addition, the rotation problem should be fixed by either defining more points or determining the rotation separately.

Improvements to the GUI would include an undo feature; at present, you need to close the image and reopen it to get a clean copy back. Also, it would be nice to have finer control over the transformation. Perhaps the composite operation would produce a candidate homography, and the user could graphically modify it to improve the visual fit.

7 Conclusion

Even in two dimensions, automatic image processing is fairly difficult. This project shows that there is some hope for more automation in image processing and artwork, but also demonstrates that this is still a fairly involved process.
Figure 4: Adding a decal to the brick building
Figure 5: Changing street signs
References


