FPGA Based Hardware Emulation of Image Morphing

Final Report
CprE 583, Fall 2011
12/19/2011

Talha Ansari
Heather Garnell
Anamika Shams

Table of Contents

1. Introduction

Morphing is an image processing technique used for the visual effect of transforming from one image to another image. Ideally this is a seamless transition over a period of time. The main concept is to create an intermediate image by mixing the pixel color of the original image with another image.

The easiest way to morph one image into another, for images that are the same width and height, is to blend each pixel in one image with the corresponding pixel in the other image. Basically, the color of each pixel is interpolated from the first image value to the corresponding second image value.

Additional, more complex, image morphing techniques exist. We focused on an approach that makes use of barycentric coordinates for our project.

The steps needed for image morphing consist of many repetitive tasks. Some aspects of the process are independent such as the computation to determine color for the pixels in the morphed image. This processing can be completed in parallel and therefore is a good candidate for optimization on FPGA.

2. Scope & Objective

The scope of this project is to implement an FPGA-based hardware implementation of an image morphing application.

The objective of the project is to learn more about reconfigurable hardware. This includes the steps involved with designing and implementing a system that makes use of reconfigurable hardware, such as an FPGA.

The final design will allow two images to be sent to a microprocessor and FPGA and will return a morphed image that can be displayed on monitor or saved to the hard disk.

3. Design Description

Software Development

The main idea of our project is to blend the pixels between two images. For this project, the concept of barycentric coordinates is used in the logic to create the morphed image.

To make use of the barycentric coordinates concept, the project must contain two images that are preferably the same size and oriented such that the outline of the object is similar.

There must be points of interest on each image that can be matched up. In this project, those points are the eyes and nose of the face. Using these points, the image is divided into triangle, such as in the figure below of the kitten and cat images divided into eight triangles, as seen in the Figure below. The pixel data in each triangle of one image corresponds to the pixel data in the same triangle in the other image.

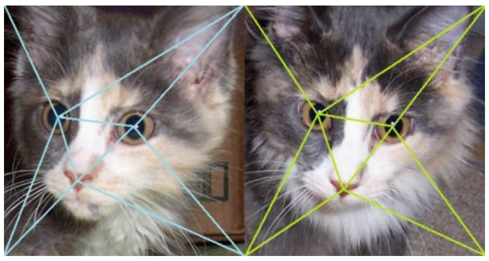


Figure 1. Triangulated images

Once the images are divided into triangles, the processing for each triangle is independent. This processing can be completed in parallel and therefore is a good candidate for optimization on FPGA.

<u>Morphing Computation Details</u>

For each triangle ($A_0B_0C_0$ in the first image, $A_1B_1C_1$ in the second image) there is a corresponding triangle $A_tB_tC_t$ for the morphed image. For this project, t = 0.5 or halfway through with image morph.

For the triangle $A_tB_tC_t$ for the morphed image, each point P_t can be determined using barycentric coordinates (α, β, γ), which are derived from the ratio of the smaller triangle formed by P (PAB, PAC, PBC) over the area of the entire triangle.

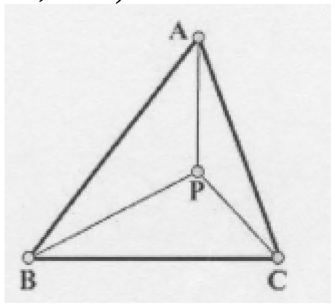


Figure 2. Final image after morphing

The equations to determine a barycentric coordinate (α, β, γ), is shown here.

$$\alpha = \frac{Area(PBC)}{Area(ABC)}, \quad \beta = \frac{Area(PAC)}{Area(ABC)} \quad \text{and} \quad \gamma = \frac{Area(PAB)}{Area(ABC)}$$

Figure 3. Equations for α, β, and γ

Using α, β, γ the corresponding points (and their pixel color, I_0 and I_1) in triangle $A_0B_0C_0$ and $A_1B_1C_1$ can be determined as shown

$P_0 = \alpha_t A_0 + \beta_t B_0 + \gamma_t C_0$

$P_1 = \alpha_t A_1 + \beta_t B_1 + \gamma_t C_1$

The color of point P_t can be determined as shown:

$I_t = (1-t)I_0 + t\, I_1$.

By doing this procedure for all points in all triangles *in* the morphed image, the pixel color for the entire image can be determined by an interpolation of the initial color and final color.

The image below shows the final morphed image that is output from the application developed for this project.

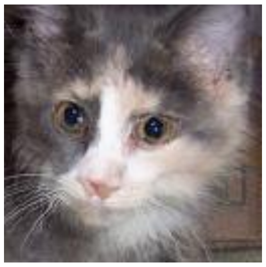


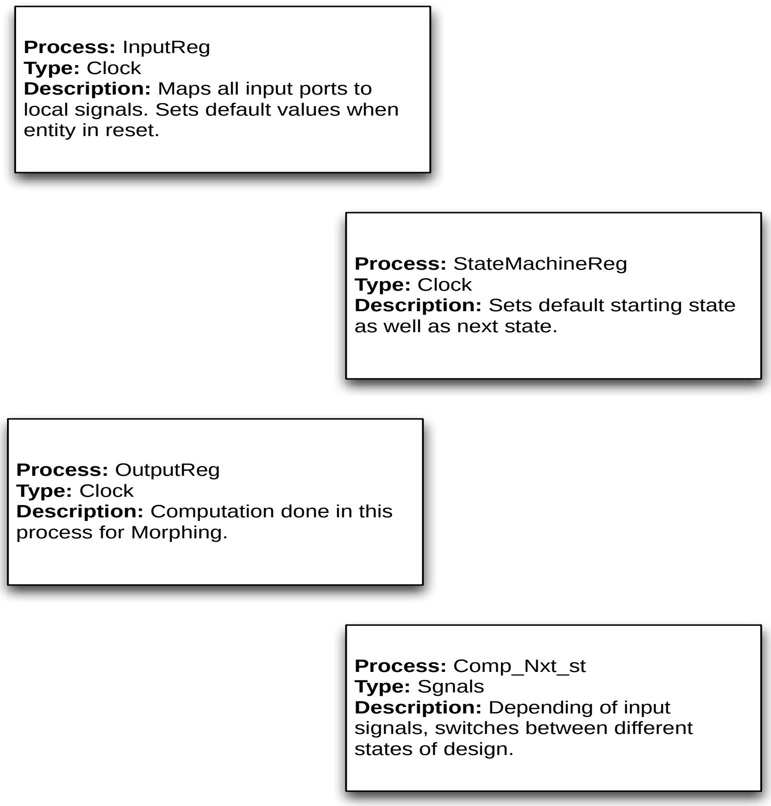
Figure 4. Final image after morphing

We created a document repository in Google docs, or project website containing the initial project proposal, weekly status reports, the draft of the final project plan presentation and final report. This helped to allow each group member to make updates at any time.

Hardware Development

The MP3 software architecture was reused for the project with the intent that integration of the image morphing functions will be much easier. Also, by using MP3 architecture the results would have easily been displayed on the monitor. APU_INV entity was reused, with slight modifications from MP3. The state machine architecture was designed to perform two successive loads followed by a store. The first load was used to store image data from the first image while the second load was used to store data from the second image. Both loads were 128 bits long, with bits 120 to 127 containing blank bits. The primary goal of each load was to store data of 5 pixels, each of which contains 24 bits of color element data. Therefore for 5 pixels total number of bits came out to be 120, hence last 8 bits were ignored.

The hardware design was broken up into 4 processes. Figure 5 below outlines each of the processes. InputReg process is where all inputs are mapped to local signals with in the entity. If reset is detected, all signals are set to “0”, otherwise they are mapped to the input ports. The second process, StateMachineReg is where the next state of the state machine is determined. As the figure states, this was a clocked process used to help sync the hardware design flow. OutPutReg is also a clocked process where image merging is carried out. The initial goal was to take average of each element from each of the two images, i.e pixel 1 red from image 1 was added to pixel 1 red from image 2 and divided by two. The last process of the hardware design is Comp_Nxt_St. This is not a clocked process and depended on bit changes to signals as shown in the apu_inv.vhd file.


Process: InputReg
Type: Clock
Description: Maps all input ports to local signals. Sets default values when entity in reset.

Process: StateMachineReg
Type: Clock
Description: Sets default starting state as well as next state.

Process: OutputReg
Type: Clock
Description: Computation done in this process for Morphing.

Process: Comp_Nxt_st
Type: Sgnals
Description: Depending of input signals, switches between different states of design.

Figure 5. Hardware design high-level block diagram.

As described earlier, the state machine for checking the instruction type is similar to the MP3 design, except the state machine is designed to track two consecutive loads. Figure 6 shows the state machine used for this project. The design defaults to INST_TYPE state where it waits for a valid instruction. If a load instruction is detected, the next state is set to WAIT_WBLV where it waits for either a “loadvalid” or a “writebackok” input, whichever comes first. Depending on which signal is detected first, the next state is either set to WAIT_LV or WAIT_WB. From here the next state is set to LOAD_DATA where data from the first image is stored. At this point local signal LDCNT is set to “1” and the next state is set to WAIT_LV to get ready for data from second load. LDCNT here is used to track the number of consecutive loads. Once the second load arrives, LDCNT is set o “2” and image merging takes place inside OutputReg. The next state is set to WAIT_INST, where the state machine either waits for the store or another set of load instructions. It is important to note that if another set of load instructions are detected before the store, result from the first two loads will be lost.

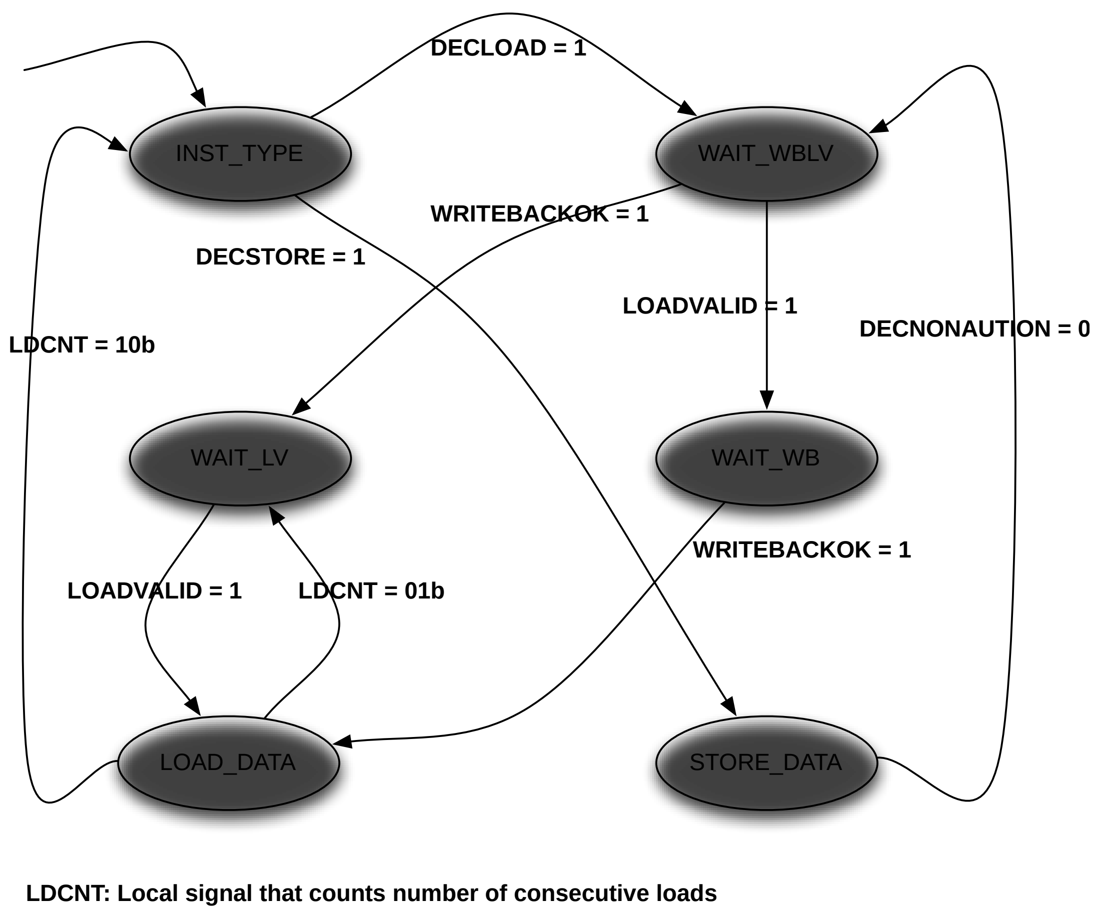


Figure 6. Hardware design state machine.

The hardware section was designed and tested with the “TestApp_Peripheral” from MP3. The file was slightly modified with two sets of two loads followed by a store. The design was tested out in simulation. A couple of run time errors were encountered where division was carried out and so this part of the code was commented out. The goal was to either do the division in software or to come back and fix the error after a successful integration had occurred.

4. Design Integration

Since the morphing algorithm was complex, we decided to break the integration into multiple steps. The main morphing algorithm was coded on a windows based machine and the hardware was developed on the Xilinx-3 machine. The goal was to design both sections independent of each other to keep the complexity of the design simple. The next steps were to integrate the software on the ML507 board without the use of the VHDL. Once everything was working, VHDL code for pixel merging would have been off loaded to the hardware. However, a critical mistake was made in this design step. We highly under estimated the integration step and spent majority of our time on the software and hardware designs. To much of our disappointment we were unable to integrate the morphing on the ML507 board. Independently, both the software and hardware features work but once combined together on the hardware we kept running into one roadblock after another. Our integration via a modified echo.c didn’t work out as expected. Our work schedules also didn’t help the cause, as most of us couldn’t meet regularly to discuss issues and come up with an appropriate solution.

5. Project Status

We were able to successfully implement the morphing algorithm in software and on any windows or linux-based machine. We were also able to code and test majority of the hardware design. However, both of the design blocks could not be combined to work on the hardware board.

6. Lessons Learned

- Use of chat programs was very helpful to keep communication lines open

- Use of tools to keep documentation in a common location is helpful (Google docs and Google sites)

- Checking for integration issues early on is important

7. Conclusion

Completion of the image morphing implementation with the FPGA was the ideal goal of the project. Another goal of the project was to speed up the processing time for the final morphing image to display to the monitor. Also, code reusability was considered while creating the project to enable this project to be reused for any kind of the image morphing project with minimal update.

We ran into more issues that expected when we tried to integrate which is captured in the lesson learned section. If the integration effort of the C code with the FPGA would have started earlier as we planned then we expect we could have resolved the issues we have currently to get our project to work on the hardware. Though our integration via a modified echo.c didn’t work out as expected, we did meet many of the goals of the project.

8. References

[1] Ching-Kuang Shene. "Barycentric Coordinates and Morphing," Department of Computer Science. Michigan Technological University. April 19, 2003.

[2] T. Balercia, A. Zitti, H. Francesconi, S. Orcioni and M. Conti. "FPGA Implementations of a Simplified Retinex Image Processing Algorithm." IEEE 2006.

[3] CprE583 Course Website, http://class.ece.iastate.edu/cpre583/.