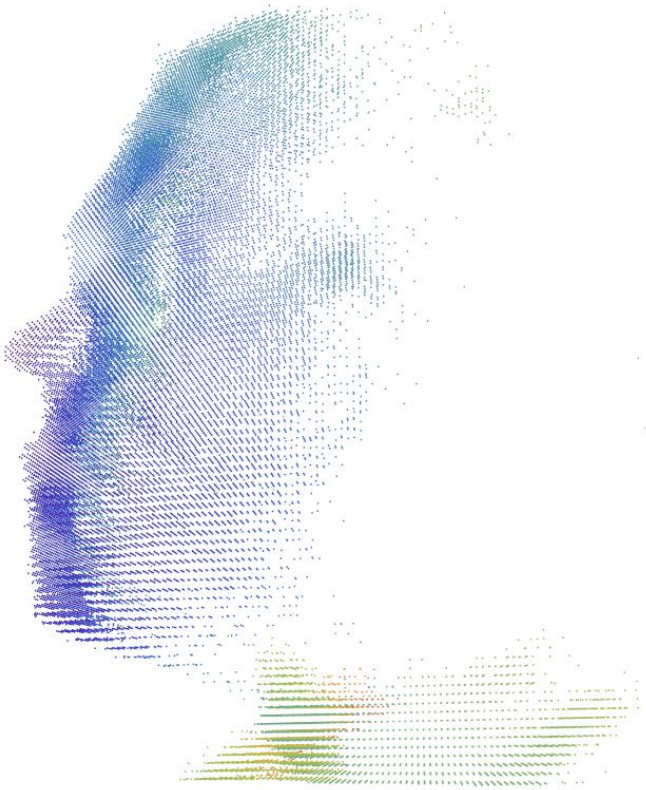
CSCE 401: SOFTWARE ENGINEERING

Saving Face

Elegant Facial Recognition

Andrew Mason (aemason@alaska.edu)
Keith Schneider (1900Geek@gmail.com)
Jacob Dempsey (Jacob@JacobsChaos.com)



CONTENTS

Overview of Requirements and Features:

Our project requires the use of an Intel Creative Camera (*Figure 1*) for the collection of three dimensional data points used in all of the functionality of our project.

Figure 1: Intel Creative Camera.

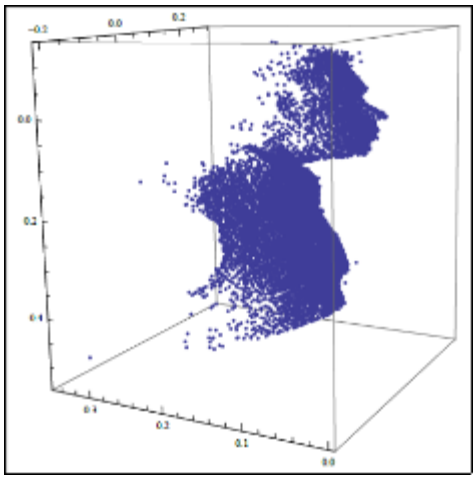


Figure 2: 3D model data.

Saving Face features 3D model reconstruction and comparison of a person's face (Figure 2). The comparison is performed by mapping retrieved depth values to a 3D model based off of the yaw, pitch, and roll of the face centering the tip of the user's nose to the origin of the 3D coordinate system (Figure 3). The model data is collected over a predefined number of frames to give a better estimate of the user's face in the model. The user's initial model, or reference model, is stored in a database, and the identification models are then compared to the models in the database to find the closest match.

Intel Creative Camera Specifications

Intel® Developer Kit (2013)

Maximum RGB Video Resolution: HD 720p (1280x720)
Maximum IR Depth Resolution: QVGA (320x240)
Frame Rate: 30 fps
FOV (Diagonal): 73°
Range: 0.5ft to 3.25ft
Size: 4.27"x2.03"x2.11"
Weight: 9.56 oz
Power: Single USB 2.0 (power < 2.5W)
Dual-array microphones
RGB + Depth frame sync

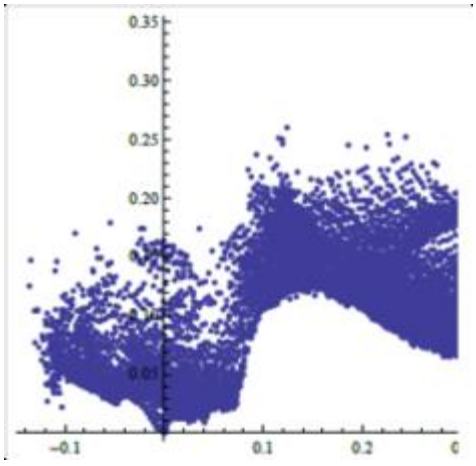


Figure 3: data centered at the origin.

Recommended System Configuration

Intel® Developer Kit (2013)

PC with 2nd Generation or newer Intel® Core™ processor
Windows® 7 with Service Pack 1 or higher
4 GB system memory
USB 2.0 port

FEATURE LIST:

Story Name	Description	NUTS
Fixed Point Orientation	Find X Y Z of at least three fixed points; Or 1 and Yaw Pitch and Roll	2
Transform Module	A module that applies the transform to bring the points into a common plane	6
Build Model Module	A module that takes the transformed data and makes a model representation	5
Compare Module	A Module that takes a stream and compares it to a list of Models and returns a ranking	5
Load Module	Load a model from a file	2
Save Module	Saves a model to a file	2
Debug Terminal	Displays Helpful Information	2
Save Depth and Color Video	A module that saves video for later re-use	4
Display Streams	A module that displays depth and color Streams	1
User Information Input	A GUI interface for inputting model Information	3
Save Image Data	A module To Save a User Photo for Later Display	2
Load Color/Depth Video	A module to load video playback for algorithm refinement and debugging	4
Load User Image	A module to load and display a user image	2
Activity: Model Collection	Collect model data to test accuracy of implementation	3

Development Estimates and Actual Progress:

Our estimates were relatively accurate for most sections of our project. Toward the end, some were altered by a few points. The burn up chart (*Figure 4*) accurately represents the team's completion of NUT's during this process. The team's iterations were not truly defined by equal amounts time so it was difficult to encapsulate those iterations in the chart. Much of the first few iteration were spent developing the framework, and development across several user stories. The amount of programming hours where not evenly distributed across iterations, with the beginning and the end receiving the most.. The burn up chart increases as we noticed we had to revisit a previous user story (increased NUT's), and we switched from a debug terminal to a GUI. There also was a function that was not implemented in the PC SDK that we were relying on, realizing that this method was not implemented we had to create our own function to convert color coordinates to depth coordinates.

Our estimates mid-semester showed us that we were not going to finish the project on time. However thanks to complete devotion at this critical point we tripled the amount of work being done. The estimates taken several weeks after showed us that we could finish the project on time.

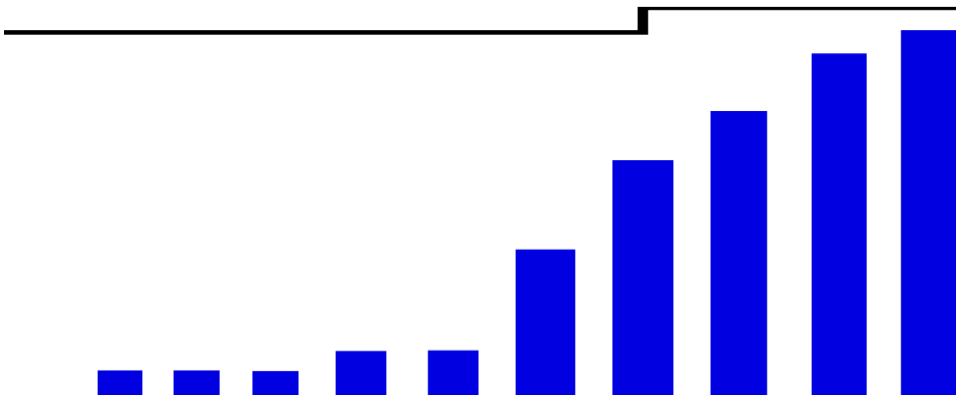


Figure 4: Saving Face burn up chart.

DESCRIPTION OF THE DESIGN:

Keith Schneider had some great foresight into the project. He designed early prototypes of the algorithms we needed to align the depth images for all models into a byte array. The functions used output from the Creative Gesture Camera that was loaded into Mathematica for testing. Here is some pseudo code of the model building algorithm:

GENERIC ALGORITHM FOR THE MODEL CONSTRUCTION

```
While[Frames < N]
      Read in byte stream
      Get Fixed Point, and Yaw Pitch and Roll From SDK
      Calculate Transformation Matrix Based On Yaw Pitch and Roll
      Determine Relative Location Of Head
      For[Vertices]
            Apply Linear Transform to bring point relative to origin
            Apply Rotational Transform To Align Head
            Map Point to Model Byte[]
            Increase that point by one
      Save Model
End
```

In order to compare models it's essential that all of the models line up the same in our vertices array. Using the yaw, pitch, and roll we calculated a transformation matrix that was applied to all vertices to make all models have the same degree of relative rotation. We also centered the nose of each model at the origin of the 3D plane. With our models in the same orientation, we are then able to compare points from the user with points of the models in our database. The more points that match, the more likely the user is the particular model in the database.

ONE GENERIC ALGORITHM FOR THE MODEL COMPARISON

```
score[Len(Model)]
For[each Vertex v]                          //Vertex is found
      For[each Model m]                     //For All Models
            If[m[v] > 0]                    //contains a data point
                  score[m] += m[v]          //count of vertex hits
For[each score s]
      s[m].pct = s[m].total/m.total
return score;
End
```

THIS WAS AN INITIAL CONCEPT. THE FLEXIBILITY OF THE MODEL ALLOWS FOR MULTIPLE IMPLEMENTATIONS THAT RANGE IN: COMPLEXITY, SCALABILITY, RUNNING TIME (FROM REAL TIME TO EVERY FEW SECONDS), AND ACCURACY.

Description of the Finished Product:

Saving Face is an application designed to build and compare 3D models using the Intel Creative Camera, by taking depth images of a user's face and building a comparable 3D model of them. The software is designed to test the feasibility of using the Intel Creative Camera for facial recognition.

Data Collection

When creating a new model for the database, the user is asked for basic information used to facilitate identification and for contacting the user to ask them to participate in future tests. The information collected includes the user's preferred salutation; first, middle, and last names; a suffix if used; the user's email address; and the persons gender. A video (color and depth) is also stored with user consent to facilitate testing multiple evolutions of recognition algorithms. A picture of the user is then taken and stored with the model for future comparison.

Model Creation

The software captures and combines a set number of depth frames from the camera into a three dimensional model of the users face. The yaw, pitch, and roll of the users head, as well as the tip of the user's nose are then calculated. This date is then used to calculate a transform matrix to orient the model so the nose is centered at the origin, and the head is set to a common orientation. The model data is stored in a byte array with the vertices stored in discrete array of fixed points rather than continuous data.

Model Storage

When creating a new model for the database, the model, associated data, and a unique identifier are stored in a database for convenient storage and access.

Model Comparison

The software creates an identification model of the user and compares the identification model to the reference models stored in the database. Each vertex of the identification model is compared to the vertices of the reference models. The number of matches are recorded and divided by the number of vertices per model to give a matching percent. The higher the percent match, the more likely the user was found in the database. A percentage threshold for a positive identification has not been established at this time.

Description of Tests:

The project employed automated testing via the Visual Studio 2012 Professional Suite for unmanaged C++ code. Throughout the development process, we did do some testing first (where applicable), but much of the testing was based on expected outcome after the fact. The automated testing system was an excellent way to track bugs and fix them. Testing first gave us insight on how to avoid bugs in a given function during development, especially regarding edge cases. Testing first may feign a delay of progress but it saved a lot of time in the long run. We have multiple tests for functions and did a good job at determining a function was working based on an correct output with given inputs.

Description of the Process:

We started the project using the Agile Method. We met with the client, created user stories, estimated NUT's for each story, and noted dependencies some of the stories had with other stories. We then met with our client again to set priorities for each story, and began working on those stories. As the work progressed, particularly when we realized we were not going to be able to complete the project based on our current number of hours being dedicated to the project, our client meetings became more informal short visits with our client as time permitted. The informal meetings were typically immediately after our client finished lecturing to his CSCE 401 class. We wrote and performed functionality tests on each section of the code as we progressed in the project, which proved invaluable in finding bugs in our implementation. However, toward the end of the project, we placed too much emphasis on completing the project; and began moving on to new user stories without first performing the corresponding client acceptance tests. With time constraints at our backs, we had to make the largest jumps we could in the shortest amount of time, however functional testing of the code remained a priority. Thus we did stick to the Agile principle of it is not done until it is tested.

Who Worked on What

Keith Schneider

Keith stepped up early in the project as the project lead. Throughout the project stayed on point as the project analyst, project lead, concept designer, lead programmer and lead tester.

Andrew Mason

Andrew laid the early ground work for the GUI in C#, added ideas and insights to the design and algorithms and functioned as an assistant programmer, assistant tester, and assistant designer.

Jacob Dempsey:

Jacob began implementing the GUI in C++ using MFC. After it was found that the MFC libraries had some incompatibilities with the PCSDK libraries, he started laying the ground work for implementing the GUI using the Win32 API. Throughout the project, he continued to work as an assistant programmer and lead GUI designer. Due to time constraints, the GUI is much more simplistic in design than was desired.

Overall Experience:

One of the biggest challenges was creating a constant schedule for our project. While other classes interfered we found that our hours of work varied week by week. Creating a definitive schedule of what needed to be done and how many hours should be done in an iteration for each team member would have helped a lot. We learned many new programming techniques and also a new and wonderful(though often frustrating) SDK. This was a very large project and in hind sight, though we met often and discussed progress openly. Team code reviews would have served to keep members on task, allowed for more direction for individual tasks and better understanding of the code implementation.

REFERENCES

Intel Corporation. (2013) “Creative Intel Developer Kit”. Retrieved from http://download-software.intel.com/sites/default/files/article/325946/creativelabs-camera-productbrief-final.pdf December 2, 2013.