

Morphing

Joaquín Pérez Araya

Departamento de Ciencias de la Computación

Universidad de Chile

Santiago, Chile

joaquin.perez.a@ug.uchile.cl

Resumen—En este documento se aborda el tema del *morphing*, una técnica de mezclado de imágenes la cual consiste en crear un proceso de metamorfosis entre una imagen y otra, adicionalmente se muestra una implementación realizada con resultados utilizando diferentes imágenes.

INTRODUCCIÓN

El *morphing* es un tipo de efecto visual el cual se produce al cambiar una imagen a otra con un efecto de metamorfosis, actualmente se utiliza principalmente para el entretenimiento, en este documento se implementará el algoritmo descrito por Beier-Neely [1], que consiste en utilizar líneas de correspondencia entre la imagen de partida y la imagen de destino para describir la forma en que la mutación se va a llevar a cabo. El documento primero se describirá el diseño en la implementación, qué partes del algoritmo se implementaron y de qué forma, luego se probará éste código con diferentes dos muestras con distintos tipos de líneas para observar qué líneas son las mejores para este algoritmo. Para finalizar con las observaciones y conclusiones de dichos experimentos.

DISEÑO E IMPLEMENTACIÓN

El código implementado define 3 funciones principales las cuales están implementadas en `morphing.py`: `warp`, `morph` y `create morphing video`: La transformación de una imagen según un par de conjuntos de líneas (`warp`), la creación de múltiples imágenes que son las que forman parte del morphing (`morph`) y finalmente la creación, a partir de las imágenes calculadas en el proceso anterior el vídeo que muestra el cambio de las imágenes a lo largo del tiempo `create morphing video`. También se dispone de un módulo adicional llamado `util.py` el cual contiene funciones auxiliares, tales como el cálculo de distancias y en general la aplicación de fórmulas detalladas en el artículo [1].

Warpping

Para el warpping se utiliza el algoritmo propuesto en el artículo [1], que está especificado en el Algoritmo 1, el cual crea una imagen vacía con las mismas dimensiones de la imagen fuente, y itera sobre ésta, calculando qué colores debería tener según las líneas de correspondencia y la imagen fuente.

La implementación usa el algoritmo en su versión *inverse mapping*, es decir durante la ejecución se recorre la imagen de destino calculando qué pixeles de la imagen original deberían estar allí utilizando interpolación bilineal de los pixeles más

Algoritmo 1: Warpping

```
Data: image imagen fuente, linessrc líneas fuente,  
linesdst líneas de destino.  
Result: Imagen que corresponde a la imagen fuente  
modificada según las líneas dadas.  
destinationImage = zeros(shape(image))  
for Pixel X in image do  
    DSUM = (0, 0)  
    weightsum = 0  
    for Line PiQi in linessrc and P'iQ'i in linesdst do  
        u, v = calculate_u_v(X, Pi, Qi)  
        X' = calculate_X'(u, v, P'i, Q'i)  
        Di = X' - X  
        weight = calculate_weight(X, u, v, Pi, Qi)  
        DSUM += Di * weight  
        weightsum += weight  
    X = X + DSUM/weightsum  
    destinationImage(X) = image(X)  
return destinationImage
```

cercanos de la imagen original, este método ofrece más simplicidad dado que se conoce los pixeles de destino de antemano por lo que el único inconveniente es el caso de que se requieren pixeles fuera de la imagen original por lo que se interpola según los pixeles ya calculados anteriormente, en cuyo caso sería:

- Si se está en el primer pixel de la imagen, el de la esquina superior izquierda, éste se calcula como el pixel del mismo punto de la imagen de origen.
- Si se está en la fila superior, se calcula usando el pixel anterior calculado, el de la izquierda de éste.
- Si se está en la columna de la izquierda, se calcula usando el pixel de la fila superior.
- Si se está en la columna de la derecha, se calcula utilizando los 3 pixeles que están cercanos a éste: Superior izquierdo, superior e izquierdo.
- De otra forma se calcula utilizando 4 pixeles cercanos: Superior izquierdo, superior, superior derecho e izquierdo.

En el módulo de utilidades, están implementadas las funciones para el cálculo de u , v , X' y $weight$ según como se indica en el artículo. Para los parámetros de cuanto peso tiene

una línea respecto al punto se eligió: $A = 1$, $B = 1$ y $p = 0,5$. No hay ningún cambio respecto a la implementación de éstas formulas.

Morphing y Video

Para la creación del conjunto de imágenes que forman el *Morphing* completo se realiza lo siguiente: se itera por el número de imágenes n que uno quiere incluir en el vídeo, se calcula el valor de entrelazado t que va desde 0 hasta 1 de $1/n$ en $1/n$ pasos, según el grado de avance del *Morphing*. Dado un t de éste proceso, para crear una imagen, se realiza un cross-fade entre dos imágenes: el *warp* de la imagen fuente con sus líneas y la interpolación de las líneas fuente hacia las líneas de destino según t , es decir $t \cdot \text{lines}_{src} + (1-t) \cdot \text{lines}_{dst}$, y el *warp* entre la imagen de destino con sus líneas y la interpolación de las líneas de destino hacia las fuentes según t , que es $(1-t) \cdot \text{lines}_{src} + t \cdot \text{lines}_{dst}$. Al final se añade a la colección de imágenes la ponderación de t por la primera imagen más $1 - t$ de la segunda.

Luego de finalizado la creación de la colección de imágenes del morphing, se utiliza la librería de OPENCV para formar un vídeo con éstas, en esta implementación, los vídeos generados están a 10 cuadros por segundo y en formato .avi.

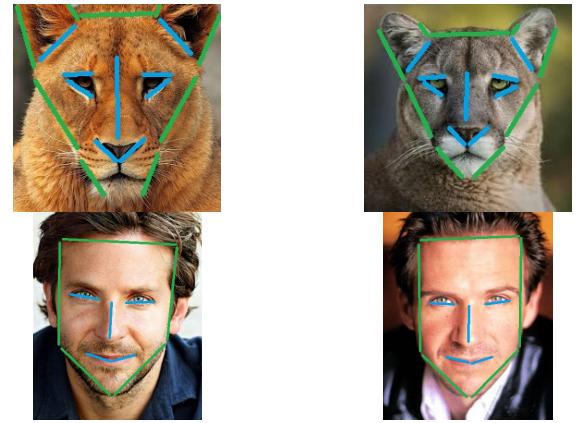


Figura 1. Imágenes usadas para la experimentación, están marcadas las líneas utilizadas para el Morphing, las verdes corresponden a contornos de la figura mientras que las azules corresponden a características intrínsecas de la imagen. Las imágenes de la izquierda corresponden a las de fuente mientras que las de la derecha corresponden a las de destino.

Para cada prueba, se ejecutó la implementación requiriendo 50 imágenes en total, de las cuales, por motivos de simplicidad, sólo se presentarán la 1^{ra}, 10^{ma}, 20^{va}, 30^{va}, 40^{va} y 50^{va} imagen de dichas ejecuciones, sin embargo dentro del directorio de la implementación estarán las pruebas con sus respectivas imágenes y vídeos. De aquí en adelante, se mencionará como imágenes verdes, las imágenes que fueron creadas a partir del conjunto de líneas verdes, e imágenes azules de forma análoga.

Imágenes de Caras

EXPERIMENTACIÓN

Para verificar la precisión de la implementación, se utilizó un caso simple que consiste en transformar un círculo rojo en un cuadrado, una simple transformación. En la sección de Anexo se encuentra el detalle respecto a la obtención de este ejemplo.

Se realizaron pruebas con dos pares de imágenes, donde se crearon las líneas a mano y después se ejecutaba el programa para revisar los efectos: Una con un par con dos felinos distintos, un puma y una leona, y la otra con un par de un mismo actor. Para prueba se crearon 3 conjuntos de líneas, uno con líneas de contorno (véase las líneas verdes en la figura 1), uno con líneas de expresión (las líneas azules en la misma figura) y por último uno con ambas líneas en conjunto.



Figura 2. Imágenes creadas con la implementación y sólo las líneas verdes. El progreso va de izquierda a derecha de arriba hacia abajo.



Figura 3. Imágenes creadas con la implementación y sólo las líneas azules. El progreso va de izquierda a derecha de arriba hacia abajo.



Figura 4. Imágenes creadas con la implementación y con ambos pares de líneas. El progreso va de izquierda a derecha de arriba hacia abajo.

En este caso, el conjunto que trae mejores resultados visuales es el Verde, dado que las líneas azules producen tanto cuando se usan solas y combinadas un mal efecto de transición en los ojos de la imagen, quedando meramente como sólo un fade. Y como se discutirá más adelante en la sub-sección de errores, la elección de las líneas no fue sencillo.

Imágenes de Felinos

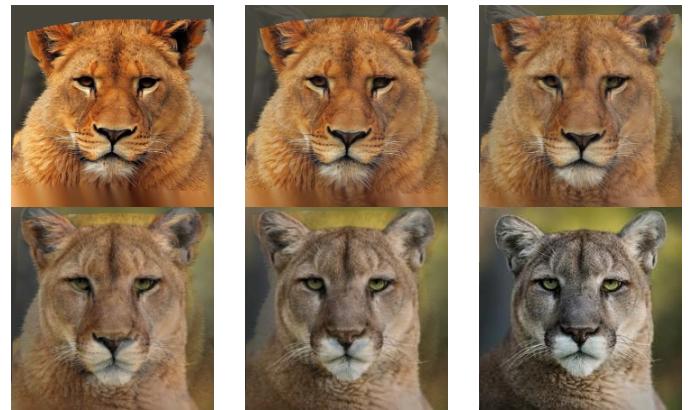


Figura 5. Imágenes creadas con la implementación usando sólo las líneas verdes. El progreso va de izquierda a derecha de arriba hacia abajo.

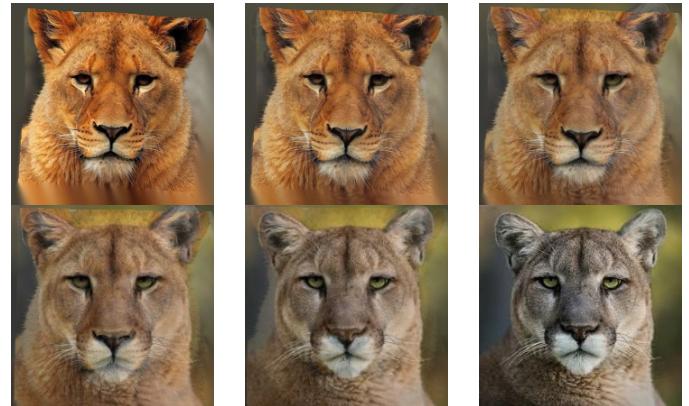


Figura 6. Imágenes creadas con la implementación usando sólo las líneas azules. El progreso va de izquierda a derecha de arriba hacia abajo.

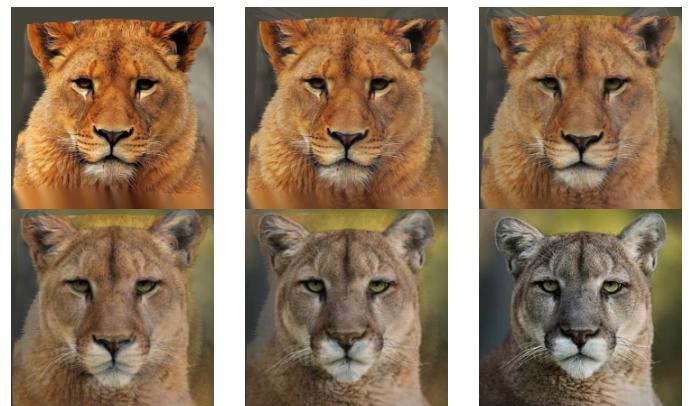


Figura 7. Imágenes creadas con la implementación usando todas las líneas. El progreso va de izquierda a derecha de arriba hacia abajo.

Con el conjunto de imágenes obtenidas con los diferentes experimentos se observa que en las imágenes creadas usando las líneas verdes (de contorno), la imagen inicial se dobla para hacer calzar las orejas de ambos felinos, creando así una apariencia de las orejas más armoniosa que en el caso de las imágenes creadas utilizando sólo las líneas azules, sin embargo, el conjunto de imágenes verdes presenta discordancias con las facciones de la cara de los felinos, éstas son más aparentes en la cuarta imagen (la primera de la segunda fila), donde se evidencian éstas en ojos y en nariz. Estos dos detalles no están presentes en las imágenes que usan todas las líneas.

Errores

Durante el desarrollo de los experimentos empezaban se obtuvieron resultados dignos de notar y mostrar:

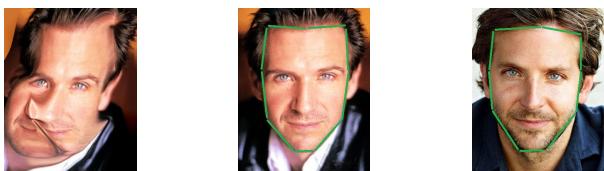


Figura 8. Morphing creado con las imágenes y líneas de la derecha a ésta.

Se puede evidenciar en el anterior ejemplo, que existe la necesidad de utilizar una cantidad mínima de líneas para indicar un mismo sector. Dado que el alto número de líneas para describir el borde de ambas caras causa altas anomalías.

Dentro de la experimentación queda en evidencia que la elección de líneas no es trivial, considerando la figura, ahí que la elección de líneas muy cercanas entre unas y otras afectan gravemente determinadas zonas, en el caso de la figura anterior la zona de los ojos.



Figura 9. Morphing creado con las imágenes y líneas de la derecha a ésta.

Estos errores permitieron realizar una mejor selección de líneas para las pruebas mostradas en la primera parte de esta sección.

CONCLUSIÓN

El uso de líneas de contorno (verdes) tiende a ser más efectivo al momento de usar *morphing* en caras que el uso de líneas de características (azules), sin embargo la falta de líneas características puede crear defectos visuales al momento de crear una animación. Por lo que es recomendable usar líneas de contorno para las figuras más una cantidad mínima de líneas de características. No es trivial la elección de líneas para la ejecución del proceso de *morphing*, si se elige de forma errónea

o usando muchas líneas pueden ocurrir errores como los vistos en la sección de experimentación, también el uso desmedido de líneas provoca que el tiempo de ejecución sea mucho más alto, por lo que reduce increíblemente el rendimiento, por lo que esta técnica sería más útil implementarla en lenguajes compilados como C o Java, así se podría ejecutar el algoritmo en tiempos menores, dado que en la implementación actual, crear la animación de las imágenes toma alrededor de una hora o más.

BONUS

Durante el proceso de creación de este trabajo, se encontraron ejemplos extra con sus líneas incluidas. Dado que no eran lo pedido como tarea, no fueron incluidos dentro de la experimentación, pero vale la pena mencionarlos, en la carpeta extra, del directorio del código se encuentran dos casos encontrados en internet:

- Dos imágenes de caras que son las caras que se estudian en el artículo del algoritmo implementado, contiene dos conjuntos de líneas, uno con 30 líneas y el otro con una cantidad abismal de líneas (42), al ser poco eficiente la implementación, ¡la ejecución del conjunto de líneas más grande toma alrededor de 8 horas!. Esta fue obtenido de <https://www.cs.toronto.edu/~mangas/teaching/320/assignments/a4>, que está dentro del código para la revisión de la tarea dada.
- Un círculo y un cuadrado, este caso se utilizó para verificar la implementación dado que sólo contiene dos líneas, es simple para verificar los errores de ejecución. Este fue obtenido de http://cns.bu.edu/~oph/cs580_assign1/p1.html.

REFERENCIAS

- [1] T. Bier, S. Neely. Feature-Based Image Metamorphosis, Computer Graphics 1992-07