Proyecto final de Visión Computacional II: Síntesis de Textura

Francisco Alfonso Alba Cadena

24 de Noviembre de 2003

Contenido

Capítulo 1

Introducción

1.1 Síntesis de textura

La textura es una característica visual de un segmento de imagen que lo identifica como perteneciente a una clase en particular. A cada una de estas clases le corresponde una interpretación física-visual distinta, tales como: cabello, madera, papel, tejido, etc. La Figura 1.1 muestra algunos ejemplos de imágenes que son consideradas texturas. La Figura 1.2 muestra imágenes que normalmente no serían consideradas como texturas. Es posible representar una textura en términos de las interacciones, tanto espaciales como de intensidad, entre los pixeles de una imagen digital. Si se logran modelar dichas interacciones, entonces es posible tanto generar (sintetizar) la textura correspondiente, como segmentar esa misma textura en otras imágenes.

Los métodos de síntesis de textura pretenden por una parte obtener un modelo matemático a partir del análisis de la imagen de una textura dada, y por otra parte, generar nuevas texturas a partir del modelo, que sean similares a la textura original. Este modelo se muestra en la Figura 1.3. Existen también métodos de síntesis que generan texturas partiendo de modelos abstractos (por ejemplo, modelos físicos) en los cuales no se parte de una imagen inicial.

Para este proyecto nos concentramos en la síntesis de texturas mediante procesos estocásticos, en particular, campos aleatorios de Markov. En estos procedimientos se parte de una textura inicial (original) para obtener un modelo con el cual será posible generar texturas similares de cualquier tamaño.

1.2 Objetivos

El objetivo principal del proyecto consiste en estudiar a fondo e implementar alguno de los algoritmos propuestos en los artículos de referencia. El algoritmo que encontramos mas atractivo para su implementación es el presentado por

Figura 1.1: Ejemplos de texturas.

Figura 1.2: Ejemplos de imágenes que no son texturas.

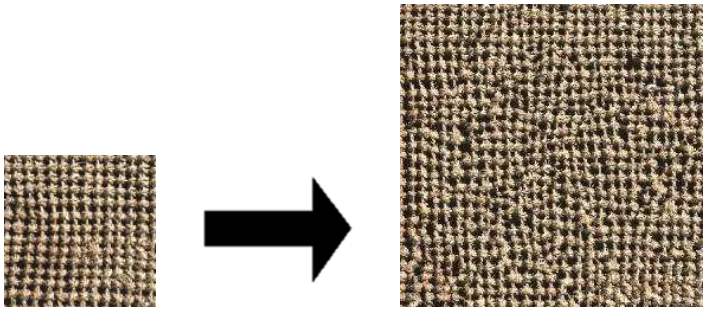

Figura 1.3: Proceso de síntesis de textura. A partir de una muestra se genera una textura de mayor tamaño.

Li-Yi Wei [1] en su tesis de doctorado "Texture Synthesis by Fixed Neighborhood Searching". También se estudian dos métodos mas: uno propuesto por Rupert Paget y Dennis Longstaff [4], y otro propuesto por Michael Ashikhmin [3]. Finalmente se pretende realizar una comparación de los tres algoritmos.

El algoritmo de Wei fué probado con múltiples ejemplos para determinar la generalidad de este; es decir, el rango de texturas para las cuales el algoritmo funciona bien, y ejemplos en los cuales el algoritmo tiene un desempeño pobre.

1.3 Aplicaciones

La síntesis de textura tiene diversas aplicaciones en graficación (renderizado, animación), procesamiento de imágenes (compresión, restauración) y visión computacional (segmentación, reconocimiento, clasificación).

En este trabajo nos concentraremos simplemente en las técnicas utilizadas para generar una textura de tamaño arbitrario a partir de una muestra dada. Para un panorama mas detallado de las aplicaciones se sugiere revisar la tesis de Wei [1].

Para las pruebas realizadas se utilizaron como texturas fuente algunas de las texturas mas representativas tomadas de la página web de Rupert Paget [13]. Cabe mencionar que la mayoría de estas texturas son fotografías de texturas reales tales como piedra, pasto, flores, nubes, tierra, etc. No tiene mucho sentido utilizar las técnicas estudiadas en texturas generadas de manera artificial mediante algún algoritmo, ya que dicho algoritmo probablemente sería capaz de generar de manera mas eficiente la textura que deseamos sintetizar.

1.4 Resumen del reporte

Este reporte comprende cinco capítulos, de los cuales el primero corresponde a esta introducción. Los capítulos 2, 3 y 4 explican los tres algoritmos revisados, mientras que el capítulo final menciona las conclusiones a las que se llegó.

En el **Capítulo 2** se revisa el algoritmo de Wei en su forma básica. Se mencionan las posibles optimizaciones de dicho algoritmo, y se muestran los resultados de algunas de las pruebas realizadas.

En el **Capítulo 3** se revisa el algoritmo de Michael Ashikhmin, el cual es una modificación al algoritmo de Wei. Se mencionan las principales diferencias entre ambos algoritmos y se muestran algunos resultados obtenidos.

En el **Capítulo 4** se estudia el algoritmo de Paget, también en su forma básica, mencionando sus ventajas y desventajas. Se muestran varios resultados obtenidos por el autor del algoritmo, comparándolos con los resultados de los otros algoritmos.

Finalmente, el **Capítulo 5** muestra nuestras conclusiones. Se hace una comparación de los tres algoritmos y se mencionan algunas ideas acerca de los mismos.

Capítulo 2

Algoritmo de Wei-Levoy

2.1 Introducción

El algoritmo propuesto por Li-Yi Wei en su tesis de doctorado "Texture Synthesis by Fixed Neighborhood Searching" [1] es capaz de sintetizar una gran variedad de texturas de manera muy eficiente. El algoritmo no produce buenos resultados en ciertos tipos de texturas, sin embargo, puede ser fácilmente modificado para solucionar algunos de los problemas que presenta. Este algoritmo también se presta para ser optimizado de varias formas, lo que lo convierte en el uno de los algoritmos de síntesis mas rápidos.

La idea básica en el algoritmo de Wei-Levoy (o simplemente, WL) es muy simple. Se toman como entradas una imagen de una textura y una imagen con ruido. El procedimiento consiste en modificar pixel a pixel la imagen de ruido para lograr que se parezca a la textura de entrada (Figura 2.1). Es posible lograr que las texturas generadas puedan ser utilizadas como mosaicos, manteniendo la continuidad en los bordes de manera toroidal.

En este capítulo se presentara el algoritmo básico y su extensión a multiresolución, lo cual mejora la calidad y eficiencia. Posteriormente se mencionará la optimización propuesta por Wei, la cual se basa en cuantización vectorial. Finalmente se mencionarán algunos detalles sobre el algoritmo y los resultados obtenidos.

2.2 Algoritmo básico

Según Wei, su algoritmo está basado, como muchos otros, en el modelo de Campos Aleatorios Markovianos (MRF). Esto es algo que no nos queda muy claro y posteriormente mencionaremos por qué. De cualquier forma, el algoritmo evita la construcción y cálculo de funciones de densidad en favor de una búsqueda vectorial.

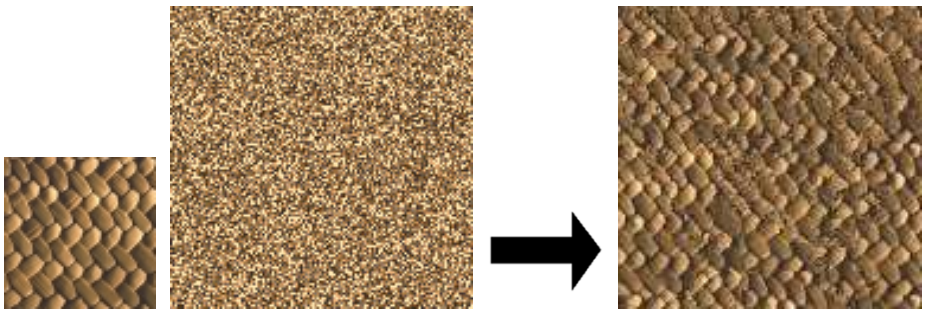

Figura 2.1: Síntesis de Wei-Levoy. Se toman una muestra I_a y una imagen de ruido I_s y se modifica I_s para que se asemeje a I_a.

El algoritmo toma como entrada una imagen I_a que contiene una muestra de alguna textura, y una imagen I_s que contiene ruido aleatorio. Se recorren los pixeles de I_s línea por línea, de arriba hacia abajo, y cada línea de izquierda a derecha ("raster scan order"). Para cada pixel p en I_s se compara su vecindad N_p con todas las posibles vecindades N_r en I_a. El valor del pixel r con la vecindad mas parecida a N_p es asignado al pixel p. Para medir la similaridad de dos vecindades, se utiliza la norma L_2 (suma de los cuadrados de las diferencias). Este proceso se repite hasta haber sintetizado todos los pixeles de I_s.

2.3 Vecindades

La elección del tamaño y forma de las vecindades tiene una importancia considerable para el correcto funcionamiento y desempeño del algoritmo. Inicialmente se consideran vecindades rectangulares de $m \times n$ pixeles. De esta forma, la vecindad de un sitio (x, y) consiste en los sitios $(x + d_x, y + d_y)$ donde $-m/2 \leq d_x \leq m/2$, $-n/2 \leq d_y \leq n/2$, y d_x, d_y no son ambos cero. Sin embargo, al sintetizar I_s siguiendo el orden "raster scan", para un pixel p en I_s, solamente la mitad de los pixeles en N_p tendrán un valor asignado por el algoritmo de síntesis, mientras que la otra mitad no han sido aún sintetizados y tendrán valores aleatorios. Según las pruebas realizadas por Wei, estas vecindades no producen los resultados deseados, por lo que se decidió utilizar vecindades que contengan únicamente pixeles ya sintetizados (excepto para los primeros pixeles en orden "raster scan"). Las vecindades N_p utilizadas por Wei comprenden únicamente los sitios en una vecindad de $n \times m$ que son anteriores a p en el orden del recorrido. Decimos que tales vecindades son *causales* debido a que el valor de un pixel dependerá únicamente de pixeles ya sintetizados (excepto para los primeros pixeles). Llamaremos *no causales* a las vecindades que comprenden los $m \times n - 1$ pixeles en su totalidad. Ambos tipos de vecindades se muestran en la Figura 2.2, donde un pixel p se muestra de color gris y su vecindad N_p son los pixeles de color blanco.

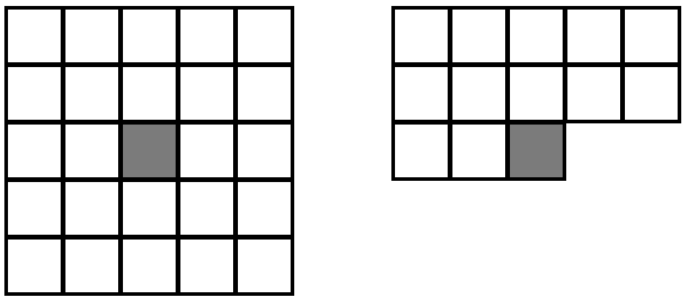

Figura 2.2: Tipos de vecindades: Vecindad no-causal de 5×5 (izquierda) y vecindad causal de 5×5 (derecha).

Si bien el sistema de vecindades propuesto por Wei produce los resultados esperados, cabe preguntarse si es un sistema de vecindades válido, ya que no cumple con la propiedad de que

$$r \in N_s \iff s \in N_r, \qquad \forall r, s$$

Debido a esto, es cuestionable que el algoritmo de Wei esté basado en campos aleatorios de Markov.

Respecto al tamaño de las vecindades, intuitivamente se elige un tamaño similar al tamaño de la estructura mas grande que aparezca en la textura, de otra forma, tales estructuras no podrían ser correctamente sintetizadas y el resultado consistiría en una imagen demasiado aleatoria.

Es posible ver una vecindad de $n \times m$ como un vector de $nm/2$ componentes cuyos valores corresponden a los valores de los pixeles de la vecindad tomados en el orden "raster scan". Así podemos ver que el algoritmo de Wei se reduce a la búsqueda del vector mas cercano (según la norma L_2) a un vector dado N_p, de entre un conjunto de vectores que consiste en todas las vecindades de I_a. Este problema se conoce como *cuantización vectorial* y existen algoritmos muy potentes para optimizar dichas búsquedas.

El tamaño de la vecindad tiene un impacto directo sobre la calidad de la textura resultante y el tiempo de cómputo. En general, vecindades de mayor tamaño producen mejores resultados, aunque esto no siempre sucede. Vecindades muy pequeñas no logran capturar algunas estructuras de la textura fuente, dando como resultado una imagen demasiado aleatoria. Sin embargo, en algunos casos se observa una degradación del resultado al utilizar vecindades muy grandes. Se ha encontrado que las vecindades de 5×5, 7×7 y 9×9 producen los mejores resultados en la mayoría de los casos. En la Figura 2.3 se muestran los resultados para varias texturas con distintos tamaños de vecindad.

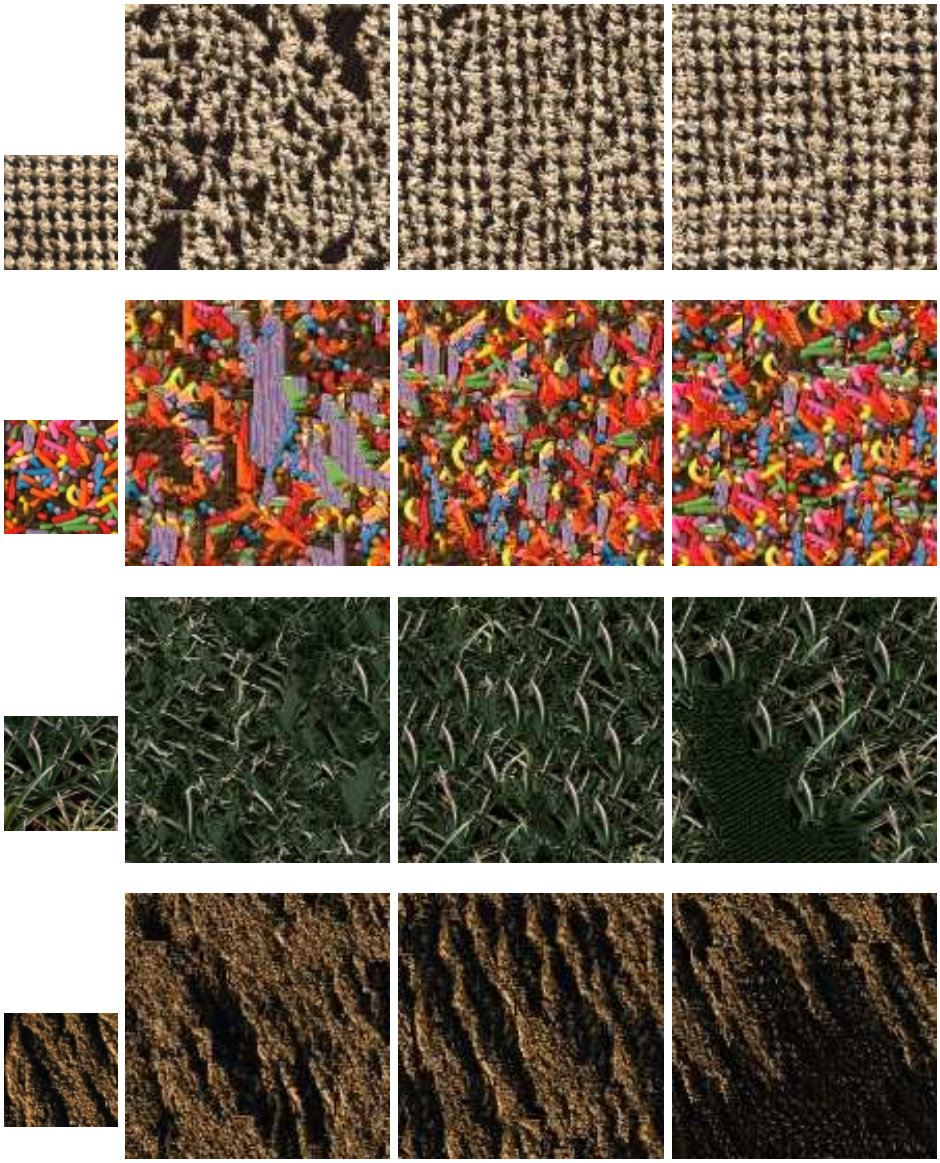

Figura 2.3: Resultados del algoritmo básico: textura fuente en la primera columna, resultados con vecindades de 5×5 en la segunda columna, 9×9 en la tercera columna y 15×15 en la cuarta.

2.4 Algoritmo Multi-Resolución

El algoritmo básico produce buenos resultados con texturas compuestas de estructuras pequeñas. Para texturas que contienen estructuras mas grandes es necesario utilizar vecindades de mayor tamaño, lo cual implica un mayor tiempo de cómputo. Este problema se puede reducir utilizando un algoritmo multiresolución, el cual logra representar estructuras grandes con una cantidad menor de pixeles en imágenes de menor resolución.

En el algoritmo multiresolución se generan dos pirámides gaussianas G_a y G_s, a partir de I_a e I_s. Una pirámide gaussiana se genera filtrando y submuestreando una imagen para obtener el siguiente nivel de menor resolución. El proceso se aplica un determinado número de veces, resultando en una serie de imágenes de mayor a menor resolución donde el nivel de mayor resolución corresponde a la imagen original. El filtrado es necesario para evitar aliasing y artefactos [12] y para realizarlo se utiliza un filtro con kernel gaussiano como el que se muestra a continuación:

$$h[x, y] = \frac{1}{z} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.1}$$

Una vez generadas las pirámides G_a y G_s, se utiliza el algoritmo de resolución simple para transformar G_s partiendo desde los niveles de mas baja resolución hacia los de alta resolución. Cada nivel en G_s se sintetiza utilizando la información existente en los niveles previamente sintetizados. La única diferencia es que para un pixel p en un nivel dado de G_s, su vecindad N_p contiene tanto pixeles en ese nivel como pixeles en niveles de resolución mas baja. De esta manera, los pixeles sintetizados en baja resolución restringen el proceso de síntesis en las resoluciones mas altas, preservando las componentes de baja frecuencia y las estructuras de mayor tamaño en la textura original.

Las vecindades utilizadas en el algoritmo multiresolución se definen como sigue: la vecindad de tamaño $\{m \times n, k\}$ de un pixel p en el nivel l de una pirámide, consiste de los sitios correspondientes a la vecindad causal N_p de resolución sencilla de tamaño $m \times n$ en el nivel l, y de los sitios correspondientes a las vecindades no causales de tamaño $m^{(i)} \times n^{(i)}$ en los niveles $l + i$ para $i = 1, \ldots, k - 1$, donde $m^{(i)}$ y $n^{(i)}$ son aproximadamente la mitad de $m^{(i-1)}$ y de $n^{(i-1)}$, y $m^{(0)} = m, n^{(0)} = n$. Los niveles con índice mayor representan niveles de menor resolución en la pirámide. Una vecindad de este tipo se muestra en la Figura 2.4.

Al utilizar una técnica multiresolución se obtienen resultados de calidad superior al algoritmo de resolución sencilla y con tiempos de cómputo reducidos. Esto se debe a que se pueden utilizar vecindades de menor tamaño las cuales abarcan mayores áreas en los niveles de baja resolución para sintetizar las estructuras mas grandes, y áreas pequeñas en niveles de alta resolución para reproducir los detalles mas finos. En la Figura 2.5 se muestran algunos resultados con distintos tamaños de vecindad.

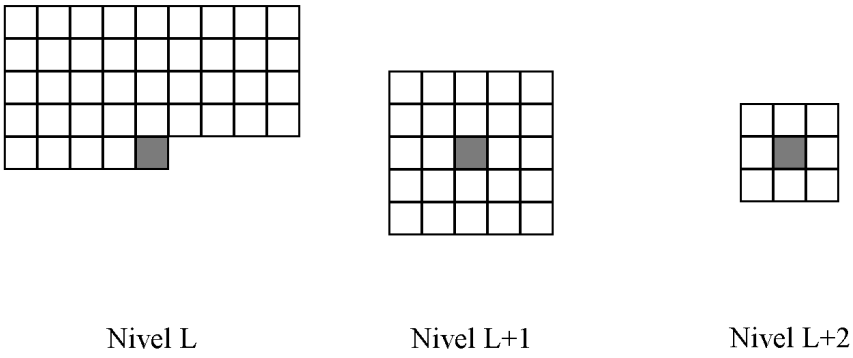

Figura 2.4: Vecindad multiresolución de $\{9 \times 9, 3\}$

2.5 Manejo de bordes

Un aspecto importante en la síntesis de texturas es el manejo de bordes. En particular, el manejo de vecindades que contienen sitios fuera de las imágenes de entrada o de salida.

Para el caso de la imagen o pirámide de entrada, se consideran siempre vecindades toroidales. Esto significa que si $G_s(L, x, y)$ representa el pixel (x, y) en el nivel L de la pirámide G_s, entonces definimos

$$G_s(L, x, y) \equiv G_s(L, x \bmod W_L, y \bmod H_L), \tag{2.2}$$

donde W_L y H_L son las dimensiones horizontal y vertical, respectivamente, del L-ésimo nivel de G_s. El manejo toroidal de bordes en la imagen de salida es imprescindible para lograr que la textura resultante sea *apilable*; es decir, que pueda ser utilizada en forma de mosaico sin perder continuidad en los bordes.

Para la pirámide de entrada G_a, utilizar vecindades toroidales puede resultar en discontinuidades, a menos que la imagen de entrada I_a sea apilable, lo cual rara vez sucede. En este caso, optamos por considerar solamente a aquellos pixeles en la vecindad que se encuentren dentro de G_a y descartar los que queden fuera. Al comparar una vecindad N_p en la pirámide de salida con una vecindad N_r en la pirámide de entrada, solamente se compararán aquellas parejas en donde el sitio correspondiente en N_r esté dento de G_a. Esto se logra calculando una máscara M, donde M es un vector del mismo tamaño que las vecindades tal que

$$M_i = \begin{cases} 0 & \text{si el } i\text{-ésimo sitio en } N_r \text{ queda fuera de } G_a, \\ 1 & \text{en caso contrario.} \end{cases} \tag{2.3}$$

Para medir la distancia entre dos vecindades, ahora es necesario considerar solo aquellos sitios i con $M_i = 1$ y normalizar respecto al número de sitios considerados. Esto puede escribirse como

$$d(N_p, N_r) = \frac{1}{\sum_i M_i} \sum_i (N_{pi} - N_{ri})^2 M_i \tag{2.4}$$

Figura 2.5: Resultados del algoritmo multiresolución: textura fuente en la primera columna, resultados con vecindades de $\{5\times5, 2\}$ en la segunda columna, resultados con vecindades de $\{9 \times 9, 3\}$ en la tercera columna.

2.6 Inicialización

El algoritmo Wei-Levoy es inherentemente determinista; sin embargo, es deseable capturar ciertos aspectos aleatorios que se presentan comúnmente en las texturas, o bien, generar múltiples texturas similares (pero no idénticas) a partir de una misma muestra I_a. Para lograr esto, Wei propone inicializar I_s con ruido, lo cual provee al algoritmo de la suficiente aleatoriedad para lograr nuestros propósitos. En el caso de resolución sencilla, solamente las primeras líneas de pixeles sintetizados harán uso del ruido inicial en I_s. Las líneas subsecuentes serán sintetizadas a partir únicamente de pixeles ya generados debido a la forma causal de las vecindades. Para el caso multiresolución, la imagen completa de ruido inicial juega un papel debido a que todos los pixeles tendrán una contribución en las resoluciones mas bajas de la pirámide.

Debido a la alta dimensionalidad del problema, Wei considera buena idea ecualizar el histograma inicial de G_s respecto al de G_a. De esta forma, las vecindades en G_s serán mas parecidas a aquellas de G_a sin perder el componente estocástico de G_s. Nosotros proponemos una manera muy sencilla de lograr el mismo objetivo. La idea consiste simplemente en inicializar I_s con los valores de pixeles tomados aleatoriamente de I_a. Dicho de otra forma, en vez de inicializar I_s con ruido y posteriormente ecualizar su histograma, simplemente inicializamos I_s con valores aleatorios cuya distribución es precisamente el histograma de I_a.

2.7 Resumen del algoritmo

Nuestra implementación del algoritmo WL toma como entradas una imagen de textura de entrada I_a, una imagen de textura de salida I_s, y el tamaño de vecindad $\{m \times n, k\}$. El proceso de síntesis se realiza como se muestra a continuación.

1. Asignar a cada pixel de I_s el valor de un pixel elegido aleatoriamente de I_a
2. Generar las pirámides gaussianas G_a y G_s a partir de I_a e I_s, respectivamente.
3. Para cada nivel L de G_s desde el nivel de menor resolución hasta el de mayor resolución, hacer lo siguiente:
   (a) Recorrer todos los pixeles p de $G_s(L)$ en orden "raster scan". Para cada uno hacer lo siguiente:
      i. Construír la vecindad N_p de p.
      ii. Encontrar $q = \text{argmin}_r\ d(N_p, N_r)$, donde r recorre todos los sitios de $G_a(L)$.
      iii. Hacer $G_s(L, p) \leftarrow G_a(L, q)$

4. Hacer $I_s \leftarrow G_s(0)$

2.8 Aceleración mediante TSVQ

Si bien el algoritmo WL es eficiente, comparado con otros algoritmos basados en modelos MRF, es posible utilizar técnicas de cuantización vectorial para acelerarlo aún mas. La cuantización vectorial es un problema que puede definirse como sigue: dado un conjunto finito S de puntos en un espacio $\Re^d$, y un punto $p \in \Re^d$, encontrar un punto $q \in S$ tal que la distancia (según una métrica dada) de p a q es menor o igual que la distancia de p a cualquier otro punto de S. Esto es precisamente lo que deseamos encontrar en el paso 3.a.ii del algoritmo presentado en la sección anterior, y también el proceso que mas nos interesa optimizar.

La cuantización vectorial es un problema difícil cuando el espacio de búsqueda es de alta dimensión. Existen algoritmos para realizar dicha búsqueda de manera eficiente. Uno de ellos consiste en preprocesar el conjunto S para crear una estructura de árbol donde los elementos de S se convierten en las hojas. Posteriormente se utiliza un algoritmo de búsqueda que comienza en la raíz del árbol y en cada nodo decide qué camino tomar según la distancia de p al vector correspondiente al nodo actual. Este proceso se conoce como Tree-structured Vector Quantization (TSVQ) [11] [10].

El algoritmo TVSQ sigue una idea similar a la búsqueda en un árbol binario, aunque en general se puede implementar con árboles m-arios. Sin embargo, la construcción del árbol de búsqueda es compleja y no es posible asegurar que se encontrará el vector $q \in S$ que minimiza la distancia a p, si bien el vector encontrado es por lo general cercano al óptimo y la búsqueda es considerablemente mas eficiente.

Nuestra implementación del algoritmo no utiliza TSVQ, por lo que no profundizaremos mas en esta técnica. Sin embargo, cabe mencionar que debido a que TSVQ no siempre encuentra el vector mas cercano, los resultados obtenidos por Wei utilizando TSVQ son de menor calidad que los resultados aplicando búsqueda exhaustiva. En el capítulo siguiente estudiaremos una modificación al algoritmo WL que resulta ser aún mas eficiente que WL con TSVQ.

2.9 Posibles mejoras

El manejo de bordes presentado en la Sección 2.5 es el que propone Wei en su tesis; sin embargo, no obtuvimos buenos resultados aplicándolo tal como se describe. Se probaron distintas modificaciones hasta encontrar una que produjo resultados satisfactorios. La modificación consistió en manejar las vecindades multiresolución en G_a de la forma descrita anteriormente, excepto para aquellos sitios que corresponden al nivel de mayor resolución en la vecindad. Para estos pixeles la máscara M_i correspondiente siempre vale uno. Esto significa que en

una vecindad multiresolución, la parte correspondiente a la mayor resolución se maneja de forma toroidal, y para las resoluciones menores solo se consideran aquellos pixeles dentro de G_a. Todos los resultados obtenidos con el algoritmo multiresolución fueron obtenidos utilizando este manejo de bordes.

Lo anterior se puede justificar de la siguiente forma: supongamos que aplicamos el algoritmo de resolución sencilla. Las vecindades en la imagen de entrada I_a son de tipo causal, pero no toroidales. Esto representa un problema, ya que, por ejemplo, el pixel $I_a(0,0)$ no tiene ningún vecino que caiga dentro de I_a, por lo tanto no es posible comparar la vecindad de $I_a(0,0)$ con la vecindad del pixel en I_s que se desea sintetizar.

Al sintetizar el nivel de mas baja resolución en la pirámide, el proceso es exactamente igual al algoritmo de resolución sencilla, por lo que una vecindad en G_a solamente contendrá pixeles en el mismo nivel de la pirámide (aquel de menor resolución). Ahora bien, si utilizamos el manejo de bordes descrito en la Sección 2.5, solamente consideraremos los sitios que estén dentro de G_a. Ya que la vecindad es de tipo causal, el pixel correspondiente a la esquina superior izquierda no tiene ningún vecino que caiga dentro de G_a. Esto aparentemente produce errores de tipo numérico que se reflejan en las imágenes sintetizadas. En el caso multiresolución en general no sucede esto, ya que la vecindad multiresolución de un pixel p en G_a consiste de la vecindad causal de resolución sencilla de p, así como de vecindades no-causales de resolución sencilla en niveles de menor resolución, las cuales siempre tienen alguna parte dentro de G_a. La excepción se produce al sintetizar el primer nivel (el de menor resolución) de G_s. En este caso, las vecindades multiresolución equivalen a vecindades de resolución simple; sin embargo, los errores generados en este nivel son transmitidos hacia los niveles de mayor resolución, afectando la calidad del resultado.

Se intentó también realizar iteraciones adicionales sobre texturas ya sintetizadas. Esto consistió simplemente en tomar texturas I_s generadas con el algoritmo multiresolución, y aplicar el algoritmo de resolución simple nuevamente a I_s, obviamente sin reemplazar I_s con ruido inicial. Para estas iteraciones adicionales se utilizan vecindades no-causales. En algunos casos esto mejoró ligeramente la calidad del resultado, en particular se mejoró la apilabilidad de las texturas (Figura 2.6), y en algunos otros se observó una degradación de la calidad (Figura 2.7). En la mayoría de los casos no se notaron diferencias significativas en calidad. Concluímos que la única ventaja real de realizar una iteración adicional consiste en disminuír las discontinuidades observadas al utilizar una textura de manera apilable.

2.10 Resultados

El algoritmo Wei-Levoy produce buenos resultados sintetizando texturas compuestas de estructuras relativamente pequeñas. Para texturas formadas por estructuras mas grandes el algoritmo produce buenos resultados en algunos casos, pero en general esto no sucede. Aumentar el tamaño de vecindad en general

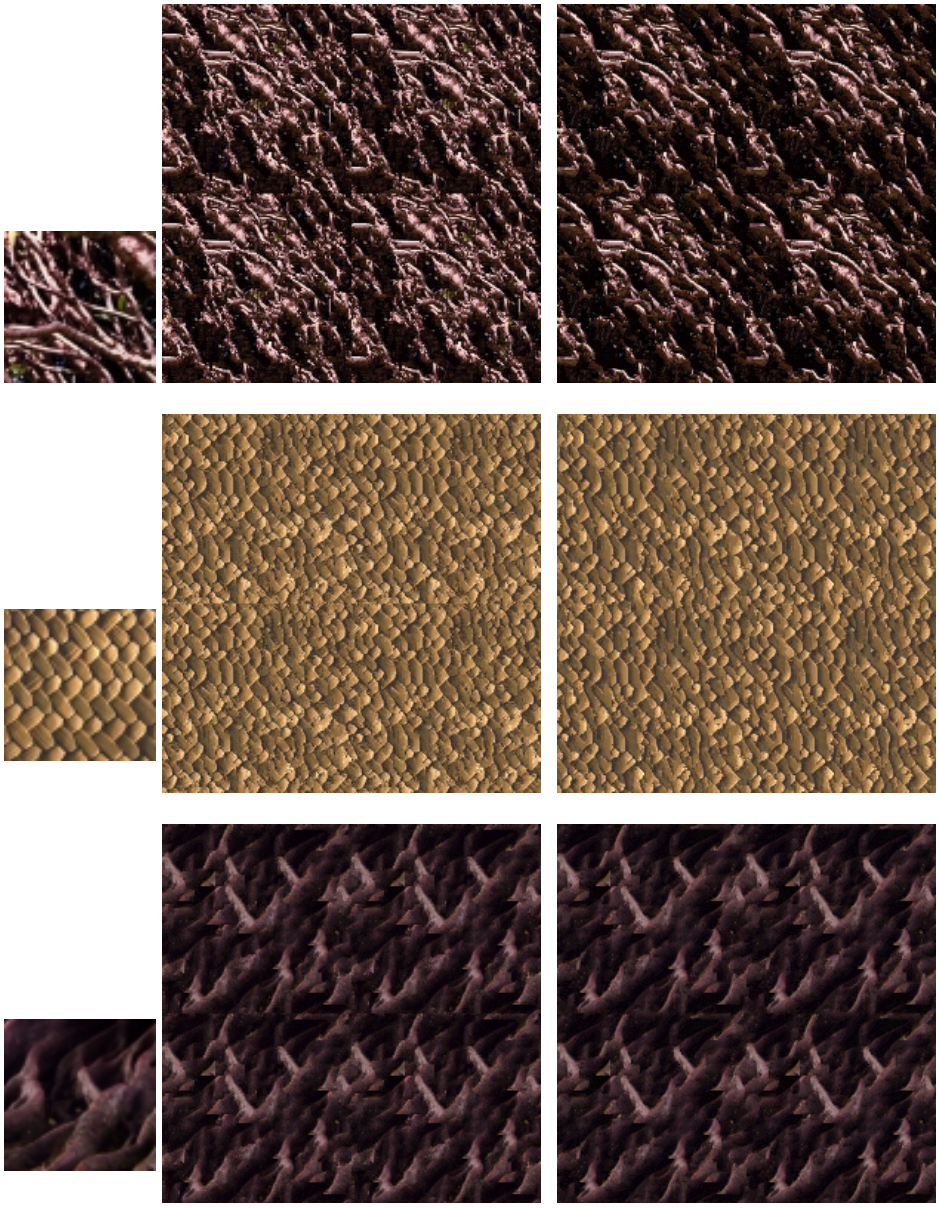

Figura 2.6: Resultados del algoritmo con iteración adicional: textura fuente en la primera columna, resultados (mosaico) sin iteración adicional en la segunda columna, resultados (mosaico) con iteración adicional en la tercera columna. En las texturas generadas sin iteración adicional se puede apreciar discontinuidad a lo largo de la línea horizontal que pasa por la mitad de la imagen. Con iteración adicional estas discontinuidades no son tan visibles.

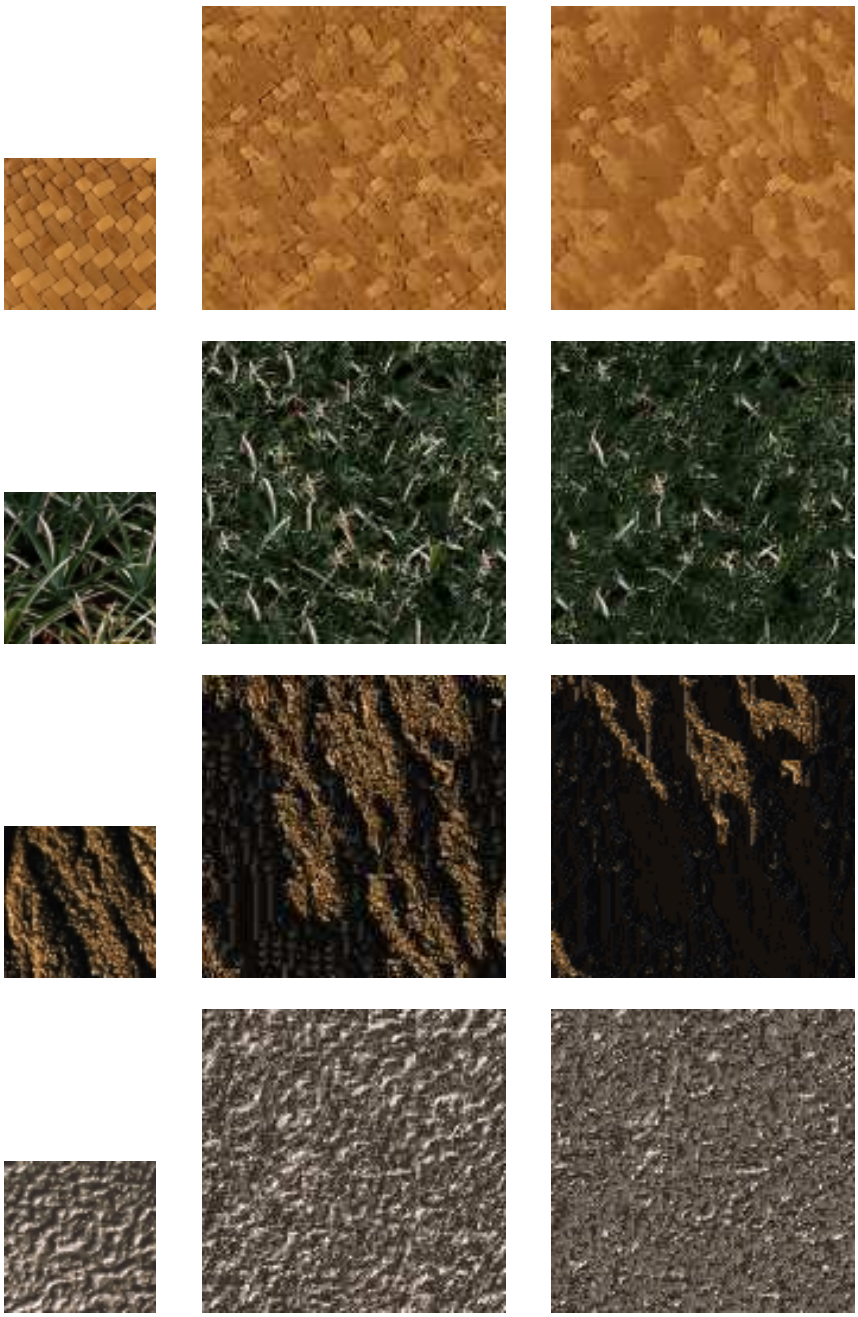

Figura 2.7: Ejemplos donde una iteración adicional degrada la calidad del resultado: textura original (primera columna), resultado sin iteración adicional (segunda columna), resultado con iteración adicional (tercera columna).

mejora la calidad de los resultados, pero también puede ocasionar que el algoritmo intente simplemente replicar grandes zonas de la textura fuente, lo cual tiene el efecto de ocasionar discontinuidades no deseadas. En general, el algoritmo funciona bien en texturas que representan el material de algún objeto, tales como piedra, metal, ladrillo, tejido, etc.

Existe un rango de texturas en el cual el algoritmo WL genera resultados muy pobres. Este consiste en las texturas que no representan un solo tipo de material, sino un arreglo irregular de objetos distintos o en distintas posiciones. Este tipo de texturas son muy comunes en la naturaleza, por ejemplo, pasto, flores, hojas, follaje, pelaje, nubes, montañas, etc. Otro tipo de texturas en las que WL no produce resultados aceptables, son aquellas en que se observa una perspectiva tridimensional. En la Figura 2.8 se muestran algunos ejemplos de estos resultados.

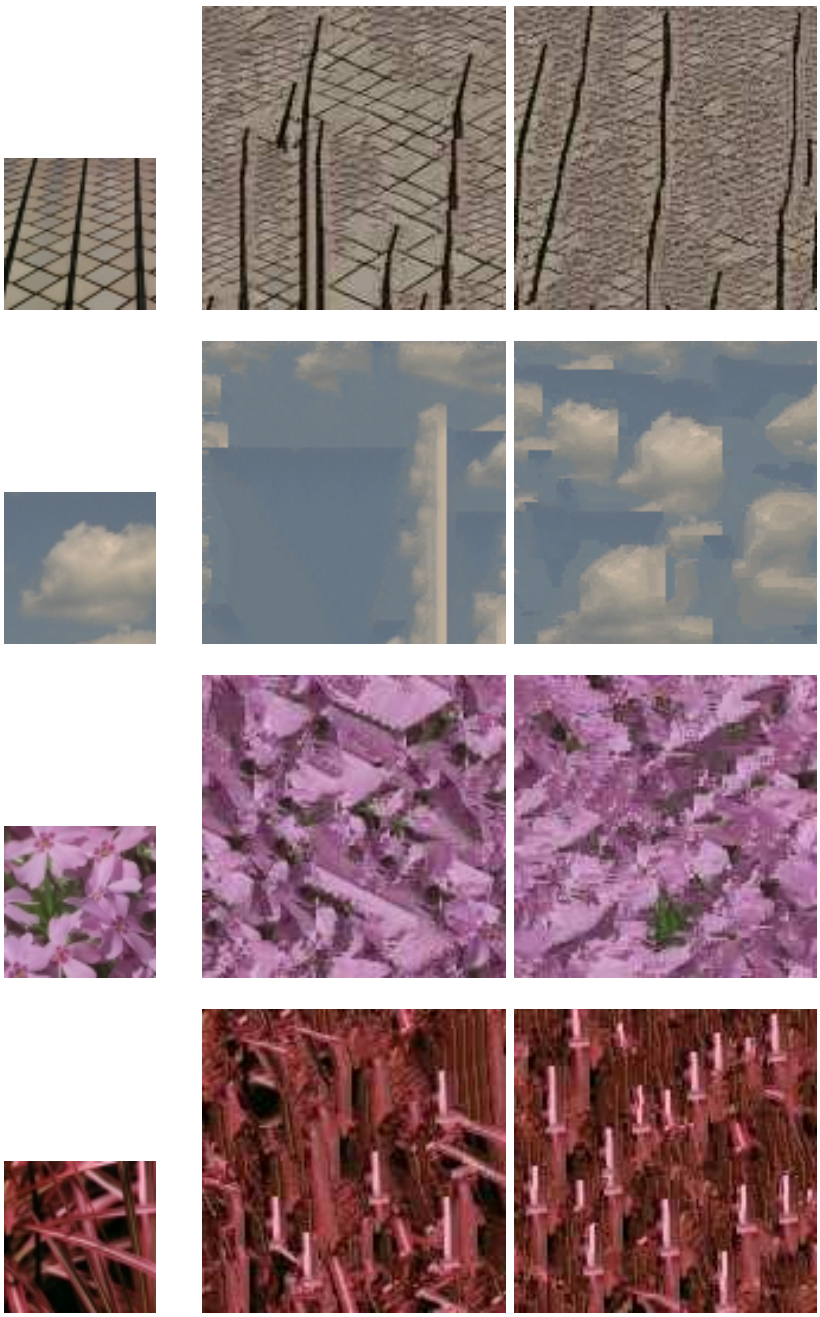

Figura 2.8: Ejemplos de resultados pobres con el algoritmo WL: textura original (primera columna), resultado con resolución simple (segunda columna), resultado con multiresolución (tercera columna).

Capítulo 3

Algoritmo de Ashikhmin

3.1 Introducción

En su artículo, "Synthesizing Natural Textures" [3], Ashikhmin propone una modificación del algoritmo de Wei-Levoy que produce mejores resultados sobre el tipo de texturas en las que WL muestra un desempeño pobre. Este rango de texturas comprende aquellas texturas formadas por arreglos de objetos en distintas posiciones y tamaños.

Una ventaja adicional del algoritmo de Ashikhmin radica en su velocidad, siendo éste significativamente mas rápido que el algoritmo WL. Esto es debido a que al sintetizar cada pixel, el espacio de búsqueda vectorial es considerablemente reducido.

3.2 Descripción del algoritmo

Ashikhmin modifica el algoritmo Wei-Levoy de manera que se fomente el replicado de estructuras grandes, sin requerir necesariamente vecindades similarmente grandes. Una observación importante que hace Ashkhmin consiste en que para cada pixel que se genera, el algoritmo WL realiza una búsqueda exhaustiva entre todas las vecindades de la imagen fuente. Esto no es del todo necesario, ya que es probable que pixeles cercanos en la imagen resultante provendrán de pixeles cercanos en la imagen fuente. El algoritmo WL no es probabilista; sin embargo, se puede aplicar la idea anterior al esquema determinista del algoritmo.

Para explicar el algoritmo, nuevamente utilizaremos la notación introducida en el Capítulo 2. I_a e I_s serán las imágenes y N_p denota la vecindad causal del pixel p. El algoritmo original asigna a cada pixel p en I_s el valor del pixel r_p en I_a cuya vecindad sea la mas parecida a N_p. Llamaremos a r_p el sitio asignado a p durante el proceso de síntesis.

Para sintetizar un pixel p, el algoritmo de Ashikhmin genera primero una lista de *candidatos* de la siguiente forma: para cada $q \in N_p$ se localiza el pixel

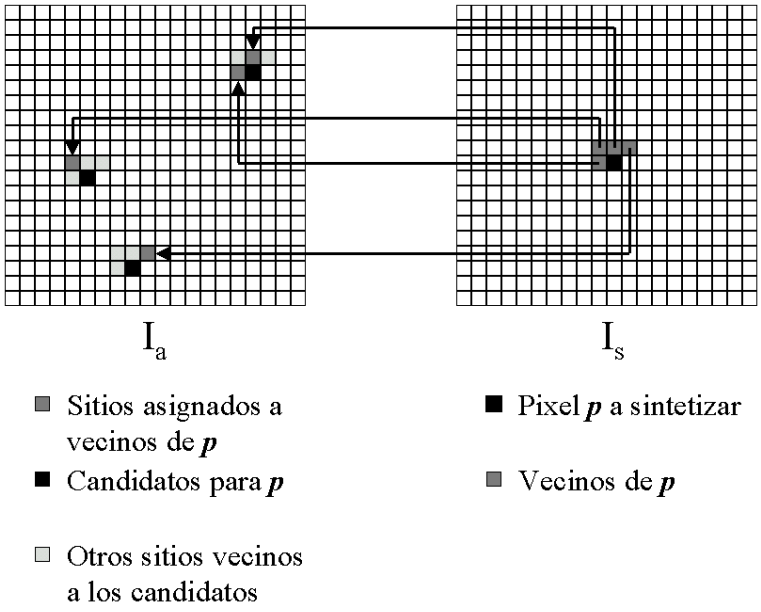

Figura 3.1: Algoritmo de Ashikhmin: En lugar de una búsqueda exhaustiva, solamente se consideran algunos "candidatos" determinados por las asignaciones previas correspondientes a los vecinos del pixel a sintetizar.

r_q en I_a cuyo valor fué previamente asignado a q en el proceso de síntesis. Se calcula el vector de desplazamiento de p a q, $d = q - p$, y se le resta a r_q. El resultado $r_q - d$ es un posible candidato a ser asignado a p. El espacio de búsqueda se reduce entonces solamente a las vecindades de los candidatos, de los cuales se selecciona aquél cuyo vecindario es mas cercano a N_p. De esta forma, el valor asignado a p depende tanto de sus vecinos $q \in N_p$, como de los sitios r_q en I_a asignados a los vecinos. La Figura 3.1 muestra un esquema del algoritmo utilizando una vecindad (causal) de 3×3.

En resumen, para sintetizar un pixel p se realiza lo siguiente:

$$I_s(p) \quad \leftarrow \quad \text{argmin}_r \; d(N_p, N_r), \qquad r \in \{r_q - (q - p) \mid q \in N_p\}. \tag{3.1}$$

Un detalle adicional es que es necesario mantener un registro de las asignaciones r_p hechas a cada pixel p. Debido a que para los primeros pixeles a sintetizar no habrá vecinos a los cuales ya se les haya asignado un sitio en I_a, es necesario inicializar el registro de asignaciones de alguna manera. Al no tener pista alguna sobre cómo se harán las asignaciones, Ashikhmin decide inicializar el registro eligiendo aleatoriamente sitios en I_a.

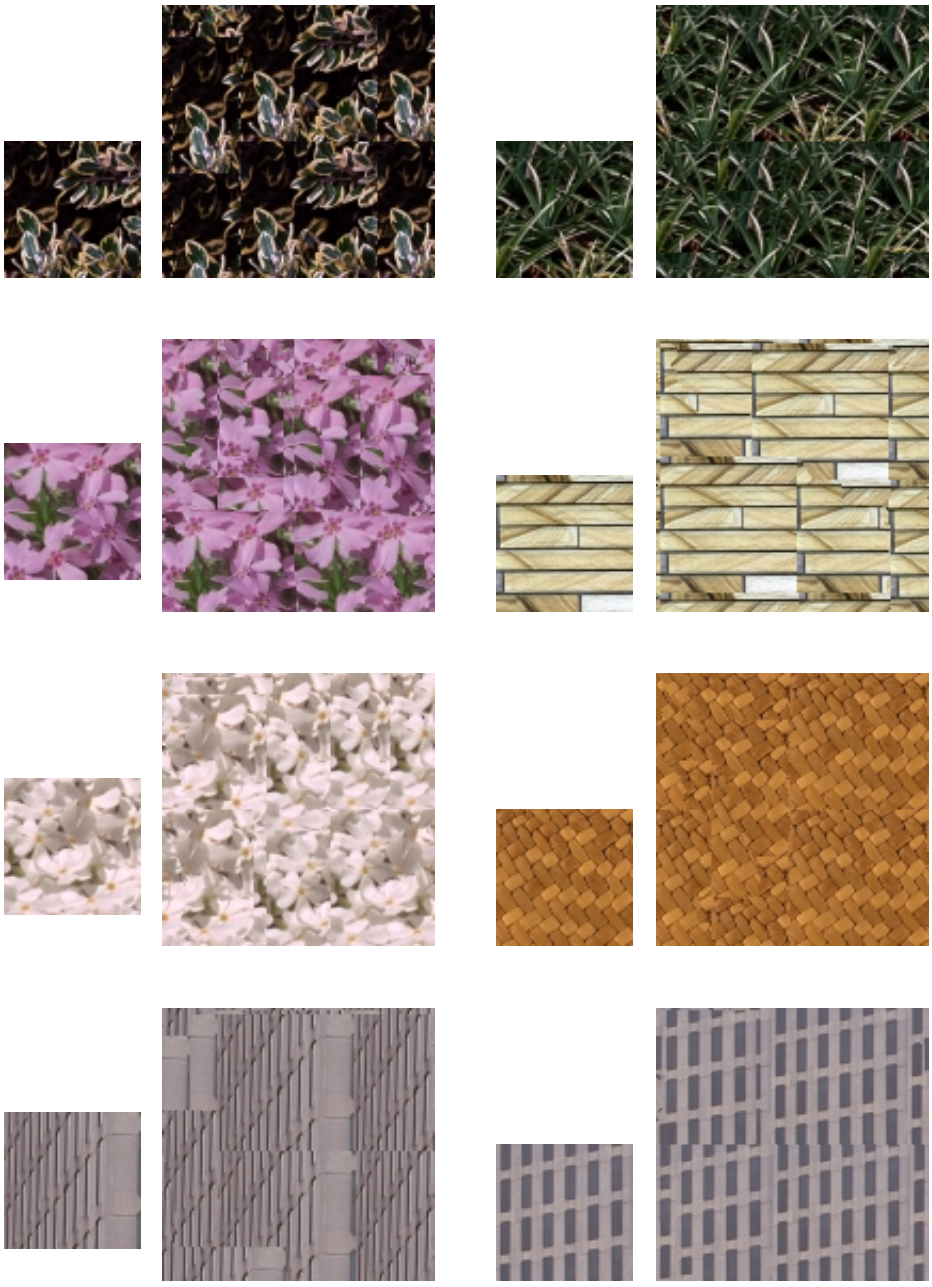

Figura 3.2: Resultados con el algoritmo de Ashikhmin: en cada pareja de imágenes se presenta la textura fuente (imagen pequeña) y la textura sintetizada (imagen grande).

3.3 Multi-resolución

El algoritmo de Ashikhmin logra capturar estructuras grandes en una textura, incluso con un tamaño de vecindad menor al tamaño de dichas estructuras. Esto se debe al altamente restringido proceso de búsqueda, el cual resulta en el copiado de grandes áreas de la textura fuente en la textura destino.

Debido a lo anterior, Ashikhmin asevera que un enfoque multiresolución es de poca utilidad para el algoritmo. Si bien puede ser indispensable para ciertas aplicaciones, Ashikhmin está a favor de la versión de resolución simple la cual es mas eficiente y fácil de implementar.

Para este proyecto solamente se implementó el algoritmo de resolución simple. Con él se obtienen buenos resultados para la clase de texturas para las cuales el algoritmo está orientado.

3.4 Manejo de bordes

El manejo de bordes se realiza de la manera descrita en la Sección 2.5. Para la imagen de salida I_s, las vecindades se manejan de forma toroidal (Ecuación 2.2). Al comparar una vecindad en I_s con las vecindades N_r en la imagen fuente I_a, solamente se consideran aquellos sitios de N_r que caen dentro de I_a. Para ello se utiliza la máscara y la distancia mostradas en las Ecuaciónes 2.3 y 2.4, respectivamente.

El problema descrito en el primer párrafo de la Sección 2.9 no se presenta en el algoritmo de Ashikhmin debido a que la búsqueda para un pixel p se restringe solamente a las vecindades correspondientes a los candidatos, los cuales son aleatorios para los primeros pixeles. Esto disminuye la probabilidad de comparar vecindades que caen en su mayor parte fuera de I_a, y también los errores numéricos asociados a tales vecindades.

También es posible aplicar una segunda iteración a I_s utilizando vecindades no-causales. Esto mejora la apilabilidad de las texturas resultantes, y en algunos casos, también mejora la calidad.

3.5 Comparación con WL

En general, el algoritmo de Ashikhmin produce resultados similares a los obtenidos con WL. El algoritmo está dirigido a texturas formadas por arreglos de objetos de distintas formas o en distintas orientaciones. Para esta clase de texturas, el algoritmo de Ashkhmin en general funciona mejor que Wei-Levoy, logrando captar las estructuras principales sin deformarlas de la manera que lo hace WL. Algunos ejemplos se muestran en la Figura 3.4. En otros tipos de texturas, el algoritmo de Ashkhmin produce resultados mas pobres (Figura 3.3), o incluso no satisfactorios. Este algoritmo es mas propenso a producir

Figura 3.3: Wei-Levoy versus Ashikhmin: textura original (primera columna), resultado con WL multiresolución (segunda columna), resultado con Ashikhmin (tercera columna). Obsérvense las discontinuidades producidas por el algoritmo de Ashikhmin.

resultados con discontinuidades claramente visibles; esto se debe probablemente a que el espacio de búsqueda es tan reducido que no siempre puede encontrarse un candidato aceptable.

Existen texturas en las cuales ambos algoritmos producen resultados pobres. En el caso de las texturas que presentan una perspectiva tridimensional, el algoritmo de Ashkhmin logra captar mejor la perspectiva que WL, pero no de manera aceptable. Algunos de estos resultados se presentan en la Figura 3.5.

En términos de velocidad, el algoritmo de Ashikhmin es sin duda mucho más eficiente que el algoritmo WL. Por ejemplo, si la textura fuente I_a tiene un tamaño de 64×64, la búsqueda exhaustiva compara, para cada pixel en I_s, las 4096 posibles vecindades en I_a, mientras que el algoritmo de Ashikhmin con vecindades causales de 5×5 solamente analiza un máximo de 12 vecindades por pixel.

3.6 Resultados

El algoritmo de Ashikhmin logra captar estructuras que se le escapan al algoritmo Wei-Levoy; sin embargo, la búsqueda vectorial es tan restringida que en

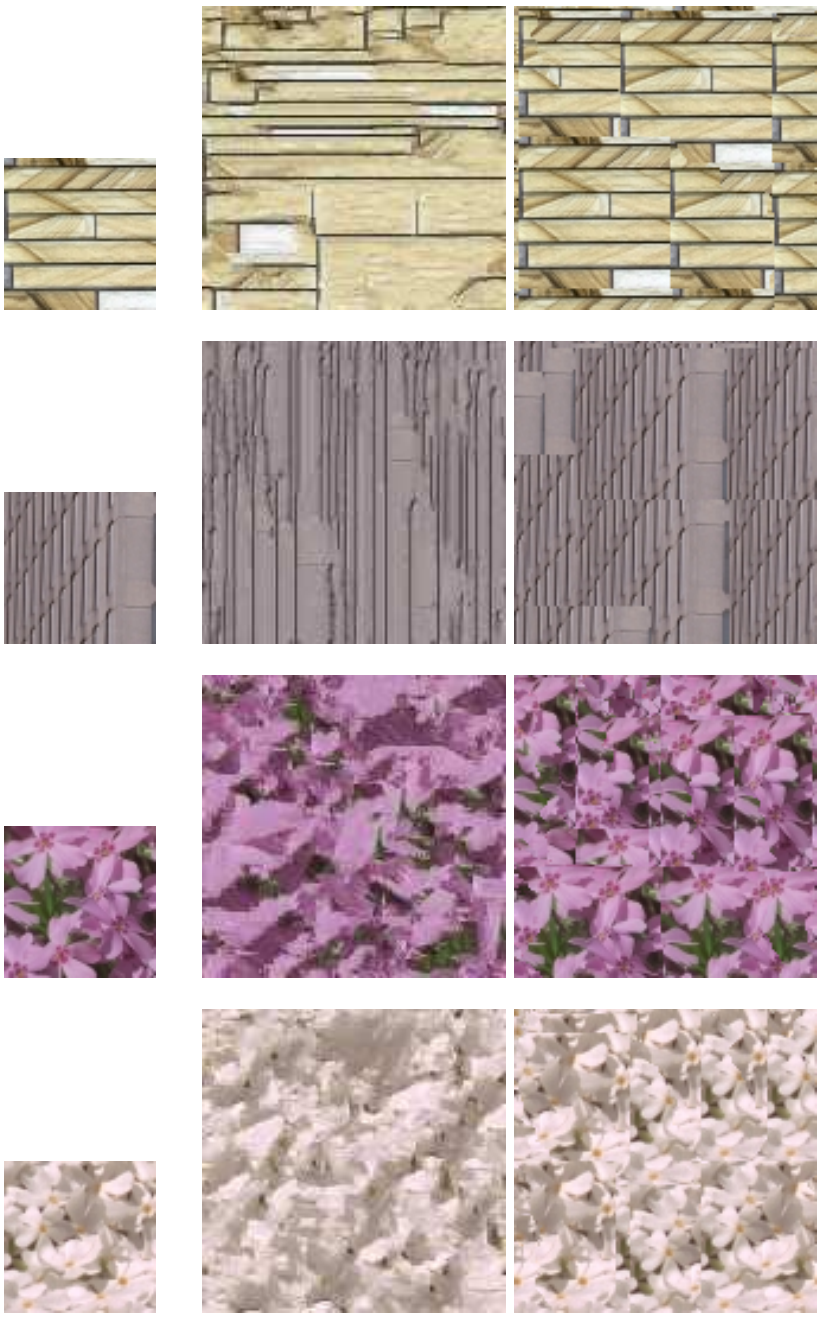

Figura 3.4: Wei-Levoy versus Ashikhmin: textura original (primera columna), resultado con WL multiresolución (segunda columna), resultado con Ashikhmin (tercera columna).

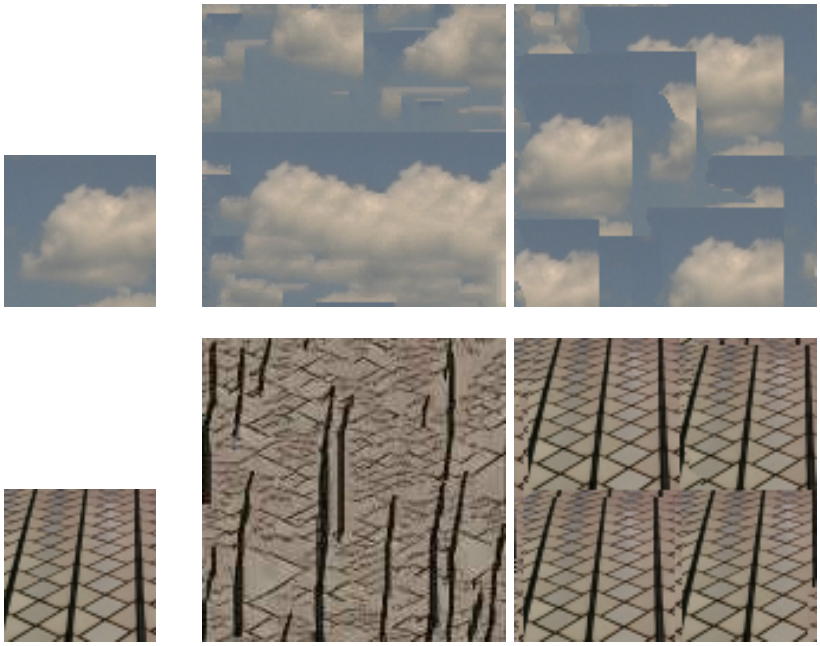

Figura 3.5: Resultados pobres con ambos algoritmos: textura original (primera columna), resultado con WL multiresolución (segunda columna), resultado con Ashikhmin (tercera columna).

muchos casos no se obtiene suficiente aleatoriedad y continuidad en los resultados.

Capítulo 4

Algoritmo de Paget

4.1 Introducción

Paget ha publicado una serie de artículos [5] [6] [4] donde desarrolla un modelo probabilista basado en Campos Aleatorios de Markov para sintetizar un amplio rango de texturas. Este modelo puede ser utilizado también con un enfoque multiresolución para mejorar la calidad de los resultados.

Suponemos que una textura puede ser modelada por un Campo Aleatorio Markoviano, bajo un sistema de vecindades $N = \{N_s | s \in S\}$. Si podemos estimar las probabilidades marginales $P(x_s \mid x_r, r \in N_s)$, entonces es posible utilizar algún método de relajación estocástica, tal como el muestreador de Gibbs, para construír una secuencia de imágenes que converja a la textura en cuestión.

Podemos estimar las probabilidades marginales de una manera no paramétrica mediante un histograma multidimensional de la textura fuente. Este histograma debe ser suavizado, y para ello se utiliza un estimador de densidad conocido como *ventana de Parzen*.

4.2 Modelo de Paget basado en MRF

Dada una textura fuente I_a, nuestro objetivo es transformar una imagen I_s de manera que se asemeje a I_a. Sea S_s al conjunto de sitios en la rejilla definida por I_s. Si suponemos que I_s puede modelarse mediante un MRF, entonces se cumple la siguiente propiedad:

$$P(I_s(r)|I_s(s), s \neq r) = P(I_s(r)|I_s(s), s \in N_r), \tag{4.1}$$

donde $N_r \in N$ es la vecindad del sitio r. El sistema de vecindades utilizado por Paget, es el propuesto por Geman y Geman [7] (Figura 4.1), donde se define el sistema de vecindades N^o, de orden o, de la manera siguiente:

$$N^o_{i,j} = \{(x, y) \in S \mid 0 < (x - i)^2 + (y - j)^2 \leq o\} \tag{4.2}$$

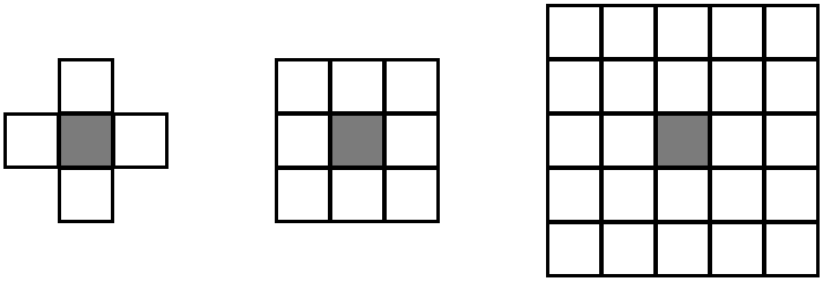

Figura 4.1: Vecindades de Geman y Geman: vecindad de primer orden $o = 1$ (izquierda), vecindad de segundo orden $o = 2$ (centro), vecindad de octavo orden $o = 8$ (derecha).

A diferencia de las vecindades utilizadas por Wei-Levoy y Ashikhmin, el sistema de vecindades de Geman y Geman es simétrico en el sentido de que $r \in N_s$ si y solo si $s \in N_r$. Dada una imagen I, podemos encontrar para cualquier vecindad $N_r, r \in S_I$ un vector $I(N_r)$ donde los elementos del vector representan la intensidad de los pixeles que forman la vecindad en orden raster scan. Definimos también una *ocurrencia* como un vector de dimensión $|N_r| + 1$, donde el primer elemento representa la intensidad de un pixel y los siguientes N_r elementos equivalen a $I(N_r)$. Una ocurrencia es entonces una variable aleatoria vectorial (X, N_X) que representa la intensidad X de un pixel y las intensidades N_X de sus vecinos, en un orden particular.

El histograma multidimensional de ocurrencias en I_a está dado por

$$F(X, N_X) = \sum_{r \in S_a} \delta(I_a(r) - X)\delta(I_a(N_r) - N_X) \tag{4.3}$$

Una burda estimación de las probabilidades marginales para I_s se obtiene al normalizar el histograma $F(X, N_X)$ respecto a X:

$$\hat{P}(I_s(r)|I_s(s), s \in N_r) = \frac{F(I_s(r), I_s(N_r))}{\sum_x F(x, I_s(N_r))}, \tag{4.4}$$

donde la sumatoria sobre x recorre el espacio de estados (intensidades) de I_s.

Desafortunadamente, dada la alta dimensionalidad del problema, los datos del histograma están dispersos de manera rala, por lo que en muchos puntos la frecuencia de aparición de la ocurrencia correspondiente es cero, lo cual ocasiona que la Ecuación 4.4 no esté definida en muchos puntos. Esto hace necesaria la utilización de algún método para "espacir" las probabilidades de las ocurrencias de la muestra tomada de I_a. Una técnica comúnmente utilizada es la ventana de Parzen, la cual calcula un histograma $\hat{F}(z)$ suavizado a partir de un conjunto

de muestras $Z_1, \ldots, Z_n$. Este histograma suavizado tiene la siguiente forma

$$\hat{F}(z) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{1}{h}(z - Z_i)\right), \tag{4.5}$$

donde h es un parámetro de la ventana y d es la dimensión del vector z y de las muestras Z_i.

La forma del histograma suavizado está determinada por el kernel K. Para este caso, se elige utilizar un kernel gaussiano multidimensional dado por

$$K(z) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}z^T z}. \tag{4.6}$$

El parámetro h determina la anchura del kernel. Un valor adecuado de h da lugar a una mejor estimación del histograma $\hat{F}$. Silverman [8] propone el siguiente valor óptimo para el parámetro de ventana:

$$h_{\text{opt}} = \sigma \left(\frac{4}{n(2d+1)}\right)^{1/(d+4)}, \tag{4.7}$$

donde σ^2 es la varianza asociada al histograma unidimensional.

El algoritmo de Paget calcula para cada pixel r de I_s, una estimación de la distribución marginal $\hat{P}(r|s, s \in N_r)$ dada por la Ecuación 4.4, donde F se sustituye por la función del histograma suavizado (Ecuación 4.5). Esto implica evaluar la función kernel para cada vecindad en I_a y para cada valor dentro del rango dinámico de I_a. Más aún, la función kernel calcula el exponencial de la magnitud de su parámetro. Si utilizamos directamente las fórmulas anteriores para sintetizar una textura dada una muestra de 64×64 pixeles y 256 tonos de gris, se requeriría calcular 10^{20} productos punto (de alta dimensión) y 10^{20} exponenciales por cada pixel en I_s. Es posible sustituír el cálculo de la exponencial por un producto de valores tomados de una tabla precalculada de la siguiente manera:

$$\begin{aligned} e^{-\frac{1}{2}z^T z} &= e^{-\frac{1}{2}(z_1^2 + z_2^2 + \ldots + z_d^2)} && (4.8)\\ &= e^{-z_1^2/2} e^{-z_2^2/2} \ldots e^{-z_d^2/2} && (4.9)\\ &= \prod_{i=1}^{d} e^{-z_i^2/2} && (4.10) \end{aligned}$$

Sustituyendo $z \to (z - Z_p)/h$, tenemos que

$$\begin{aligned} K\left(\frac{1}{h}(z - Z_p)\right) &= \frac{1}{(2\pi)^{d/2}} \prod_{i=1}^{d} e^{-(z_i - Z_{pi})^2/2h^2} && (4.11)\\ &= \frac{1}{(2\pi)^{d/2}} \prod_{i=1}^{d} T[z_i, Z_{pi}], && (4.12) \end{aligned}$$

donde T es una tabla definida como

$$T[a,b] = e^{-(a-b)^2/2h^2} \tag{4.13}$$

Aún con la optimización descrita, el algoritmo de Paget es significativamente mas lento que el algoritmo de Wei-Levoy. Paget utilizó una implementación paralela de su algoritmo para realizar las pruebas, y ejecutó el programa en una máquina con 16384 procesadores, uno dedicado a cada pixel de la textura resultante. En nuestro caso, utilizando una máquina con procesador AMD Athlon a 1.8 GHz, las pruebas requerían desde varios minutos para sintetizar texturas pequeñas (32×32 pixeles) hasta varias horas para texturas mas grandes. Debido a esto no fué posible completar una versión depurada del algoritmo, y mucho menos realizar el número de pruebas que se hicieron con los algoritmos de Wei-Levoy y de Ashikhmin. Las pruebas del algoritmo de Paget que mostramos fueron tomadas directamente de la página web de Paget [13].

4.3 Multiresolución

Paget también adopta un enfoque multiresolución para mejorar la calidad del algoritmo sin necesidad de aumentar el tamaño de vecindad. Sin embargo, a diferencia de Wei, Paget no redefine su sistema de vecindades para definir una vecindad multiresolución, sino que una vez sintetizado un nivel, reconstruye el siguiente nivel (de mayor resolución) mediante un proceso de supermuestreo (sin interpolación), y aplica el proceso de síntesis al nivel reconstruído, pero modificando solamente los pixeles introducidos por el supermuestreo. La pirámides multiresolución son obtenidas mediante simple decimación de las imágenes originales, sin realizar ningún tipo de prefiltrado.

Desafortunadamente, no fué posible verificar con exactitud el proceso que realiza Paget en su enfoque multiresolución, el cual parece ser más sencillo que el utilizado por Wei ya que evita la compleja construcción de vecindades multiresolución. Por otra parte, Paget no menciona en sus artículos la manera en que maneja los bordes, por lo que no estamos seguros si el algoritmo de Paget asegura la apilabilidad de las texturas generadas.

Paget incorpora un par de elementos mas a su algoritmo. El primero es una función de temperatura de pixel, que realiza un proceso de recocido local, disminuyendo a lo largo del tiempo la probabilidad de que un pixel cambie su valor de manera significativa. El segundo elemento es una implementación paralelizada del algoritmo, el cual aumenta considerablemente la eficiencia, suponiendo que se tiene el hardware adecuado para ejecutar dicha implementación. A primera vista, los algoritmos de Wei-Levoy y Ashikhmin no pueden ser fácilmente paralelizados debido a que para sintetizar un pixel es necesario haber sintetizado previamente a todos sus vecinos.

Figura 4.2: Resultados con el algoritmo de Paget: en cada pareja de imágenes se presenta la textura fuente (imagen pequeña) y la textura sintetizada (imagen grande).

4.4 Resultados

En muchos casos, el algoritmo de Paget produce resultados buenos, o al menos aceptables. Se observan buenos resultados incluso en texturas con las cuales Wei-Levoy y Ashikhmin produjeron resultados pobres. También se observa que en algunos casos, Paget obtiene resultados de menor calidad que los otros algoritmos estudiados. La Figura 4.2 muestra algunos de los resultados exitosos obtenidos con el algoritmo de Paget. Las Figuras 4.3 y 4.4 muestra resultados obtenidos con los tres algoritmos en los cuales se observan diferencias notables en la calidad de los resultados. Para estos resultados se utilizaron vecindades multiresolución de $\{5 \times 5, 2\}$ para el algoritmo WL, vecindades de 5×5 para el de Ashikhmin, y vecindades de orden $o = 4$ para el algoritmo de Paget. Tal vez fuimos un poco injustos al no utilizar vecindades mayores para el algoritmo WL, pero decidimos que el tamaño de vecindad fuera lo mas parecido para los tres algoritmos. Finalmente, la Figura 4.5 muestra algunas texturas que ninguno de los algoritmos pudo sintetizar satisfactoriamente.

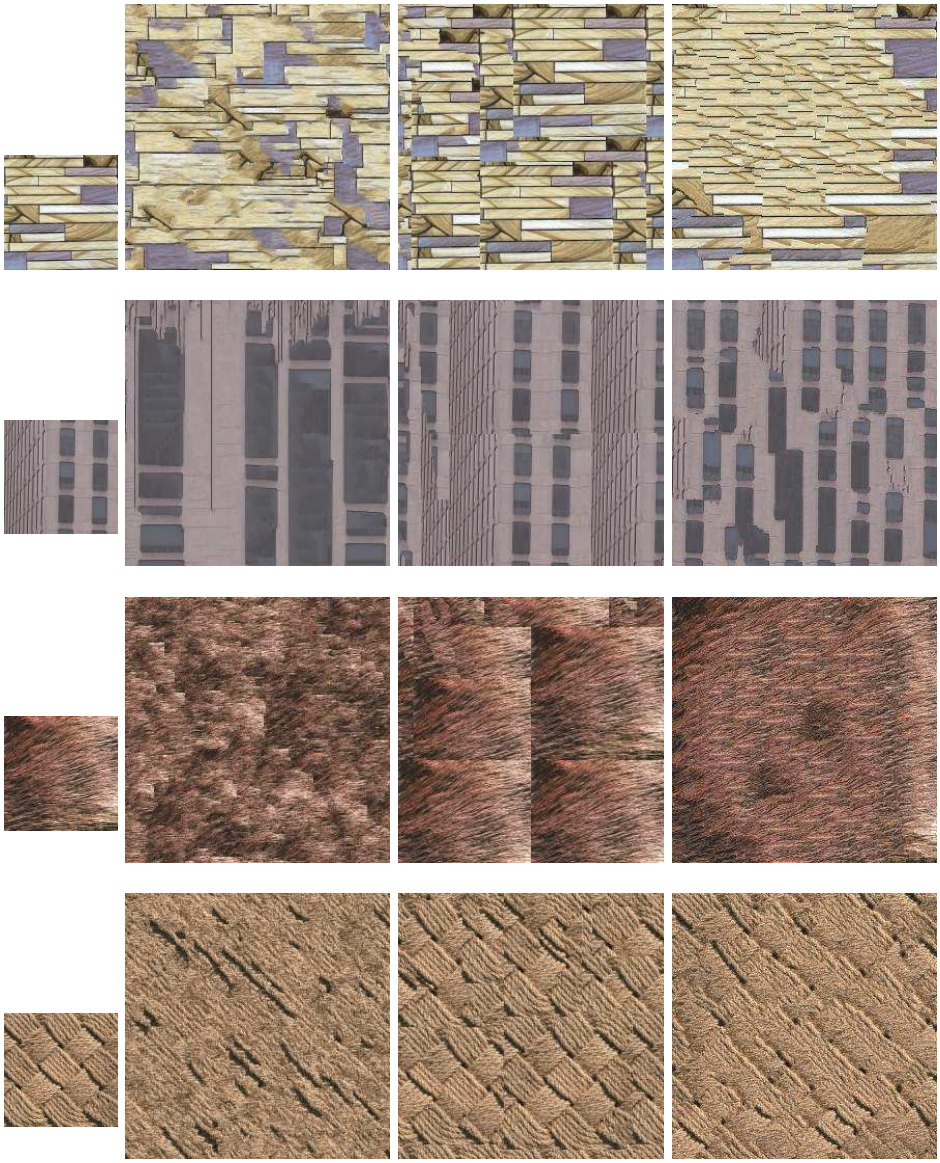

Figura 4.3: Comparación de algoritmos (parte 1): textura fuente en la primera columna, resultados con Wei-Levoy en la segunda columna, resultados con Ashikhmin en la tercera columna y resultados con Paget en la cuarta.

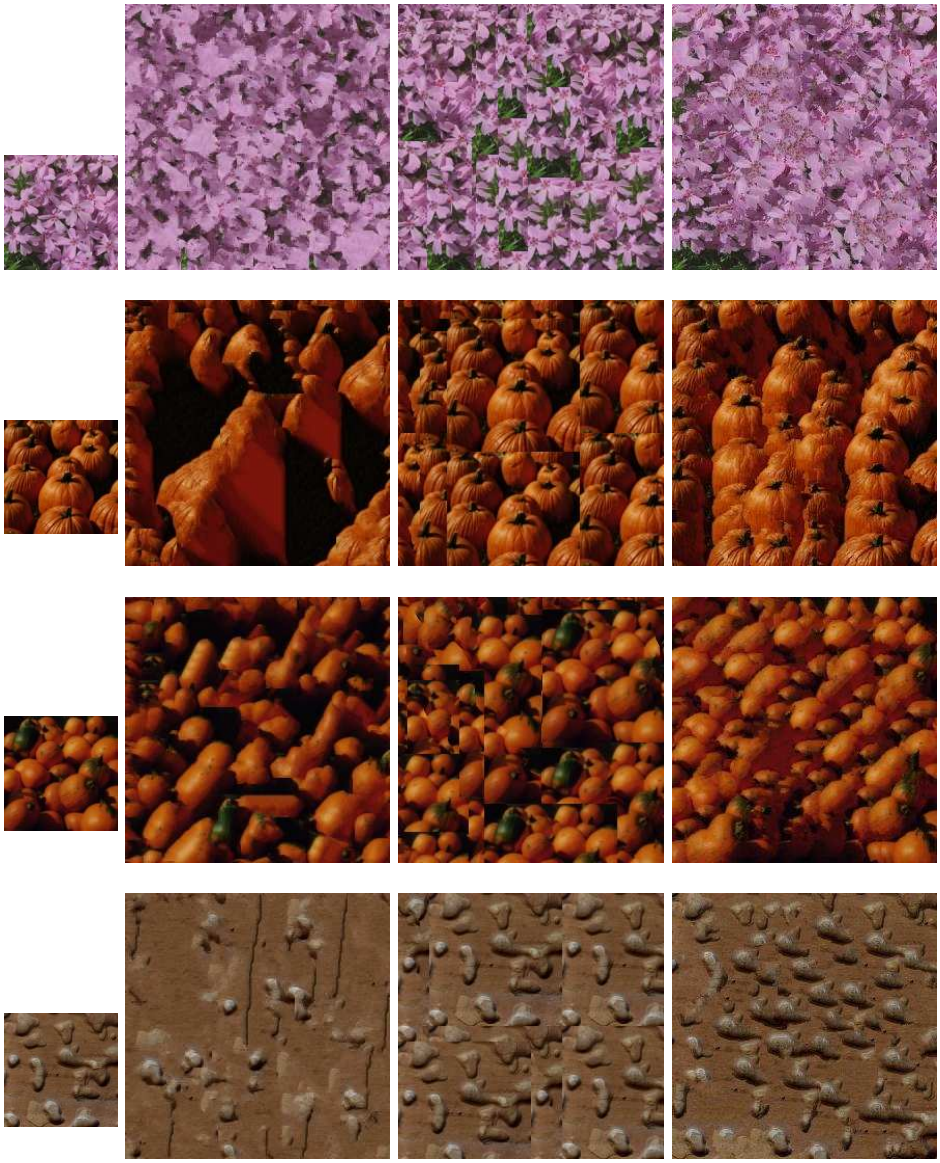

Figura 4.4: Comparación de algoritmos (parte 2): textura fuente en la primera columna, resultados con Wei-Levoy en la segunda columna, resultados con Ashikhmin en la tercera columna y resultados con Paget en la cuarta.

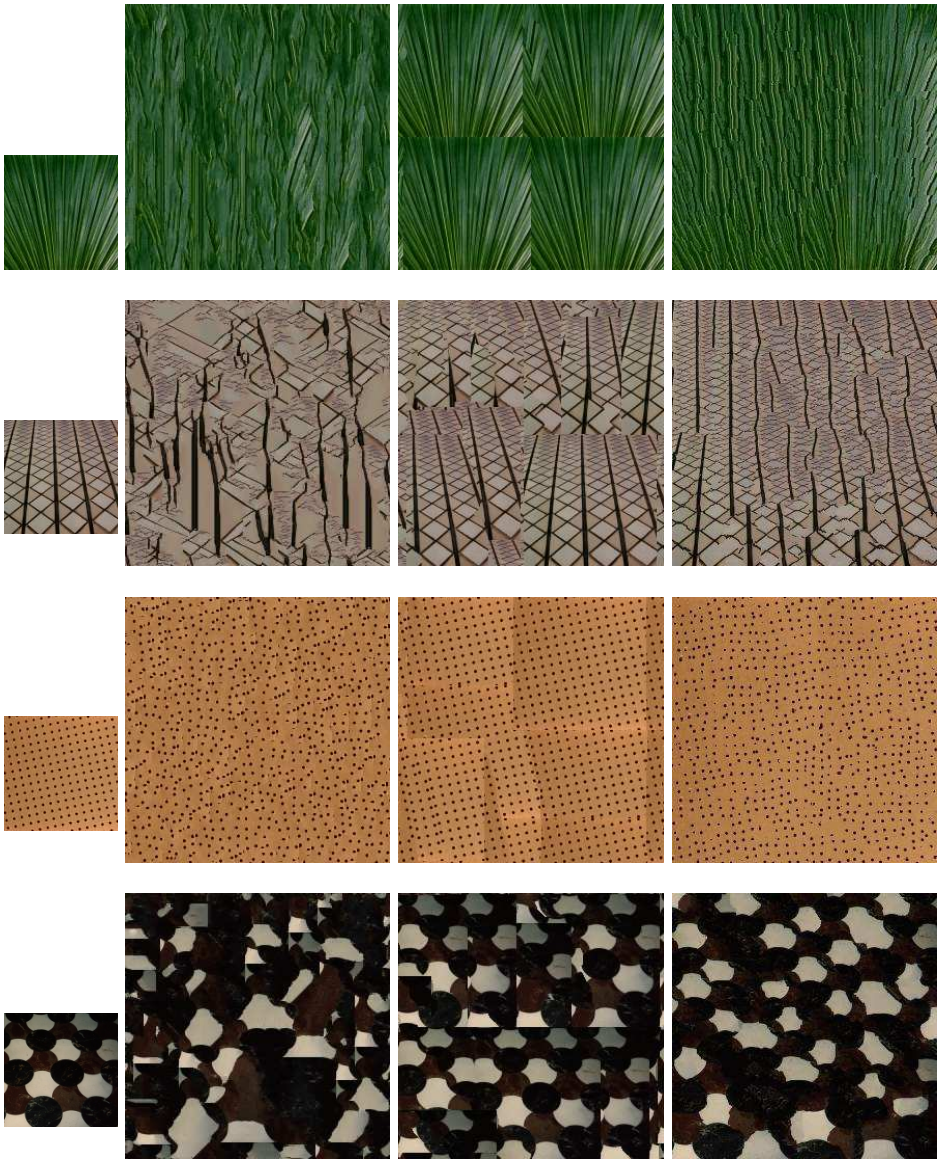

Figura 4.5: Resultados pobres: textura fuente en la primera columna, resultados con Wei-Levoy en la segunda columna, resultados con Ashikhmin en la tercera columna y resultados con Paget en la cuarta.

Capítulo 5

Conclusiones

5.1 Resumen de los algoritmos

Se estudiaron tres algoritmos de síntesis de textura: algoritmo de Wei-Levoy, algoritmo de Ashikhmin y algoritmo de Paget. Los dos primeros fueron implementados satisfactoriamente, cosa que no se logró con el algoritmo de Paget.

El algoritmo de Wei-Levoy toma como entradas una muestra I_a de alguna textura y una imagen I_s con ruido. Para cada pixel p en I_s se compara su vecindad N_p con todas las vecindades existentes en I_a y se asigna a p el valor de aquel pixel en I_a cuya vecindad sea la mas parecida a N_p. Las vecindades que se utilizan son de tipo causal, de manera que al sintetizar la mayoría de los pixeles en I_s, las vecindades de éstos contengan pixeles ya sintetizados.

Se puede aplicar un enfoque multiresolución al algoritmo WL, definiendo vecindades multiresolución que contengan pixeles tanto en el nivel que se está sintetizando, como en los niveles de menor resolución ya sintetizados. Esto aumenta la eficiencia y la calidad de los resultados. También es posible optimizar el algoritmo utilizando técnicas de cuantización vectorial.

El algoritmo de Ashikhmin es una modificación del algoritmo WL que reduce la cuantización vectorial realizada por Wei, a una búsqueda en un conjunto muy reducido de *candidatos* para cada pixel. Esto añade restricciones adicionales al proceso de síntesis, con lo cual se logran capturar estructuras grandes utilizando vecindades relativamente pequeñas. La búsqueda restringida utilizada por Ashikhmin aumenta considerablemente la velocidad del algoritmo, siendo éste el más rápido de los algoritmos estudiados.

El algoritmo de Paget se basa en modelar las texturas con un Campo Aleatorio Markoviano. Se calcula un histograma multidimensional suavizado mediante la ventana de Parzen, y se estiman las probabilidades marginales a partir de dicho histograma. Con lo anterior, es posible utilizar un muestreador de Gibbs o un algoritmo estilo ICM para generar una secuencia de imágenes que converjan

a una textura con la distribución deseada. También es posible extender el algoritmo a multiresolución con un enfoque ligeramente distinto al utilizado por Wei-Levoy. El proceso seguido por Paget es en general muy lento pero en la mayoría de los casos produce resultados de muy buena calidad.

5.2 Comparación de los algoritmos

Presentamos ahora una comparación de los tres algoritmos en tres áreas distintas: calidad de los resultados, velocidad del algoritmo, y rango de aplicaciones. Esta comparación se basa únicamente en nuestras observaciones y no pretende discriminar la utilización de cualquiera de los algoritmos en una aplicación determinada.

5.2.1 Calidad de los resultados

Esta es, posiblemente, la comparación mas difícil a la que nos enfrentamos. Para cada uno de los algoritmos, se han obtenido resultados, en ciertos casos, mejores que los producidos por los otros algoritmos. Por otra parte, existen texturas que ninguno de los algoritmos presentados ha podido sintetizar satisfactoriamente. Sin embargo, consideramos esta área la mas importante dado que el objetivo de un algoritmo de síntesis es generar una imagen que se asemeje (visualmente) a la textura fuente.

El algoritmo Wei-Levoy funciona bien en la mayoría de los casos en que las texturas representan el material de algún objeto, por ejemplo, tejido, arena, metal, roca, hilado, piel, etc. En algunos casos, puede reproducir texturas formadas por varios objetos en distintas posiciones y tamaños, por ejemplo, flores, pasto, hojas, granos, etc. Para la mayoría de estas texturas, el método genera estructuras deformes o borrosas. En el caso de las texturas que presentan una estructura muy rígida, o una perspectiva tridimensional, WL es incapaz de capturar satisfactoriamente estas características.

El algoritmo de Ashikhmin produce en general resultados muy parecidos a los de Wei-Levoy, pero en particular destaca en aquellas texturas formadas por diversos objetos en distintas posiciones (flores, hojas, etc.). Esto se debe a que el algoritmo tiende a duplicar grandes trozos de la textura fuente, intentando mantener cierta continuidad. Esto no es siempre posible; de hecho si uno observa con cuidado, la mayoría de los resultados obtenidos con el algoritmo de Ashikhmin presentan múltiples discontinuidades visibles. Estas discontinuidades también pueden presentarse con el algoritmo Wei-Levoy, en particular si se optimiza mediante TSVQ, pero no son tan perceptibles.

Finalmente, el algoritmo de Paget utiliza un enfoque formalmente basado en MRF. Una diferencia muy importante respecto a los otros dos algoritmos, es que Paget construye una distribución de probabilidades a partir del histograma

multidimensional suavizado de la textura fuente. Este suavizado abre la posibilidad de que al sintetizar un pixel, se le asigne a éste un valor que los otros algoritmos ni siquiera considerarían. Esto elimina en gran medida las discontinuidades perceptibles en la textura resultante. Consideramos que el hecho de no introducir discontinuidades artificiales mejora el aspecto de los resultados mas que el grado de detalle y similaridad respecto a la textura fuente generado por el algoritmo de Ashikhmin.

Aunque Paget puede producir resultados de menor calidad que los otros algoritmos, hemos observado que la mayoríde las veces los resultados son al menos tan buenos como los de Wei-Levoy o Ashikhmin, y si bien hay texturas con las cuales Paget genera resultados pobres, nos parece que el algoritmo de Paget funciona bien en un rango de texturas mayor que en el caso de los otros algoritmos. Debido a esto, el algoritmo de Paget es nuestro preferido para resultados donde la calidad es lo mas importante.

5.2.2 Velocidad de los algoritmos

Contrario al aspecto de calidad de resultados, la comparación de velocidad resulta muy sencilla. No nos es posible presentar resultados exactos sobre los tiempos de cómputo debido a que las pruebas fueron realizadas en dos máquinas con distintos procesadores, y utilizando dos versiones distintas del compilador Visual C++ (una de las cuales optimiza mejor que la otra). Los tiempos que se presentan a continuación son aproximados y corresponden a la ejecución de los algoritmos bajo un procesador AMD Athlon a 1.8 GHz.

El algoritmo de Ashikhmin tardó menos de un segundo en generar texturas de 128×128 pixeles. Poco mas de un segundo y medio al aplicar una iteración adicional, y solamente unos cuantos segundos para cada textura de 256×256. El tamaño de la textura fuente no influye mucho, ya que el tamaño del espacio de búsqueda solamente depende del tamaño de las vecindades.

En cambio, el algoritmo WL multiresolución tarda alrededor de 10 minutos para texturas de 128×128, entre 15 y 18 minutos con iteración adicional. Esto es con una textura fuente de 64×64. Con texturas fuente de 128×128, Wei-Levoy tarda alrededor de una hora y media para generar una textura de 256×256. Recordemos que este algoritmo puede ser altamente acelerado mediante la utilización de un árbol de cuantización vectorial (TSVQ). Según los resultados publicados, WL con TSVQ no es tan rápido como el algoritmo de Ashikhmin, pero creemos que permitiría generar texturas de 256×256 en menos de un minuto.

El algoritmo de Paget resultó tan lento que nos fué imposible terminar la implementación del mismo. Aún así Paget publica en su página de internet [13] tiempos de cómputo que varían entre 30 minutos y varias horas, utilizando distintos tamaños de vecindad para sintetizar texturas de 256×256 a partir de muestras de 128×128. No estamos seguros si estos tiempos corresponden al algoritmo paralelizado corriendo bajo una máquina multiprocesador.

La velocidad de un algoritmo es, por supuesto, un factor importante, y no hay duda de que el algoritmo de Ashikhmin es mucho mas rápido que los otros algoritmos. Sin embargo, el algoritmo WL con TSVQ no se queda atrás, y creemos que el algoritmo de Paget puede ser también altamente optimizado. De hecho, Paget parece haber adoptado algunas ideas del algoritmo de Ashikhmin para reducir el número de vecindades que se consideran al sintetizar un pixel dado, y ha implementado una versión mas rápida de su algoritmo. Esta información fué descubierta mientras se escribía este reporte, por lo que no nos fué posible estudiar dicha implementación.

5.2.3 Rango de aplicaciones

No estudiamos a fondo las aplicaciones particulares de cada algoritmo, sin embargo, podemos dar un panorama muy general según las características de cada método.

La tesis de Wei [1] no comprende solamente su algoritmo de síntesis, sino la aplicación de dicho algoritmo en múltiples problemas. Uno de ellos se conoce como *síntesis de textura restringida*, donde se pretende sintetizar la textura solamente en un área delimitada A_s de I_s. El resto de I_s no debe ser modificado. El problema es que A_s puede tener cualquier forma (no necesariamente rectangular) y debe haber ciertas condiciones de continuidad entre A_s y el resto de I_s. La síntesis de textura restringida tiene aplicaciones en edición y restauración de imágenes.

Otra aplicación realizada por Wei es lo que él llama *síntesis temporal de textura*. En este caso, el objetivo no es sintetizar una imagen que se asemeje a la textura fuente, sino toda una secuencia de imágenes $I_1, \ldots, I_n$. Esto debe hacerse de manera que la transición de I_k a I_{k+1} sea suave. Con esto es posible generar texturas animadas, tales como fuego, humo y agua.

WL también es capaz de realizar síntesis de textura en superficies. En este caso, se sintetiza la textura directamente sobre la superficie de un objeto tridimensional. Esto disminuye las distorsiones y discontinuidades ocasionadas por los algoritmos comunes de mapeo de texturas.

Otras extensiones que pueden hacerse al algoritmo Wei-Levoy consisten en la síntesis de textura a partir de múltiples texturas fuente, y una modificación del algoritmo que permite sintetizar texturas recorriendo los pixeles en cualquier orden, sin que el orden influya en los resultados. Otra característica a favor del algoritmo de Wei, es que éste puede ser acelerado de manera significativa, aunque a expensas de la calidad de los resultados. Esto permite elaborar programas hasta cierto punto interactivos, donde se utilice TSVQ para generar un versión "draft" de los resultados, y una búsqueda exhaustiva para generar un resultado final.

El algoritmo se Ashikhmin tiene como característica mas importante su velocidad de ejecución. Esto lo hace ideal para aplicaciones interactivas o incluso

en tiempo real. Además, ya que el algoritmo de Ashikhmin se deriva de WL, algunas de las aplicaciones mencionadas en los párrafos anteriores podrían ser implementadas con este algoritmo. En particular creemos que podría realizarse síntesis de textura en superficies y síntesis con múltiples fuentes. Sin embargo, podrían presentarse varios problemas al tratar de implementar síntesis restringida o síntesis temporal con este algoritmo (cosa que menciona Ashikhmin en su artículo).

Paget no menciona ninguna clase de aplicaciones en sus artículos, sin embargo, pensamos que la síntesis restringida es muy posible con el algoritmo de Paget, y probablemente también la síntesis temporal y la síntesis en superficies. Este algoritmo difícilmente se presta para aplicaciones interactivas, sin embargo, al estar basado en un modelo probabilístico, podría ser de mucha utilidad en segmentación de texturas, aplicación en la cual Wei no profundiza.

En general, el método de Wei-Levoy parece ser el más flexible a la hora de adaptarlo a distintas aplicaciones, aunque como suele ocurrir, las características de cada aplicación particular determinan cual es el método que conviene utilizar.

5.3 Ideas

Bajo la premisa de que no es posible llegar a un algoritmo que logre sintetizar cualquier textura dada, creemos que lo mejor es diseñar algoritmos especializados, donde cada uno pueda sintetizar de manera satisfactoria una o mas clases de texturas. Por lo general, las texturas tienen estructuras que se repiten, sin ser necesariamente idénticas. En la mayoría de los casos, los algoritmos estudiados logran captar dichas estructuras, y reproducirlas en la textura resultante; sin embargo, fallan al intentar reproducir tales estructuras en distintas posiciones o tamaños, de manera que se preserve la continuidad.

Esto nos lleva a pensar que se requiere algo mas que la simple búsqueda de la vecindad mas parecida. Básicamente, los algoritmos de Wei-Levoy y Ashikhmin buscan, para cada pixel r, un pixel q en la textura fuente de manera que N_q se parezca lo mas posible a N_r. Por su parte, el algoritmo de Paget construye la distribución $P(f(r)|f(s), s \in N_r)$ a partir de todos los N_q donde q es un sitio de I_a, por lo que no solamente se busca la vecindad mas parecida a N_p, sino también el valor de p que hace que (p, N_p) se parezca lo mas posible a (q, N_q), para algún sitio q en I_a.

De cualquier forma, ninguno de los algoritmos considera la comparación de N_p con distintas deformaciones de las vecindades N_q. Creemos que si se pudieran realizar tales comparaciones de una forma controlada (restringiendo la posible deformación de N_q), podrían sintetizarse texturas que contienen arreglos complejos de objetos en varias posiciones y tamaños. Por ejemplo, las texturas presentadas en la Figura 4.4.

Claramente esto supone un algoritmo mucho mas complejo que los aquí presentados. Una posible aproximación consiste en considerar solamente la rotación

de vecindades de la manera siguiente: para sintetizar un pixel p, se obtiene primero una secuencia $N_p^1, \ldots, N_p^M$, donde N_p^i se obtiene al rotar N_p en un ángulo $2\pi i/M$. Posteriormente se comparan todos los N_p^i con las vecindades de los candidatos $N_{q_1}, \ldots, N_{q_m}$ (en el caso de la búsqueda exhaustiva de Wei-Levoy, los candidatos son todos los pixeles en I_a), y se selecciona aquel candidato que minimize la distancia entre vecindades. Adicionalmente, puede ser necesario imponer restricciones de suavidad sobre los ángulos por los que se puede rotar N_p. Esto podemos hacerlo utilizando un campo $a, a(r) \in \{1, \ldots, M\}$ para todo sitio r en I_s. Podemos inicializar a con ceros y para cada pixel p en I_s encontrar q^*, i^* de la manera siguiente:

$$(q^*, i^*) = \text{argmin}_{(q,i)} \left[d(N_p^i, N_q) + \lambda \sum_{s \in \tilde{N}_p} V(i, a(s)) \right], \tag{5.1}$$

donde $i = 1, \ldots, M$ y q recorre el conjunto de candidatos para p. $\tilde{N}$ es un sistema de vecindades en a, V es una función de potencial adecuada y λ es un parámetro de granularidad. Las vecindades N_q probablemente dependerían también de i, de manera que tengan la misma forma que N_p^i.

Una vez calculados q^* e i^*, se procede a actualizar la imagen I_s y el campo a de la manera siguiente:

$$I_s(p) \leftarrow I_a(q^*) \tag{5.2}$$
$$a(p) \leftarrow i^* \tag{5.3}$$

Existen varios detalles que se deben pulir. En primer lugar, la rotación da lugar a coordenadas no enteras, lo cual hace necesario un proceso de interpolación para obtener $d(N_p^i, N_q)$ y $I_a(q^*)$. La propiedad de causalidad de las vecindades no puede mantenerse bajo la rotación, así que bien podrían utilizarse vecindades no causales en todo el proceso y realizar iteraciones adicionales para suavizar las discontinuidades. Un recocido simulado sobre λ podría utilizarse para ayudar a estabilizar a, reduciendo poco a poco la probabilidad de tener variaciones significativas en los ángulos de rotación. Por supuesto, es posible obtener N_p^i a partir de transformaciones mas generales, pero hay que tener cuidado de que las condiciones de suavidad impuestas sobre a tengan sentido en términos de las transformaciones asignadas a cada elemento de a.

Desafortunadamente, no tuvimos tiempo suficiente para poner estas ideas en prueba y pulirlas. Esperamos que alguien pueda retomarlas y experimentar con ellas, dado que aún existen muchas posibilidades que explorar en el campo de síntesis de textura.

Referencias

[1] "Texture Synthesis by Fixed Neighborhood Searching" (doctorate thesis), Li-Yi Wei, *November, 2001*

[2] "Fast Texture Synthesis using Tree-structured Vector Quantization", Li-Yi Wei, Mark Levoy, *SIGGRAPH 2000 Conference Proceedings, pages 479-488, July 2002*

[3] "Synthesizing Natural Textures", Michael Ashikhmin, *2001 ACM Symposium on Interactive 3D Graphics, pages 217-226, March 2001*

[4] "Texture Synthesis via a Noncausal Nonparametric Multiscale Markov Random Field", Rupert Paget, I. D. Longstaff, *IEEE Transactions on Image Processing, Vol. 7, No. 6, 1998*

[5] "Texture synthesis via a nonparametric Markov random field", R. Paget and I. D. Longstaff, *Proceedings of DICTA-95, Digital Image Computing: Techniques and Applications (A. Maeder and B. Lovell, eds.), vol. 1, (Brisbane, Australia)*, pags. 547-552, Australian Pattern Recognition Society, December 1995.

[6] "A nonparametric multiscale Markov random field model for synthesising natural textures", R. Paget and I. D. Longstaff, *Fourth International Symposium on Signal Processing and its Applications, vol. 2, (Gold Coast, Australia)*, pags. 744-747, ISSPA 96, August 1996.

[7] "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", Stuart Geman, Donald Geman, *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 6, No. 6*, pags 721-741, 1984.

[8] "Density estimation for statistics and data analysis", B.W. Silverman, *Chapman and Hall, London, 1986*

[9] Digital Image Processing, Bernd Jähne, *Springer,* 5^a *edición*, pag. 125-139

[10] "An Algorithm for Vector Quantizer Design", Yoseph Linde, Andrés Buzo, Robert M. Gray, *IEEE Transactions on Communications, Vol. COM-28, No. 1, Jan. 1980*

[11] Vector Quantization and Signal Compression, Allen Gersho, Robert M. Gray, *Kluwer Academic Publishers*

[12] Discrete-Time Signal Processing, Alan V. Oppenheim, Ronald W. Schafer, *Prentice Hall* pag. 80-87

[13] `http://www.vision.ee.ethz.ch/~rpaget/`

Página Web de Rupert Paget.