Proyecto Individual: Desarrollo de una aplicación para la generación de gráficos y texto

1st Gabriel Guzmán Rojas
Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
gabguzman@estudiantec.cr

I. Listado de requerimientos del sistema

El sistema debe permitir hacer una interpolación bilineal a un cuadrante de 16 opciones de una imagen 390×390, que esté en escala de grises. El formato de la imagen es .img. De la misma manera, en un .py aparte, en la carpeta del programa, se encuentra un convertidor de imagen .jpeg o .png a .img. El cuadrante seleccionado de la imagen tiene que guardarse, de la misma manera, en formato .img, y este pasaría a ensamblador para ser interpolado.

El sistema es una combinación de lenguaje de alto nivel y bajo nivel, siendo el de alto nivel utilizado únicamente para hacer una interfaz que le permita al usuario seleccionar de una imagen el cuadrante a interpolar y, después de seleccionado, mostrar el cuadrante seleccionado y el cuadrante ya interpolado, haciendo una comparación entre ellos.

A. Requerimientos funcionales

Con la anterior explicacion basica de lo que tiene que realizar el programa, se pueden desprender los siguientes requerimientos funcionales:

- Leer y guardar una imagen en formato .img
- La imagen tiene que ser de 390x390
- El usuario tiene la posibilidad de elegir un cuadrante de 16 en que se divide la imagen
- Se tiene que utilizar Ensamblador para interpolar la imagen. Las opciones disponibles son ARM, X86 o RiscV.
- Se tiene que utilizar un lenguaje de alto nivel para la visualizacion de las imagenes.
- El lenguaje de alto nivel se utiliza unicamente para mostrar la imagen de entrada, el cuadrante seleccionado y el cuadrante ya interpolado. El lenguaje de alto nivel no interpola ni hace nada a la imagen.

B. Requerimientos no funcionales

Entre los requerimientos no funcionales del sistema, tenemos que se tiene que trabajar en git, se necesita utilizar Syscall y por lo mismo se tiene que utilizar Linux.

C. Estado del Arte

La interpolación bilineal es un método de remuestreo que escala o manipula imágenes y cuadrículas. En esencia, esta técnica consiste en estimar un nuevo valor de píxel dentro de una matriz utilizando los cuatro valores de píxel más cercanos, directamente adyacentes tanto vertical como horizontalmente. Este método calcula el nuevo valor mediante una interpolación lineal, primero en una dirección y luego en la otra. Al ampliar una imagen, a menudo se encuentran intersecciones en blanco con datos desconocidos. La interpolación bilineal cubre estas lagunas considerando la proximidad de los valores de píxel conocidos alrededor del punto en blanco. La importancia de este método, o el uso que se le da, radica en su capacidad para mejorar la calidad de imagen, preservar los detalles y ofrecer buena eficiencia computacional [1].

Desde una perspectiva académica, la interpolación bilineal también se estudia por su facilidad de implementación matemática y su aplicabilidad directa en arquitecturas de computadores. Implementarla a bajo nivel (por ejemplo, en lenguaje ensamblador) permite desarrollar habilidades clave en el manejo de memoria, control de registros y optimización de algoritmos en contextos donde no se puede usar hardware gráfico especializado. En resumen, la interpolación bilineal representa una solución eficiente y efectiva para generar imágenes más suaves y detalladas, tanto en aplicaciones prácticas como en entornos educativos centrados en arquitectura y procesamiento digital.

D. Estandares y Normas

Para este proyecto se deben respetar una serie de estándares definidos en el enunciado. En primer lugar, el procesamiento de la imagen debe realizarse estrictamente en uno de los lenguajes ensamblador permitidos: ARM, x86 o RISC-V. La imagen utilizada debe tener dimensiones fijas de 390×390 píxeles (se puede hablar con el profesor, pero para efecto de este trabajo se utiliza ese tamaño), y debe estar almacenada en formato .img. Este formato de imagen representa los valores de cada píxel en una escala de grises de [0–255], donde cada valor ocupa un byte. Asimismo, el cuadrante seleccionado y procesado también debe almacenarse en formato .img antes y después de ser interpolado.

Además, se debe cumplir con las siguientes normas:

- Separación clara entre visualización (alto nivel) y procesamiento (bajo nivel).
- Uso de solo syscalls para el manejo de archivos.
- Correcto uso y desarrollo del repositorio, aplicando buenas practicas.

II. Opciones de solución

A. Basada del Enunciado

La solución principal que se pensó para realizar este problema, y de la cual se fueron derivando otras soluciones, es la solución que se nos brindó en el enunciado del proyecto mismo, que indica cómo manejar la matriz y cómo manejar los valores nuevos. La solución que se propone en base al enunciado consiste en obtener un cuadrante de 97×97 de imagen, y la imagen que se crea para interpolar va a tener un tamaño de 289×289, esto para que todos los píxeles sean cubiertos. Esto se obtuvo en base a la fórmula:

$$Tamaño = 97 + (97 - 1) * 2$$

Utilizando buenas prácticas, el objetivo es dejar bien planteado cómo se aborda el problema en ensamblador, por lo que se tiene como objetivo separar claramente en el código cada parte de la interpolación, de forma que se trabaje de una forma modular en el mismo archivo, como se ve en el siguiente diagrama.

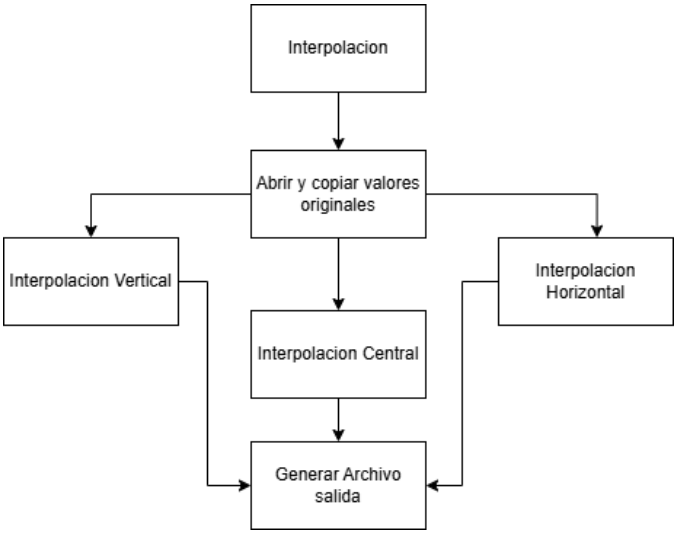

Fig. 1. Diagrama modular

En base a ese tamaño, y siguiendo el enunciado se puede expandir la matriz de forma que pase de la original de 2x2 a 4x4.

A	B
C	D

A	e	f	B
j	X	Y	m
n	W	Z	K
C	g	h	D

Fig. 2. Uso de matriz similar al enunciado

Al usar esta expansion me permite utilizar las siguientes formulas para los valores expandidos.

$$e = \frac{2A+B}{3}$$

$$f = \frac{A+2B}{3}$$

Y, de la misma manera, se utilizan estas mismas fórmulas para el cálculo de los valores verticales, pero usando los originales correspondientes. La matriz, de igual forma, se va recorriendo cumpliendo con avanzar 3 espacios, es decir, pasar a la siguiente matriz utilizando la siguiente columna, como se ve en la siguiente figura.

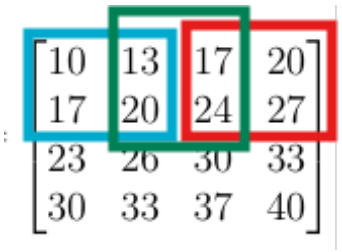

Fig. 3. Recorrido visual matrices

Donde se modifica lo que se presenta en el enunciado, es en el calculo de los valores centrales, ya que se penso que seria mas efectivo utilizar la siguiente formula:

$$X = \frac{2A+B+C+D}{6}$$

$$Y = \frac{A+2B+C+D}{6}$$

$$W = \frac{A+B+2C+D}{6}$$

$$Z = \frac{A+B+C+2D}{6}$$

Siendo esta solución la que mayor mantiene una calidad visual, una alta precisión al estar trabajando con ecuaciones más exactas y cumple con lo que se propone mayormente en el enunciado. De la misma manera, en el siguiente diagrama se puede apreciar el proceso de interpolado de la imagen:

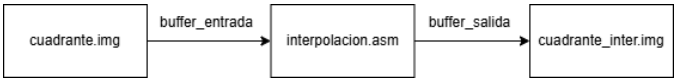

Fig. 4. Diagrama de proceso de la imagen

Como se aprecia en la figura anterior, el interpolado es bastante simple, donde el mismo recibe un buffer de entrada, que va a ser el que contiene todos los valores de píxeles originales del cuadrante original, mientras que lo que se genera gracias al interpolado es un buffer de salida de tamaño 289×289, que contiene todos los valores del nuevo cuadrante, copiando los originales en las posiciones correspondientes y agregando los valores interpolados entre los mismos.

B. Expansion 2x2 a 3x3

Esta propuesta de solución parte de la iniciativa de utilizar casi todo lo propuesto en la primera solución, que es bastante fiel al documento del proyecto dado por el profesor Luis Chavarría, en el sentido de utilizar las mismas fórmulas para calcular los valores interpolados y casi el mismo concepto de recorrido de la matriz. Pero la diferencia mayor es que, en esta propuesta, se pensó en utilizar una matriz 3×3, esto para reducir el recorrido que se hace en la matriz y optimizar el cálculo de valores intermedios, calculando únicamente uno cada par de píxeles.

Solucion: Expansión de 2×2 a 3×3

A	B
C	D

A	e	B
g	i	f
C	h	D

Fig. 5. Representacion matriz 3X3

Siendo de esta manera que, como se aprecia en la imagen, se crearían e, g, f, h, i como valores intermedios de la matriz, que, como se ve en la imagen, va diferente a la idea de que cada dos píxeles originales se inserten dos intermedios.

Una de las principales ventajas de este enfoque es que reduce considerablemente la carga computacional. Al requerir solo un cálculo de interpolación por dirección, disminuye:

- El número total de operaciones aritméticas (particularmente divisiones)
- El uso de registros temporales en ensamblador
- El tamaño total de la imagen interpolada

Otra ventaja, si así se pudiese considerar, de esta solución es que permite generar una imagen interpolada de tamaño 193×193 en vez de los 289×289 que generaría la solución 4×4. En esta solución también se tiene que contemplar que hay que seleccionar una de las dos fórmulas de los intermedios que se conocen y se utilizan en la primera solución, es decir, las que operan con 2/3 o 1/3, o cambiar la fórmula para el cálculo. Por lo tanto, la expansión de 2×2 a 3×3 representa una solución intermedia entre precisión y eficiencia. Ofrece un resultado visual razonablemente bueno, manteniendo las ventajas estructurales de la interpolación bilineal, pero con una implementación más sencilla y ligera.

C. Divisiones Exactas

Otra de las posibles soluciones que se pensó es utilizar diferentes fórmulas para obtener los valores intermedios y medios. Esto debido a que, en ensamblador y más precisamente en x86, que es el elegido para la realización del sistema, si no se utiliza punto flotante para las divisiones, el sistema suele "truncar" los valores de la división. En la fórmula original, se presentan dos divisiones diferentes, que serían el caso de 1/3 y 2/3, y en lo que se pensó fue en reducir la ecuación a una versión más simplificada para obtener los valores intermedios horizontales y verticales con solo una división para el resultado.

A	B
C	D

A	e	f	B
j	X	Y	m
n	W	Z	K
C	g	h	D

Fig. 6. Representacion de Matriz dvisiones exactas

$$e = \frac{A+B}{3}$$

$$f = \frac{A+B}{2}$$

Estas operaciones evitan el uso de multiplicaciones o sumas ponderadas, lo cual reduce el número de instrucciones y, más importante aún, evita divisiones que pueden llegar a ser propensas a error en x86 sin punto flotante.

Por otro lado, como se aprecia en la figura anterior, se genera una matriz 4×4, lo que haría que se tuviesen que calcular dos intermedios cada dos pares, que es lo propuesto en el enunciado del proyecto, pero que se calcularían los valores interpolados de una forma diferente. Esta solución presenta la ventaja de utilizar casi la misma memoria que consume la imagen original, las operaciones son más simples, por lo que los datos evitarían truncamientos, y además es la que mejor rendimiento daría.

D. Separacion de Archivos

Otra solución que se tiene contemplada, pero que no modifica directamente el método de interpolación, sino que plantea una forma alternativa de estructurar el procesamiento, consistiría en dividir el archivo .img del cuadrante completo en archivos más pequeños que pueden ser procesados individualmente. La idea central es segmentar el cuadrante seleccionado

en subimágenes de menor tamaño, por ejemplo, bloques de 8×8 o 16×16, que luego se interpolan por separado y se unen al final para formar la imagen completa. Esto tiene diversos motivos de por qué se piensa de esta manera, ya que, reiterando lo ya mencionado, la interpolación se mantendría utilizando las mismas fórmulas de la solución 1 o la solución 3, pero el resultado final variaría porque no es un solo .img. Esta propuesta viene pensada desde un punto de vista técnico, y para no exigir demasiado al sistema, más aún considerando que en mi caso se estaría utilizando una máquina virtual de Ubuntu, y el recorrido de una matriz gigante consume muchos recursos, por lo que esta solucion permite tener más control en el consumo de recursos, el manejo de memoria y la carga de registros. De una forma resumida, esta solución funcionaría de la siguiente manera:

1) Se seleciona el cuadrante a trabajar en python
2) Se separa el cuadrante en diferentes archivos .img. Ejemplo: Bloque1.img
3) Cada uno de estos archivo se pasan a interpolar a ensamblador
4) Una vez interpolados y generados por ensamblador pasan a otro archivo de python
5) El archivo de python unicamente une estos archivos en un solo .img.
6) Este archivo final .img seria el que se mostraria en la interfaz de python.

En este diagrama se puede apreciar mejor la idea.

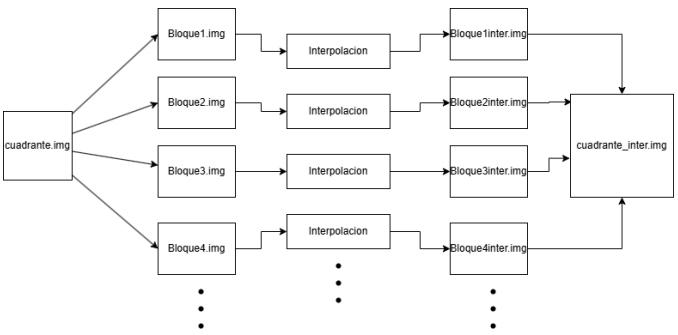

Fig. 7. Diagrama manejo Separacion de Archivos

E. Valores escritos directamente en el archivo

Una quinta solución propuesta, que de la misma forma que la cuarta solución no tocaría directamente el método de interpolación, consiste en eliminar el uso de un buffer de salida para almacenar la imagen interpolada. A diferencia de cómo funcionan las demás soluciones, en su lugar se estaría realizando la interpolación en tiempo real, escribiendo directamente al archivo de salida .img conforme se van generando los valores interpolados y se recorren los valores originales. Por lo que, de forma resumida, en esta propuesta lo que se busca es que, al igual que en la cuarta propuesta se divide el archivo de entrada en diferentes archivos para el ensamblador, aquí en vez de ser diferentes archivos, se estaría copiando una parte de la matriz, es decir, una pequeña matriz 2×2, como se plantea en las diferentes soluciones, pero que una vez interpolada, en vez de guardarse en el buffer, se carga directamente en el archivo de salida inmediatamente.

Este diagrama ilustra el proceso

Fig. 8. Diagrama escritura directa

Este enfoque permite reducir considerablemente el uso de memoria, porque no se estaría guardando un buffer de la imagen interpolada con todos los valores, sino que, al ya escribirse directamente en el archivo de salida, se ahorra ese consumo de recursos extra. Además, como se mencionó en la cuarta solución, en la realización de este proyecto se va a usar una máquina virtual, en donde la RAM es crítica y lo mejor es tener programas poco exigentes. También, otro punto por el que esta solución se vuelve útil es que permite hacer el sistema útil para cualquier tamaño de imagen, por lo que, si se expande a un tamaño incorrecto, se puede modificar rápidamente. Por lo tanto, esta solución representa una alternativa eficiente y robusta si se prioriza el rendimiento del sistema y la gestión óptima de memoria.

III. Comparacion propuestas de solucion

Cada una de las soluciones tiene sus ventajas y sus desventajas, por lo que, comparando las tres opciones propuestas para implementar el algoritmo de interpolación bilineal, se tienen que tomar en cuenta criterios técnicos y teóricos relevantes como la fidelidad al modelo de forma matemática, la eficiencia computacional en ensamblador, la precisión numérica y el efecto visual sobre la imagen resultante.

La primera solución que se propuso fue la de interpolación bilineal con expansión 2×2 a 4×4, que sigue fielmente lo que se propone en el enunciado del proyecto, aplicando ponderaciones exactas de 2/3 y 1/3 en los píxeles intermedios horizontales y verticales, y 1/6 combinando los cuatro píxeles A, B, C y D para calcular los valores centrales de la matriz generada. Esta implementación mantiene la coherencia con el modelo teórico de interpolación bilineal 2D, respetando las proporciones y generando transiciones suaves en todas direcciones. Sin embargo, esta opción implica mayor complejidad computacional, mayor presión sobre los registros y uso frecuente de divisiones enteras, pero realmente es la única que garantiza resultados visualmente precisos y matemáticamente consistentes. Basándonos en el contexto del proyecto, donde la calidad visual y la adherencia al modelo son prioritarios, esta es la solución más adecuada.

La segunda solución que se propuso, que fue la de interpolación bilineal con expansión 2×2 a 3×3, se basa en un modelo de interpolación mínima, en el cual solo se genera un valor intermedio entre cada par de píxeles, tanto en dirección horizontal como vertical, y un único valor central.

Esta solución permite reducir significativamente el número de cálculos y el uso de memoria, por lo que es altamente eficiente desde la perspectiva de arquitectura de la computadora y la exigencia a la misma. Sin embargo, desde el punto de vista teórico, esta solución no cumple completamente con el modelo de interpolación bilineal, ya que ignora las relaciones diagonales entre píxeles. Visualmente, esto puede generar artefactos como bordes angulosos o transiciones abruptas, y en la parte matemática puede generar resultados que van a afectar directamente a la calidad visual, principalmente la calidad del suavizado.

La tercera solución propuesta, que es la correspondiente a divisiones simplificadas, propone reemplazar las fórmulas bilineales exactas por aproximaciones fácilmente evaluables en x86, como (A + B)/2 o (A + B)/3. Este enfoque va a permitir maximizar la eficiencia, ya que puede evitar divisiones complejas y, en su lugar, utilizar desplazamientos binarios o sumas simples. Esta opción no es tan exigente al procesador ni a los registros, y el recorrido de memoria es similar al de la primera solución. Pero este enfoque rompe completamente con la fidelidad al modelo bilineal, ya que no considera las posiciones relativas ni los pesos correctos de los píxeles, lo que introduce errores acumulativos debido al uso de los mismos valores en otras matrices y distorsiones visuales. Desde una perspectiva teórica, no puede ser considerado como interpolación bilineal propiamente dicha, sino como una interpolación media aproximada.

La cuarta solución propuesta fue la de separación de archivos, que se basa en la separación del archivo .img en múltiples archivos más pequeños, los cuales se interpolan por separado y luego se recombinan para formar la imagen final expandida. Esta solución no modifica realmente la interpolación propuesta, sino que organiza el proceso de manera modular. El objetivo es dividir para hacer el código más manejable, ya que al procesar cada uno de forma independiente mediante el mismo código ensamblador, y al final unirlos, se reduce significativamente la presión sobre los registros y se simplifica el manejo de índices, ya que se trabaja con estructuras más pequeñas y localizadas. Además, permite reutilizar un mismo algoritmo de interpolación sin necesidad de adaptarlo para matrices grandes, por lo que se facilita el proceso de depuración y, hasta pensando en los temas vistos en arquitectura, se permitiría paralizar si fuese lo que se estuviese pidiendo. Pero el problema de esta solución es que se requiere que el lenguaje de alto nivel se encargue de dividir y recombinar los bloques, lo cual haría un paso más en el flujo de trabajo, lo que podría llegar a complicarlo, porque puede dar problemas el reconocimiento de los bloques a la hora de unirlos y también puede provocar muchos píxeles vacíos al unirlos, lo que daría una mala imagen. Aun así, sigue siendo una solución sólida, eficiente y escalable que permite mantener una buena calidad visual y que, además, distribuye mejor la carga de trabajo de cada parte.

La quinta solución propuesta consiste en eliminar completamente el uso de un buffer de salida para la imagen expandida y realizar la interpolación escribiendo directamente los valores, tanto originales como interpolados, directamente en el archivo de salida. Esta solución mantiene el buffer de entrada para acceder a los píxeles originales, pero evita almacenar toda la imagen interpolada en memoria, lo cual representa una optimización significativa del uso de recursos. Esta propuesta permite reducir la presión sobre el segmento .bss del programa y minimizar el uso de registros, lo cual es ideal cuando se cuenta con recursos limitados para trabajar, e inclusive cuando la arquitectura es la que tendría que exigirle mucho al sistema para ejecutarlo. También esta solución es perfecta cuando se desea mayor eficiencia. Además, hace posible procesar imágenes de mayor tamaño sin modificar la estructura general del código. Sin embargo, esta solución exige un mayor control sobre las posiciones de escritura en el archivo, lo que la vuelve más compleja de implementar en ensamblador. Además, hay que sumar que cada error en la interpolación va a ir directamente en el archivo, ya que no hay una etapa de revisión previa antes de guardar. A pesar de todas las dificultades mencionadas, esta opción es altamente eficiente, especialmente desde una perspectiva de gestión de memoria, y representa una alternativa avanzada para optimizar el rendimiento del sistema sin afectar el método original.

En términos de rendimiento, las soluciones 2 y 3 ofrecen mayor eficiencia y menor complejidad, pero comprometen la calidad. Por otro lado, la solución 4 puede llegar a complicar más el problema, ya que sumarle el paso de juntar los archivos puede derivar en pérdida de píxeles, lo que no sería bueno. La solución 5, en cambio, a las otras 3 mencionadas, es la que mejor rendimiento presentaría, pero tiene el mismo problema que tiene la solución 4, que es la complejidad de implementarla y el hecho de que, al trabajar sobre el archivo directamente, cualquier error será guardado. Mientras que por otro lado, la solución 1 es la más viable y, aunque sea más costosa computacionalmente, cumple todos los estándares académicos, técnicos y funcionales del proyecto.

IV. Seleccion de la propuesta final

Después de un análisis exhaustivo para la selección de la solución que mejor se adapta al proyecto, se decidió usar la primera solución, es decir, la expansión de 2×2 a 4×4. Esto por diversos factores que, como se mencionó en el punto anterior, inclinan más la balanza a esa selección, como por ejemplo, el simple hecho de que, así como se propuso esta solución, es como viene la propuesta del enunciado del proyecto, lo que daría más viabilidad y efectividad porque así es como lo planteó el profesor. Además, es la solución que da una mejor calidad de imagen y que es más precisa, comparándola con las soluciones 2 y 3, que son las que alteran principalmente el código de interpolación. Comparándolo más profundamente con las soluciones 4 y 5, aunque estas mismas se pudiesen implementar con esta solución, se optó mejor por mantenerse con la solución original porque, reiterando el punto anterior, ambas de estas implementaciones tienen su complicación a la hora de programar, ya sea agregando un paso extra en lenguaje de alto nivel o quitando un paso en ensamblador que realmente puede garantizar más seguridad

del código generado, porque se pueden evitar ciertos errores que se pueden arrastrar al no usar un buffer. De la misma manera, como se indicó en esta, el código planea hacerse de una forma simple y no tan complicada, por lo que se mantendría el modelo original de un simple archivo .img que se volvería para ensamblador un buffer de tamaño 97×97, mientras que la imagen de salida, también en formato .img, también va a ser un buffer de salida, pero de tamaño 289×289. La manera más simple de implementar esto es, de primeras, abrir el buffer de entrada y recorrerlo copiando los valores originales en el buffer de salida para poder así comenzar con la interpolación, empezando a recorrer con los índices de la matriz filas y columnas, permitiendo así saber que cada dos pares van a ir dos valores interpolados, por lo que, para ese manejo, se avanzan 3 posiciones para poder iniciar la siguiente matriz con los valores de la última columna de ese rango.

REFERENCES

[1] Cloudinary, “What is Bilinear Interpolation?,” *Cloudinary Glossary*. [Online]. Available: https://cloudinary.com/glossary/bilinear-interpolation.

[2] L. Chavarria, ”Proyecto Individual: Desarrollo de una aplicación para la generación de gráficos y texto”, Instituto Tecnológico de Costa Rica, Cartago, Costa Rica.