**Instituto Politécnico Nacional  
 Escuela Superior de Cómputo  
Ingeniería en   
Sistemas Computacionales**

**Unidad de Aprendizaje:**

**VISIÓN POR COMPUTADORA**

**Grupo: 7CV6**

**Práctica 7: Transformaciones en el dominio de la frecuencia**

**Alumno:**

**Bautista Ríos Alfredo**

Docente: Saul De la O Torres

# Introducción

En el dominio de la frecuencia, una imagen se representa como la combinación de senos y cosenos (o exponenciales complejas) de distintas frecuencias espaciales. La Transformada de Fourier 2D permite separar componentes de baja frecuencia (variaciones suaves) y alta frecuencia (bordes, detalles finos), lo cual es clave para análisis y filtrado. En esta práctica se implementa una aplicación en C++/GTKmm (arquitectura MVC) que calcula la FFT 2D (directa) y la IFFT (inversa) sobre imágenes cuadradas cuyo tamaño sea potencia de dos, mostrando la magnitud (en escala logarítmica) y la fase, con opción de centrar el espectro mediante un shift de cuadrantes.

# Objetivos

* Crear una interfaz para cargar una imagen y ajustarla automáticamente a N×N, donde N es potencia de dos.
* Aplicar la Transformada Rápida de Fourier 2D (FFT) a la imagen (directa) y visualizar magnitud (log) y fase.
* Aplicar la Transformada Inversa (IFFT) para reconstruir la imagen espacial a partir del espectro.
* Validar que la reconstrucción conserva la información (diferencias mínimas por redondeo/clamp) y que el espectro responde a la estructura de la imagen.

# Marco teórico

Requisito N×N potencia de dos. La FFT radix-2 requiere que la longitud de la señal sea 2^k. Para imágenes se usa la separabilidad, aplicando FFT 1D en filas y luego en columnas. Por ello el proyecto ajusta la imagen a N×N con N = 2^k mediante reescalado bilinear.

Transformada de Fourier 2D. Para una imagen f(x,y), la DFT 2D produce un espectro complejo F(u,v). La magnitud |F| indica la energía por frecuencia, y el ángulo arg(F) representa la fase. En visualización se usa log(1+|F|) para comprimir el rango dinámico.

Shift de cuadrantes. La DFT coloca la componente DC (baja frecuencia) en la esquina (0,0). Para una lectura más intuitiva, se intercambian cuadrantes I↔III y II↔IV, centrando las bajas frecuencias en el medio de la imagen.

Transformada inversa. La IFFT reconstruye la imagen espacial a partir del espectro. En este proyecto, la normalización se realiza dentro de la FFT 1D cuando dir=-1, por lo que la IFFT 2D queda escalada por 1/N^2 tras las dos pasadas (filas y columnas).

# Metodología y desarrollo

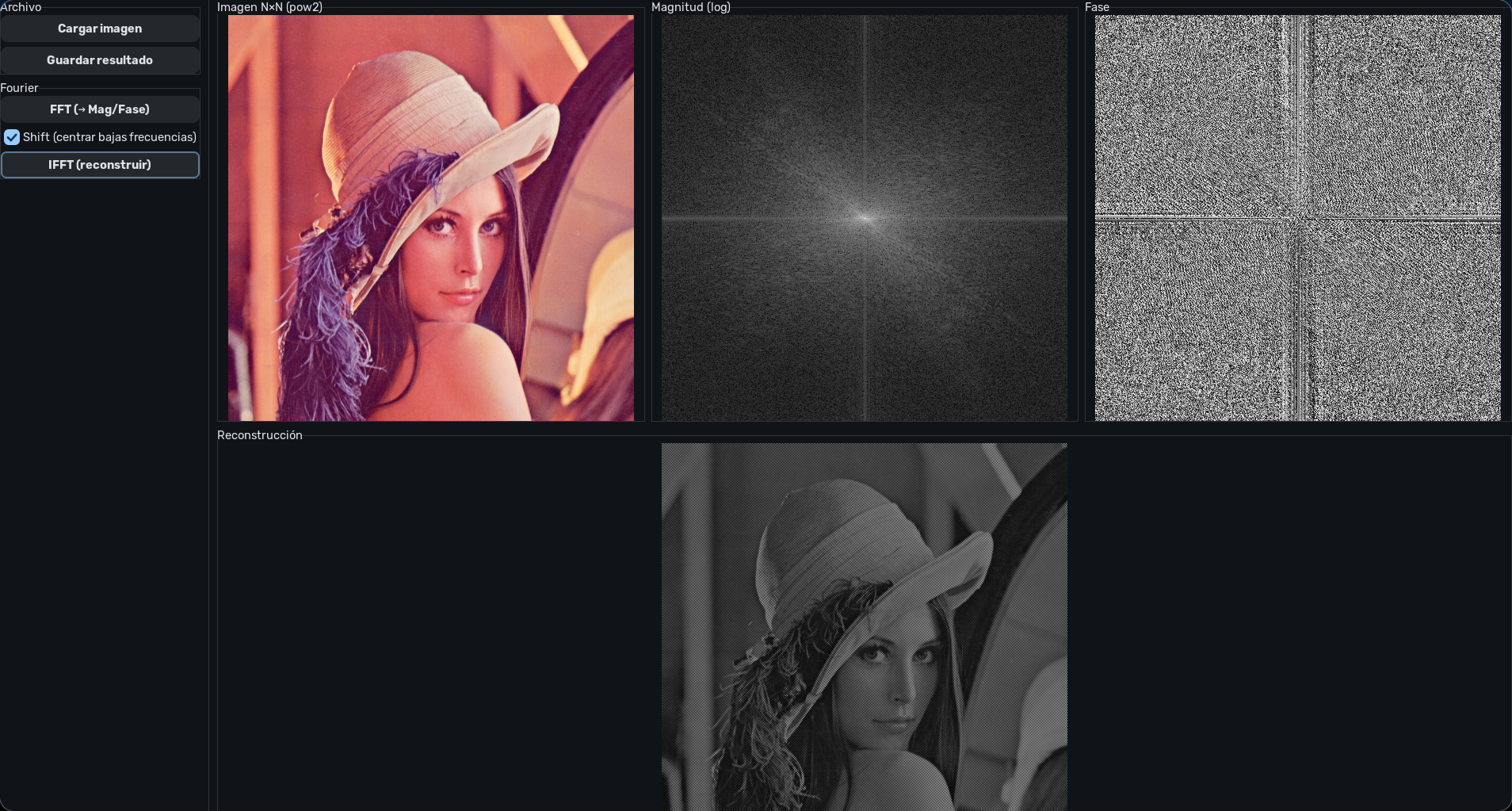
1. Compilación : instalar gtkmm3 y cmake; compilar con CMake y ejecutar el binario `fourier`.
2. Flujo: (1) cargar imagen, (2) ajustar a N×N potencia de dos, (3) convertir a escala de grises normalizada [0,1], (4) promover a buffer complejo F(x,y)=g(x,y)+j·0, (5) aplicar FFT 2D, (6) opcionalmente aplicar shift, (7) extraer magnitud y fase y renderizarlas como imágenes, (8) aplicar IFFT para reconstrucción, (9) guardar resultados.
3. Reglas de entrada: el programa asume que la imagen de trabajo ya es cuadrada y pow2 después del ajuste automático. N se calcula como la siguiente potencia de dos del máximo entre ancho y alto.

# Resultados y validación

* Validación de tamaño: al cargar, la imagen se reescala a N×N con N = NextPow2(max(w,h)). Esto garantiza la condición de potencia de dos solicitada.
* Validación de FFT: al ejecutar FFT, se generan dos salidas visuales: magnitud logarítmica y fase. Con Shift activado, la energía de bajas frecuencias tiende a concentrarse en el centro.
* Validación de IFFT: tras IFFT se reconstruye la imagen usando la parte real del buffer complejo (clamp a [0,1]). En condiciones normales, la reconstrucción debe verse muy similar a la imagen de entrada ajustada.

# Evidencia de salida

* Ventana principal: botones \*\*Cargar imagen\*\*, \*\*FFT\*\*, \*\*IFFT\*\*, \*\*Guardar resultado\*\* y checkbox \*\*Shift (centrar bajas frecuencias)\*\*.
* Panel de visualización con marcos: \*\*Imagen N×N (pow2)\*\*, \*\*Magnitud (log)\*\*, \*\*Fase\*\*, \*\*Reconstrucción\*\*.



# 

# 

# 

# Bloques de código importantes

## 1) Cálculo de potencia de dos (src/utils/ImgUtils.cpp)

int NextPow2(int v){ int p=1; while(p<v) p<<=1; return p; }  
bool IsPow2(int v){ return v>0 && (v&(v-1))==0; }

## 2) Ajuste automático a N×N pow2 (src/utils/ImgUtils.cpp)

Glib::RefPtr<Gdk::Pixbuf> ToSquarePow2(const Glib::RefPtr<Gdk::Pixbuf>& src, int& N\_out){  
 int w=src->get\_width(), h=src->get\_height();  
 int M = std::max(w,h);  
 int N = NextPow2(M);  
 N\_out = N;  
 auto scaled = src->scale\_simple(N,N, Gdk::INTERP\_BILINEAR);  
 return scaled;  
}

## 3) FFT 1D radix-2 (src/utils/FFT.cpp)

void fft1d(std::vector<cd>& a, int dir){  
 const size\_t n = a.size();  
 int lg = 0; while((1u<<lg) < n) ++lg;  
  
 // Bit-reversal  
 for(size\_t i=0;i<n;++i){  
 size\_t j = reverse\_bits((unsigned)i, lg);  
 if(i<j) std::swap(a[i], a[j]);  
 }  
  
 const double PI = std::acos(-1.0);  
 for(size\_t len=2; len<=n; len<<=1){  
 double ang = 2\*PI/len \* (dir>0? -1 : 1); // dir: +1 FFT, -1 IFFT  
 cd wlen = std::polar(1.0, ang);  
 for(size\_t i=0; i<n; i+=len){  
 cd w = 1.0;  
 for(size\_t j=0;j<len/2;++j){  
 cd u = a[i+j];  
 cd v = a[i+j+len/2] \* w;  
 a[i+j] = u+v;  
 a[i+j+len/2] = u-v;  
 w \*= wlen;  
 }  
 }  
 }  
 if(dir<0){  
 for(size\_t i=0;i<n;++i) a[i] /= (double)n;  
 }  
}

## 4) FFT 2D por separabilidad (src/utils/FFT.cpp)

void fft2d(std::vector<cd>& img, int N, int dir){  
 // Rows  
 std::vector<cd> temp(N);  
 for(int y=0;y<N;++y){  
 for(int x=0;x<N;++x) temp[x] = img[y\*N+x];  
 fft1d(temp, dir);  
 for(int x=0;x<N;++x) img[y\*N+x] = temp[x];  
 }  
 // Cols  
 for(int x=0;x<N;++x){  
 for(int y=0;y<N;++y) temp[y] = img[y\*N+x];  
 fft1d(temp, dir);  
 for(int y=0;y<N;++y) img[y\*N+x] = temp[y];  
 }  
 // Global normalization already done by two passes when dir<0 (1/N then 1/N), so scale by 1/N again:  
 if(dir<0){  
 double invN = 1.0; // already normalized fully by two 1D ifft passes  
 (void)invN;  
 }  
}

## 5) Shift de cuadrantes (src/utils/FFT.cpp)

void fftshift2d(std::vector<cd>& F, int N){  
 int h = N/2;  
 for(int y=0;y<h;++y){  
 for(int x=0;x<h;++x){  
 std::swap(F[y\*N+x], F[(y+h)\*N + (x+h)]); // I <-> III  
 std::swap(F[y\*N + (x+h)], F[(y+h)\*N + x]); // II <-> IV  
 }  
 }  
}

## 6) Ejecución de FFT y render de magnitud/fase (src/model/ImageModel.cpp)

bool ImageModel::do\_fft(bool shift, std::string& err){  
 if(!img\_in\_){ err="Carga primero una imagen."; return false; }  
 int N = img\_in\_->get\_width();  
 N\_ = N;  
 // To grayscale double  
 auto g = ImgUtils::PixbufToGray(img\_in\_);  
 // Promote to complex  
 F\_.assign(N\*N, cd(0,0));  
 for(int i=0;i<N\*N;++i) F\_[i] = cd(g[i], 0.0);  
  
 // 2D FFT  
 FFT::fft2d(F\_, N, +1);  
  
 // Visualizations  
 if(shift) FFT::fftshift2d(F\_, N);  
 std::vector<double> mag, phase;  
 FFT::mag\_phase(F\_, N, mag, phase);  
 img\_mag\_ = ImgUtils::MagToPixbuf(mag, N);  
 img\_phase\_ = ImgUtils::PhaseToPixbuf(phase, N);  
 img\_result\_.reset();  
 return true;  
}

## 7) IFFT y reconstrucción (src/model/ImageModel.cpp)

bool ImageModel::do\_ifft(std::string& err){  
 if(F\_.empty()){ err="No hay espectro FFT disponible. Ejecuta primero FFT."; return false; }  
 int N = N\_;  
 // If magnitude was shifted for display, assume F\_ está en el mismo estado (sin deshacer shift)  
 // Para IFFT correcta, no es necesario deshacer shift si no se modificó F\_. (Solo era visualización.)  
 FFT::fft2d(F\_, N, -1); // IFFT  
 // Extrae parte real normalizada  
 std::vector<double> rec(N\*N);  
 for(int i=0;i<N\*N;++i){  
 double v = F\_[i].real();  
 rec[i] = std::clamp(v, 0.0, 1.0);  
 }  
 img\_result\_ = ImgUtils::GrayToPixbuf(rec, N);  
 return true;  
}

## 8) Wiring MVC (src/main.cpp)

#include <gtkmm/application.h>  
#include "view/MainWindow.hpp"  
#include "model/ImageModel.hpp"  
#include "controller/AppController.hpp"  
  
int main(int argc, char\* argv[]) {  
 auto app = Gtk::Application::create(argc, argv, "org.example.fourier");  
 auto model = std::make\_shared<ImageModel>();  
 MainWindow win(model);  
 AppController controller(model, win);  
 return app->run(win);  
}

# Conclusión

Se construyó una aplicación MVC en C++/GTKmm para aplicar la FFT 2D (directa e inversa) sobre imágenes, cumpliendo el requisito de trabajar con imágenes cuadradas de tamaño potencia de dos mediante un ajuste automático por reescalado bilinear. La FFT se implementa con un esquema radix-2 in-place y se extiende a 2D por separabilidad (filas y columnas). Como resultados, el sistema visualiza magnitud (en log para mejorar el contraste del espectro) y fase, y permite centrar el espectro con un shift de cuadrantes. Finalmente, la IFFT reconstruye la imagen espacial a partir del espectro, lo que sirve como validación práctica de la correcta implementación de la transformada.

# Referencias

* R. C. Gonzalez, R. E. Woods, Digital Image Processing, Pearson.
* W. K. Pratt, Digital Image Processing, Wiley.
* Documentación de GTKmm 3 (widgets, Pixbuf) y Cairo/Pango (render).