Edge-Aware Image Scaling (EASR) Implementation

Abstract

This document presents a comprehensive implementation of an Edge-Aware Super Resolution (EASR) algorithm. The system combines advanced image processing techniques including edge-aware sharpening, anisotropic diffusion, and intelligent prefiltering to achieve high-quality image scaling. The implementation consists of two modules: the core algorithm library (`easr.py`) and a demonstration script (`demo_usage.py`).

Contents

1 Core Algorithm Library (easr.py)

This module contains the core image processing functions for high-quality image scaling. Key features include:

- Edge-aware unsharp masking for selective sharpening
- Anisotropic diffusion for edge-preserving smoothing
- Intelligent prefiltering for anti-aliasing
- Sobel edge detection for edge weighting

1.1 Source Code

```python
import numpy as np
from PIL import Image

def _to_np(img):
    if isinstance(img, Image.Image):
        arr = np.array(img.convert("RGB"), dtype=np.float32) / 255.0
    else:
        arr = img.astype(np.float32)
        if arr.max() > 1.0:
            arr /= 255.0
    return arr

def _to_img(arr):
    arr = np.clip(arr * 255.0, 0, 255).astype(np.uint8)
    return Image.fromarray(arr)

def load_image(path):
    return Image.open(path).convert("RGB")
```

```
def save_image(img, path):
    if isinstance(img, np.ndarray):
        img = _to_img(img)
    img.save(path)


def box_blur_np(img, k=3):
    arr = _to_np(img)
    if k <= 1:
        return _to_img(arr)
    pad = k//2
    # horizontal pass: pad width only
    arr_pad_w = np.pad(arr, ((0,0),(pad,pad),(0,0)), mode="edge")
    kernel = np.ones((k,), dtype=np.float32) / k
    tmp = np.apply_along_axis(lambda m: np.convolve(m, kernel, mode="valid"), axis=1,
    arr=arr_pad_w)
    # vertical pass: pad height only
    tmp_pad_h = np.pad(tmp, ((pad,pad),(0,0),(0,0)), mode="edge")
    out = np.apply_along_axis(lambda m: np.convolve(m, kernel, mode="valid"), axis=0,
    arr=tmp_pad_h)
    return _to_img(out)

def sobel_edges_gray(img):
    arr = _to_np(img)
    gray = 0.299*arr[:,:,0] + 0.587*arr[:,:,1] + 0.114*arr[:,:,2]
    Kx = np.array([[-1,0,1],[-2,0,2],[-1,0,1]], dtype=np.float32)
    Ky = np.array([[-1,-2,-1],[0,0,0],[1,2,1]], dtype=np.float32)
    def conv2d(mat, K):
        pad = 1
        m = np.pad(mat, ((pad,pad),(pad,pad)), mode="edge")
        H,W = mat.shape
        out = np.zeros_like(mat)
        for i in range(H):
            for j in range(W):
                out[i,j] = np.sum(m[i:i+3, j:j+3]*K)
        return out
    gx = conv2d(gray, Kx)
    gy = conv2d(gray, Ky)
    mag = np.sqrt(gx*gx + gy*gy)
    mag /= (mag.max() + 1e-8)
    return mag

def unsharp_mask_edgeaware(img, blur_k=5, amount=0.5, edge_weight=None):
    arr = _to_np(img)
    blurred = _to_np(box_blur_np(img, k=blur_k))
    high = arr - blurred
    if edge_weight is None:
        edge_weight = sobel_edges_gray(img)
    edge_weight = np.expand_dims(edge_weight, 2)
    sharpened = arr + amount * high * (0.25 + 0.75*edge_weight)
    return _to_img(np.clip(sharpened, 0, 1))

def anisotropic_diffusion(img, niter=5, kappa=20.0, gamma=0.15):
    arr = _to_np(img)
    H,W,_ = arr.shape
    L = 0.299*arr[:,:,0] + 0.587*arr[:,:,1] + 0.114*arr[:,:,2]
    for _ in range(int(niter)):
        Lp = np.pad(L, ((1,1),(1,1)), mode="edge")
        N = Lp[:-2,1:-1] - L
        S = Lp[2:,1:-1]  - L
        E = Lp[1:-1,2:]  - L
        W = Lp[1:-1,:-2] - L
        cN = np.exp(-(N/kappa)**2)
        cS = np.exp(-(S/kappa)**2)
        cE = np.exp(-(E/kappa)**2)
        cW = np.exp(-(W/kappa)**2)
        L = L + gamma*(cN*N + cS*S + cE*E + cW*W)
    eps = 1e-6
    Y = L
    denom = (0.299*arr[:,:,0] + 0.587*arr[:,:,1] + 0.114*arr[:,:,2] + eps)
    ratio = (arr + eps) / np.expand_dims(denom,2)
    out = np.clip(ratio * np.expand_dims(Y,2), 0, 1)
```

```python
    return _to_img(out)

def resize_image(img, scale, prefilter=True, method_up="bicubic", method_down="lanczos",
                 sharpness=0.4, aa_strength=0.3, edge_preserve=0.6, diffusion_iters=3):
    if not isinstance(img, Image.Image):
        img = _to_img(_to_np(img))
    W, H = img.size
    target = (max(1, int(W*scale)), max(1, int(H*scale)))
    if scale < 1.0 and prefilter:
        k = max(3, int(2/scale)+1)
        img = box_blur_np(img, k=k)
    if scale >= 1.0:
        base = img.resize(target, Image.BICUBIC if method_up=="bicubic" else Image.LANCZOS)
    else:
        base = img.resize(target, Image.LANCZOS if method_down=="lanczos" else Image.BICUBIC)
    if sharpness > 1e-6 and scale >= 1.0:
        edges = sobel_edges_gray(base)
        amount = sharpness * (0.25 + 0.75*edge_preserve)
        base = unsharp_mask_edgeaware(base, blur_k=5, amount=amount, edge_weight=edges)
    if aa_strength > 1e-6:
        iters = max(0, int(diffusion_iters * (0.5 + 0.5*aa_strength)))
        if iters > 0:
            base = anisotropic_diffusion(base, niter=iters, kappa=25.0, gamma=0.15)
    return base

def process(path_in, path_out, scale=2.0, sharpness=0.4, aa_strength=0.3, edge_preserve=0.6,
            diffusion_iters=3, prefilter=True):
    img = load_image(path_in)
    out = resize_image(img, scale=scale, prefilter=prefilter, sharpness=sharpness,
                       aa_strength=aa_strength, edge_preserve=edge_preserve, diffusion_iters=diffusion_iters)
    save_image(out, path_out)
```

Listing 1: easr.py - Core image scaling functions

1.2 Algorithm Overview

The core scaling algorithm follows this workflow:

1. **Prefiltering (Downscaling only):** Applies box blur when $scale < 1.0$ to prevent aliasing
2. **Base Scaling:** Uses PIL's high-quality resampling (BICUBIC/LANCZOS)
3. **Edge-Aware Sharpening (Upscaling only):**
    - Detects edges using Sobel operator
    - Applies unsharp masking with edge weighting
    - Stronger sharpening on edges than flat areas
4. **Edge-Preserving Smoothing:**
    - Uses anisotropic diffusion for noise reduction
    - Preserves edges while smoothing textures

2 Demonstration Script (demo_usage.py)

This script demonstrates usage of the EASR algorithm:

- Generates a comprehensive test pattern with:
    - Checkerboard pattern
    - Color gradients

– Text overlay

- Demonstrates both downscaling (50%) and upscaling (200%)
- Shows parameter tuning for different scaling scenarios

2.1 Source Code

```python
from PIL import Image, ImageDraw, ImageFont
import numpy as np, os
from easr import resize_image, save_image

# === Change this to your desired output folder ===
out_dir = r"D:\Program Files\PyCharm 2025.1.1\PythonProject\easr_outputs"
os.makedirs(out_dir, exist_ok=True)

def make_test_image(W=320, H=180):
    img = Image.new("RGB", (W,H), (128,128,128))
    px = img.load()
    tile = 16
    for y in range(H):
        for x in range(W):
            c = 64 if ((x//tile + y//tile) % 2 == 0) else 192
            r = int(c + 63*np.sin(2*np.pi*x/W))
            g = int(c)
            b = int(c + 63*np.cos(2*np.pi*y/H))
            px[x,y] = (max(0,min(255,r)), max(0,min(255,g)), max(0,min(255,b)))
    dr = ImageDraw.Draw(img)
    msg = "EASR demo"
    try:
        fnt = ImageFont.truetype("DejaVuSans.ttf", 24)
    except:
        fnt = ImageFont.load_default()
    dr.text((10,10), msg, font=fnt, fill=(255,255,255))
    return img

if __name__ == "__main__":
    src = make_test_image()
    save_image(src, os.path.join(out_dir, "src.png"))
    down = resize_image(src, scale=0.5, sharpness=0.0, aa_strength=0.8, diffusion_iters
=2)
    save_image(down, os.path.join(out_dir, "down_0p5.png"))
    up = resize_image(src, scale=2.0, sharpness=0.6, aa_strength=0.3, edge_preserve=0.7,
 diffusion_iters=4)
    save_image(up, os.path.join(out_dir, "up_2x.png"))
    print(f"Wrote images to: {out_dir}")
```

Listing 2: demo_usage.py - Demonstration of EASR capabilities

2.2 Test Pattern Features

The generated test image includes:

- **Checkerboard Pattern:** Tests aliasing and moiré effects
- **Sinusoidal Gradients:** Evaluates color preservation
- **Text Overlay:** Assesses edge sharpness preservation
- **High-Frequency Elements:** Challenges the scaling algorithm

2.3 Parameter Recommendations

The system follows a sophisticated image processing pipeline:

1. Input image conversion to normalized floating point array
2. Context-aware processing path selection:

Parameter	Downscaling (0.5x)	Upscaling (2x)
Sharpness	0.0	0.6
AA Strength	0.8	0.3
Edge Preserve	N/A	0.7
Diffusion Iterations	2	4
Prefilter	Enabled	Enabled

Table 1: Optimal parameters for different scaling operations

- Downscaling path: Anti-aliasing prefilter
- Upscaling path: Edge-aware enhancement

3. Multi-stage processing with intermediate representations

4. Output conversion to standard image formats

3 Conclusion

The EASR implementation provides:

- Superior edge preservation compared to traditional scaling
- Adaptive processing based on scaling direction
- Parameterized control over sharpness and anti-aliasing
- Comprehensive test pattern generator for validation

This implementation demonstrates professional-grade image scaling techniques suitable for computer vision, digital photography, and graphics applications.