Lab 5: SIMD

Vector programming for image processing (2)

In this lab, we continue developing and testing vector programming examples applied to images. We begin with **image rotation** for both grayscale and color images. Next, we demonstrate image creation through two simple examples: drawing **squares** and **lines**. Finally, we present two examples of dynamic (animated) displays using **OpenGL**.

5.1 Image rotation

5.1.1 Image rotation - grayscale

The following example shows simple image rotation by 90 degrees.

5.1.1.1 Assembly code

```
-------------------------------------------------------------------------------------------
    .text
    .globl  rotate90_gray_rvv
    .align  2

rotate90_gray_rvv:
    li      t6, 512             # N
    li      t0, 0               # r = 0
    mv      t5, t6              # stride bytes for vsse8.v: N
1:  beq     t0, t6, 9f          # if (r == N) return
    # srow = src + r*N
    mul     t1, t0, t6
    add     t2, a0, t1          # t2 = srow
    # dst_ptr = dst + (N-1 - r)   (starting at column 0)
    addi    t3, t6, -1          # N-1
    sub     t3, t3, t0          # N-1-r
    add     t3, a1, t3          # t3 = &dst[0*N + (N-1-r)]
    mv      t4, t6              # remaining columns in this row: rem = N

2:  beqz    t4, 8f              # done this row?

    # Set vl (elements) for this chunk (SEW=8)
    vsetvli t1, t4, e8, m1, ta, ma   # t1 = vl
    mv      a2, t1                   # save vl
    # Load 'vl' bytes from source row
    vle8.v  v0, (t2)
    # Strided store into destination column: stride == N bytes
    vsse8.v v0, (t3), t5
    # Advance pointers: srow += vl; dst_ptr += vl*N; rem -= vl
    add     t2, t2, a2              # srow += vl
    mul     t1, a2, t6              # t1 = vl*N
    add     t3, t3, t1              # dst_ptr += vl*N
    sub     t4, t4, a2              # rem -= vl
    j       2b

8:  addi    t0, t0, 1               # r++
    j       1b

9:  ret
-------------------------------------------------------------------------------------------
```

5.1.1.2 Test C code

```
-------------------------------------------------------------------------------------------

#include <opencv2/opencv.hpp>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define N 512  // image is 512x512
extern "C" void rotate90_gray_rvv(const uint8_t *src, uint8_t *dst);

// ------------------- "namespace" ns_: plain C-style prefix -------------------
// 90° clockwise: (r, c) -> (c, N-1-r)
//
static void rotate90_cw_u8_c(const uint8_t *src, uint8_t *dst) {
    for (int r = 0; r < N; ++r) {
        const uint8_t *srow = src + r * N;
        for (int c = 0; c < N; ++c) {
            dst[c * N + (N - 1 - r)] = srow[c];
        }
    }
}
int main(int argc, char **argv) {
    if (argc != 4) {
        printf("Usage: %s <grayscale_512x512.png>  <rot_image_ass.png> <rot_image_c>\n", argv[0]);
        return 1;
```

```
    }
    const char *in_path = argv[1];
    const char *out_path_ass = argv[2];
    const char *out_path_c = argv[3];
    clock_t startv, endv;
    clock_t start, end;
    double elapsed_secv;
    double elapsed_sec;
    // Load as 8-bit grayscale with OpenCV
    cv::Mat img = cv::imread(in_path, cv::IMREAD_GRAYSCALE);
    if (img.empty()) {
        printf("Error: could not open image: %s\n", in_path);
        return 1;
    }
    if (img.cols != N || img.rows != N) {
        printf("Error: expected %dx%d, got %dx%d\n", N, N, img.cols, img.rows);
        return 1;
    }
    if (img.type() != CV_8UC1) {
        printf("Error: expected 8-bit single-channel image.\n");
        return 1;
    }
    if (!img.isContinuous()) {
        img = img.clone(); // ensure tight rows for pointer math
    }

// Rotate using the C-style function
    uint8_t *dst_ass = (uint8_t*)malloc((size_t)N * N);
    uint8_t *dst_c = (uint8_t*)malloc((size_t)N * N);
    if (!dst_ass || !dst_c) {
        printf("Error: out of memory\n");
        return 1;
    }
    startv=clock();
    rotate90_gray_rvv(img.data, dst_ass);
    endv=clock();
    elapsed_secv = (double)(endv - startv) / CLOCKS_PER_SEC;
    printf("Vector execution time in sec: %.6f\n",elapsed_secv);
    start=clock();
    rotate90_cw_u8_c(img.data, dst_c);
    end=clock();
    elapsed_sec = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Scalar execution time in sec: %.6f\n",elapsed_sec);
    printf("speed-up: %.3f\n",elapsed_sec/elapsed_secv);
    // Wrap output buffer and save with OpenCV
    cv::Mat rot_ass(N, N, CV_8UC1, dst_ass);
    cv::Mat rot_c(N, N, CV_8UC1, dst_c);
    if (!cv::imwrite(out_path_ass, rot_ass)) {
        printf("Error: failed to write %s\n", out_path_ass);
        free(dst_ass);
        return 1;
    }
    if (!cv::imwrite(out_path_c, rot_c)) {
        printf("Error: failed to write %s\n", out_path_c);
        free(dst_c);
        return 1;
    }
    printf("Wrote: %s\n", out_path_ass);
    printf("Wrote: %s\n", out_path_c);
    cv::imshow("Rotatds assembler",rot_ass);
    cv::imshow("Rotated C",rot_c);
    cv::waitKey(0);
    free(dst_ass);
    free(dst_c);
    return 0;
}
-----------------------------------------------------------------------------------------
g++ -march=rv64gcv rotate90_gray_rvv.s rotate90_gray.cpp -o rotate90_gray $(pkg-config --cflags --
libs opencv4)
% ./rotate90_gray pictures/deer.512.512.grey.png r90_ass.png r90_c.png
Vector execution time in sec: 0.016417
Scalar execution time in sec: 0.048995
speed-up: 2.984
Wrote: r90_ass.png
Wrote: r90_c.png
-----------------------------------------------------------------------------------------
```

After the compilation with optimized C++ code:

```
-----------------------------------------------------------------------------------------
%  g++ -O2 -march=rv64gcv -fopt-info-vec rotate90_gray_rvv.s rotate90_gray.cpp -o rotate90_gray $
(pkg-config --cflags --libs opencv4)
/usr/include/c++/14/bits/stl_vector.h:99:4: optimized: basic block part vectorized using 16 byte
vectors
/usr/include/opencv4/opencv2/core/types.hpp:1675:7: optimized: basic block part vectorized using 8
byte vectors
% ./rotate90_gray pictures/deer.512.512.grey.png r90_ass.png r90_c.png
Vector execution time in sec: 0.016330
Scalar execution time in sec: 0.017655
speed-up: 1.081
Wrote: r90_ass.png
```

```
Wrote: r90_c.png
```

Fig 5.1 Rotated grayscale image by assembler vectorized function

5.1.2 Image rotation - RGB

With the following example the program rotates 90 degrees an RGB image.

5.1.2.1 Assembly code

```
    .text
    .align  2
    .option push
    .option arch, +v
    .globl  rotate_rgb_90ccw_rvv
# a0=src, a1=srcW, a2=srcH, a3=srcStep, a4=dst, a5=dstW, a6=dstH, a7=dstStep
rotate_rgb_90ccw_rvv:
    # Basic sanity: sizes must be positive and match rotation (dstW=srcH, dstH=srcW)
    bge     x0, a1, .Lret               # srcW <= 0
    bge     x0, a2, .Lret               # srcH <= 0
    bne     a5, a2, .Lret               # dstW != srcH  -> bail (optional)
    bne     a6, a1, .Lret               # dstH != srcW  -> bail (optional)
    beqz    a0, .Lret
    beqz    a4, .Lret
    li      a6, 3                       # a6 = 3 (byte stride between channels for vlse/vsse)
    li      t0, 0                       # y = 0
# ================= Row loop over source rows (y = 0..srcH-1) =================
.Lrow:
    bge     t0, a2, .Lret
    # row0 = src + y*srcStep
    mul     t1, t0, a3
    add     t2, a0, t1                   # t2 = row0 base
    # Destination column index for CCW: dx = dstW - 1 - y
    addi    t3, a5, -1                   # t3 = dstW - 1
    sub     t3, t3, t0                   # t3 = dx
    # Column byte offset = 3*dx
    slli    t4, t3, 1                    # 2*dx
    add     t4, t4, t3                   # t4 = 3*dx
    # processed x (source columns)
    li      t5, 0
# ------------------ Vector loop across the source row (x = 0..srcW-1) ------------------
.Lcol:
    bge     t5, a1, .Lnext_row
    # VL = min(remaining, VLMAX); we just set by remaining
    sub     t1, a1, t5                   # remaining = srcW - processed
    vsetvli t1, t1, e8, m1               # elements = remaining bytes (per channel)
    csrr    t6, vl                       # t6 = VL (number of pixels in this chunk)
    # byte offsets:
    #   src_chunk_base = row0 + 3*processed
    slli    t1, t5, 1                    # 2*processed
    add     t1, t1, t5                   # t1 = 3*processed
    add     t1, t2, t1                   # t1 = src_chunk_base
    #   dst_row_start = dst + (processed)*dstStep
    mul     t3, t5, a7
    add     t3, a4, t3                   # t3 = dst + processed*dstStep
    #   dst_chunk_base for this column dx: add 3*dx
    add     t3, t3, t4           # t3 = base of column (x-start row), at pixel column dx
    # ---------- LOAD channels from source row (stride = 3 bytes) ----------
    vlse8.v  v0, (t1), a6                # v0 <- R[x .. x+VL-1]
    addi     t1, t1, 1
    vlse8.v  v1, (t1), a6                # v1 <- G[x .. x+VL-1]
```

```
    addi     t1, t1, 1
    vlse8.v  v2, (t1), a6               # v2 <- B[x .. x+VL-1]
    # ---------- STORE channels down the destination column (stride = dstStep) ----------
    vsse8.v  v0, (t3), a7               # write R at (dy .. dy+VL-1, dx)
    addi     t1, t3, 1
    vsse8.v  v1, (t1), a7               # write G
    addi     t1, t3, 2
    vsse8.v  v2, (t1), a7               # write B
    # Advance processed x by VL
    add      t5, t5, t6
    j        .Lcol
# ------------------ Next source row ------------------
.Lnext_row:
    addi     t0, t0, 1
    j        .Lrow
.Lret:
    ret
    .option pop
```

5.1.2.2 Test C code

```
#include <opencv2/opencv.hpp>
#include <stdint.h>
#include <stdio.h>
#include <time.h>

    clock_t startv, endv;
    clock_t start, end;
    double elapsed_secv;
    double elapsed_sec;

extern "C" void rotate_rgb_90ccw_rvv(const uint8_t* src, int srcW, int srcH, int srcStep, uint8_t*
dst, int dstW, int dstH, int dstStep);

void rotate_rgb_90cw(const uint8_t* src, int srcW, int srcH, int srcStep, uint8_t* dst, int dstW,
int dstH, int dstStep)
{
    if (!src || !dst) return;
    if (dstW != srcH || dstH != srcW) {
        printf("Error: dst size must be (srcH x srcW) for 90-degree rotation.\n");
        return;
    }
    for (int y = 0; y < srcH; ++y) {
        const uint8_t* srow = src + (size_t)y * srcStep;
        for (int x = 0; x < srcW; ++x) {
            const uint8_t* spix = srow + 3 * x;
            // Mapping: (x,y) in source -> (y, dstH-1-x) in dest
            int dx = y;
            int dy = dstH - 1 - x;
            uint8_t* drow = dst + (size_t)dy * dstStep;
            uint8_t* dpix = drow + 3 * dx;
            dpix[0] = spix[0];
            dpix[1] = spix[1];
            dpix[2] = spix[2];
        }
    }
}

void rotate_rgb_90ccw(const uint8_t* src, int srcW, int srcH, int srcStep,
                      uint8_t* dst, int dstW, int dstH, int dstStep)
{
    if (!src || !dst) return;
    if (dstW != srcH || dstH != srcW) {
        printf("Error: dst size must be (srcH x srcW) for 90-degree rotation.\n");
        return;
    }
    for (int y = 0; y < srcH; ++y) {
        const uint8_t* srow = src + (size_t)y * srcStep;
        for (int x = 0; x < srcW; ++x) {
            const uint8_t* spix = srow + 3 * x;
            // Mapping: (x,y) in source -> (dstW-1-y, x) in dest
            int dx = dstW - 1 - y;
            int dy = x;
            uint8_t* drow = dst + (size_t)dy * dstStep;
            uint8_t* dpix = drow + 3 * dx;
            dpix[0] = spix[0];dpix[1] = spix[1];dpix[2] = spix[2];
        }
    }
}

int main(int argc, char** argv)
{
    if (argc < 2) {
        printf("Usage: %s input.png [output.png]\n", argv[0]);
        return 1;
    }
    const char* inPath  = argv[1];
    const char* outPath = (argc > 2) ? argv[2] : "rotated90.png";
    // Read color image
```

```
    cv::Mat src = cv::imread(inPath, cv::IMREAD_COLOR);
    if (src.empty()) {
        printf("Error: cannot read %s\n", inPath);
        return 2;
    }
    printf("Loaded %s: %dx%d\n", inPath, src.cols, src.rows);
    // Destination size = (srcH x srcW)
    cv::Mat dst(src.cols, src.rows, CV_8UC3);
    // Rotate 90° clockwise
    start=clock();
    rotate_rgb_90cw(src.data, src.cols, src.rows, (int)src.step, dst.data, dst.cols, dst.rows,
(int)dst.step);
    end=clock();
    elapsed_sec = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Scalar execution time in sec: %.6f\n",elapsed_sec);
    startv=clock();
    rotate_rgb_90ccw_rvv(src.data, src.cols, src.rows, (int)src.step, dst.data, dst.cols, dst.rows,
(int)dst.step);
    endv=clock();
    elapsed_secv = (double)(endv - startv) / CLOCKS_PER_SEC;
    printf("Vector execution time in sec: %.6f\n",elapsed_secv);
    printf("Speed-up: %.3f\n",elapsed_sec/elapsed_secv);
    // Save & show
    cv::imwrite(outPath, dst);
    printf("Wrote rotated image: %s\n", outPath);
    cv::imshow("Original", src);
    cv::imshow("Rotated 90 CW", dst);
    printf("Press any key in the image window to exit...\n");
    cv::waitKey(0);
    return 0;
}
-----------------------------------------------------------------------------------------------
% g++ -march=rv64gcv rgb_rotation90_rvv.s  rgb_rotation90.cpp -o rgb_rotation90 $(pkg-config --
cflags --libs opencv4)
% ./rgb_rotation90 pictures/deer.512.512.png rot_lena_color.png
Loaded lena_color.png: 512x512
Scalar execution time in sec: 0.127974
Vector execution time in sec: 0.030558
Speed-up: 4.188
-----------------------------------------------------------------------------------------------
```

After the compilation with optimized (vectorized) C code:

```
-----------------------------------------------------------------------------------------------
% g++ -O2 -march=rv64gcv -fopt-info-vec rgb_rotation90_rvv.s  rgb_rotation90.cpp -o rgb_rotation90 $
(pkg-config --cflags --libs opencv4)
/usr/include/opencv4/opencv2/core/types.hpp:1675:7: optimized: basic block part vectorized using 8
byte vectors
/usr/include/c++/14/bits/stl_vector.h:99:4: optimized: basic block part vectorized using 16 byte
vectors
% ./rgb_rotation90 pictures/deer.512.512.png
Loaded lena_color.png: 512x512
Scalar execution time in sec: 0.033407
Vector execution time in sec: 0.029483
Speed-up: 1.133
Wrote rotated image: rotated90.png
-----------------------------------------------------------------------------------------------
```

Fig 5.2 Rotated **RGB** image by assembly vectorized function

5.2 Image creation

5.2.1 Image creation (square)

In this example, we explore how to **create a simple grayscale image** with a **black background** and a generated white shape—a small square.

5.2.1.1 Image creation (square) – assembly with vector instructions

```
-------------------------------------------------------------------------------------------
.text
.align 2
.option push
.option arch, +v
.globl create_square_rvv

# void create_square_rvv(uint8_t* image, int center_x, int center_y, int size, uint8_t value)
# a0 = image, a1 = center_x, a2 = center_y, a3 = size, a4 = value
create_square_rvv:
    # Early return if size <= 0
    blez    a3, .Lret
    # Save registers
    addi    sp, sp, -32
    sd      s0, 0(sp)
    sd      s1, 8(sp)
    sd      s2, 16(sp)
    sd      s3, 24(sp)
    # Calculate top-left corner: start_x = center_x - size/2
    srli    t0, a3, 1                   # t0 = size / 2
    sub     t1, a1, t0                  # t1 = start_x = center_x - size/2
    sub     t2, a2, t0                  # t2 = start_y = center_y - size/2
    # Clamp start coordinates to image bounds (0 <= start_x, start_y <= 511)
    li      t3, 0                       # t3 = 0 (min bound)
    li      t4, 512                     # t4 = 512 (max bound)
    # Clamp start_x to [0, 511]
    blt     t1, t3, .Lclamp_x_min
    bge     t1, t4, .Lclamp_x_max
    j       .Lclamp_x_done
.Lclamp_x_min:
    li      t1, 0
    j       .Lclamp_x_done
.Lclamp_x_max:
    li      t1, 511
.Lclamp_x_done:
    # Clamp start_y to [0, 511]
    blt     t2, t3, .Lclamp_y_min
    bge     t2, t4, .Lclamp_y_max
    j       .Lclamp_y_done
.Lclamp_y_min:
    li      t2, 0
    j       .Lclamp_y_done
.Lclamp_y_max:
    li      t2, 511
.Lclamp_y_done:
  # Calculate end coordinates
    add     t5, t1, a3                  # t5 = end_x = start_x + size
    add     t6, t2, a3                  # t6 = e.Lret:
    ret
.option pop
nd_y = start_y + size
    # Clamp end coordinates to [0, 512]
    blt     t5, t3, .Lclamp_ex_min
    bge     t5, t4, .Lclamp_ex_max
    j       .Lclamp_ex_done
.Lclamp_ex_min:
    li      t5, 0
    j       .Lclamp_ex_done
.Lclamp_ex_max:
    li      t5, 512
.Lclamp_ex_done:
    blt     t6, t3, .Lclamp_ey_min
    bge     t6, t4, .Lclamp_ey_max
    j       .Lclamp_ey_done
.Lclamp_ey_min:
    li      t6, 0
    j       .Lclamp_ey_done
.Lclamp_ey_max:
    li      t6, 512
.Lclamp_ey_done:
    # Calculate actual width and height after clamping
    sub     s0, t5, t1                  # s0 = actual_width
    sub     s1, t6, t2                  # s1 = actual_height
    # Early return if no area to fill
    blez    s0, .Lcleanup
    blez    s1, .Lcleanup
    # Calculate image row stride (512 bytes)
    li      s2, 512                     # s2 = image width (stride)
    # Calculate starting address in image buffer
    mul     t3, t2, s2                  # t3 = start_y * stride
    add     t3, t3, t1                  # t3 += start_x
```

```
    add     s3, a0, t3                  # s3 = image + start_y*stride + start_x
    # Broadcast the fill value to a vector register
    vsetvli t4, x0, e8, m1, ta, ma      # Set SEW=8, no elements yet
    vmv.v.x v1, a4                      # v1 = broadcast(value)
    # Loop through each row
    li      t0, 0                       # t0 = row counter
.Lrow_loop:
    bge     t0, s1, .Lcleanup           # Exit when all rows processed
    # Calculate current row address
    mul     t1, t0, s2                  # t1 = row_offset = row * stride
    add     t2, s3, t1                  # t2 = current_row_address
    # Fill the row using vector instructions
    mv      t3, s0                      # t3 = remaining width
    mv      t4, t2                      # t4 = current address
.Lfill_row:
    # Set vector length for this chunk
    vsetvli t5, t3, e8, m1, ta, ma      # t5 = min(remaining, VLMAX)
    # Store vector to memory
    vse8.v  v1, (t4)
    # Advance pointers
    add     t4, t4, t5                  # Move destination pointer
    sub     t3, t3, t5                  # Decrement remaining count
    bnez    t3, .Lfill_row              # Continue if more to fill
    # Next row
    addi    t0, t0, 1                   # row_counter++
    j       .Lrow_loop
.Lcleanup:
    # Restore registers
    ld      s0, 0(sp)
    ld      s1, 8(sp)
    ld      s2, 16(sp)
    ld      s3, 24(sp)
    addi    sp, sp, 32
.Lret:
    ret
.option pop
```

5.2.1.2 Image creation (square) – C test function

The following test program integrates the external **`create_square_rvv()`** function with the local scalar function **`create_square_scalar()`**. Both functions take the same arguments and generate 512×512 pixel images, but with different placements of the small square. The resulting image buffers are stored as **`.pgm`** files.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#define WIDTH 512
#define HEIGHT 512
#include <time.h>

    clock_t startv, endv;
    clock_t start, end;
    double elapsed_secv;
    double elapsed_sec;
    int steps=100000;

// Assembly function prototype
extern void create_square_rvv(uint8_t* image, int center_x, int center_y, int size, uint8_t value);
// Function to save image as PGM
void save_pgm(const char *filename, uint8_t *image, int width, int height) {
    FILE *fp = fopen(filename, "wb");
    if (!fp) {
        printf("Error opening file %s\n", filename);
        return;
    }
    fprintf(fp, "P5\n%d %d\n255\n", width, height);
    size_t written = fwrite(image, 1, width * height, fp);
    fclose(fp);
    printf("Image saved as %s (%zu bytes written)\n", filename, written);
}
// Simple scalar version for comparison
void create_square_scalar(uint8_t *image, int center_x, int center_y, int size, uint8_t value) {
    if (size <= 0) return;
    int start_x = center_x - size/2;
    int start_y = center_y - size/2;
    // Clamp coordinates
    start_x = (start_x < 0) ? 0 : (start_x >= WIDTH) ? WIDTH-1 : start_x;
    start_y = (start_y < 0) ? 0 : (start_y >= HEIGHT) ? HEIGHT-1 : start_y;
    int end_x = start_x + size;
    int end_y = start_y + size;
    end_x = (end_x > WIDTH) ? WIDTH : end_x;
    end_y = (end_y > HEIGHT) ? HEIGHT : end_y;
    for (int y = start_y; y < end_y; y++) {
        for (int x = start_x; x < end_x; x++) {
            image[y * WIDTH + x] = value;
        }
```

```
    }
}

int main() {
    printf("Testing create_square_rvv function...\n");
    // Allocate and initialize image buffer
    uint8_t *image = (uint8_t *)calloc(WIDTH * HEIGHT, sizeof(uint8_t));
    if (!image) {
        printf("Memory allocation failed\n");
        return 1;
    }
    printf("Creating 10x10 white square at center (256,256)...\n");
    // Test with a simple case first
    startv=clock();
    for(int i=0;i<steps;i++) create_square_rvv(image, 256, 256, 10, 255);
    endv=clock();
    elapsed_secv = (double)(endv - startv) / CLOCKS_PER_SEC;
    printf("Vector execution time in sec: %.6f\n",elapsed_secv);
    // Verify the result by checking if any pixels were set
    int count = 0;
    for (int i = 0; i < WIDTH * HEIGHT; i++) {
        if (image[i] == 255) count++;
    }
    printf("Pixels set to white: %d (expected: 100)\n", count);
    // Save the result
    save_pgm("square_rvv.pgm", image, WIDTH, HEIGHT);
    // Test edge cases
    printf("Testing edge case: square at corner...\n");
    memset(image, 0, WIDTH * HEIGHT); // Clear image
    //create_square_rvv(image, 5, 5, 20, 255);
    start=clock();
    for(int i=0;i<steps;i++) create_square_scalar(image, 5, 5, 20, 255);
    end=clock();
    elapsed_sec = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Scalar execution time in sec: %.6f\n",elapsed_sec);
    printf("Speed-up: %.3f\n",elapsed_sec/elapsed_secv);
    save_pgm("square_corner.pgm", image, WIDTH, HEIGHT);
    free(image);
    printf("Test completed successfully!\n");
    return 0;
}
```

```
% gcc -march=rv64gcv create_square_rvv.s create_square.c -o create_square -lm
% ./create_square
Testing create_square_rvv function...
Creating 10x10 white square at center (256,256)...
Vector execution time in sec: 0.012225
Pixels set to white: 100 (expected: 100)
Image saved as square_rvv.pgm (262144 bytes written)
Testing edge case: square at corner...
Scalar execution time in sec: 0.639294
Speed-up: 52.294
```

After the compilation with optimization:

```
%gcc -O2 -march=rv64gcv create_square_rvv.s create_square.c -o create_square -lm
 ./create_square
Testing create_square_rvv function...
Creating 10x10 white square at center (256,256)...
Vector execution time in sec: 0.011749
Pixels set to white: 100 (expected: 100)
Image saved as square_rvv.pgm (262144 bytes written)
Testing edge case: square at corner...
Scalar execution time in sec: 0.007318
Speed-up: 0.623
Image saved as square_corner.pgm (262144 bytes written)
Test completed successfully!
```

Note that the C optimization with vectorization provides better result that manual code with vector instructions !

```
%display  square_corner.pgm
```

```
%display  square_rvv.pgm
```

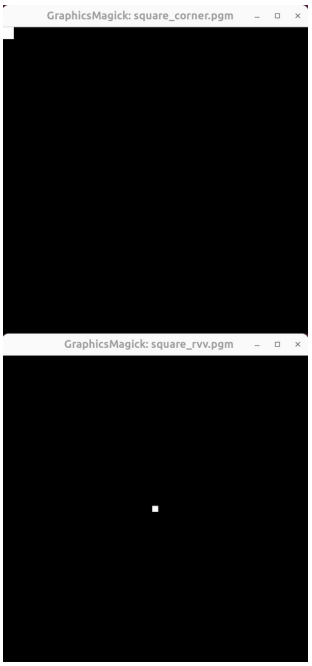

Fig 5.3 Generated grayscale (512x512) images by C code function (**`square_corner.pgm`**) and assembler vectorized function (**`square_rvv.pgm`**)

5.2.2 Image creation (line)

In this example we explore how to create a simple grayscale image with a black background and generated shape: white – line.

5.2.2.1 Image creation (line) – assembly with vector instructions

```
# ------------------------------------------------------------------------------------------
# void draw_line(uint8_t *image, int x0, int y0, int x1, int y1, uint8_t value)
# a0=image, a1=x0, a2=y0, a3=x1, a4=y1, a5=value
# Requires: RV64GCV + Zbb

    .text
    .align 2
    .globl draw_line_rvv
    .type  draw_line_rvv, @function
    .equ WIDTH,  512
    .equ HEIGHT, 512

draw_line_rvv:
    addi    sp, sp, -112
    sd      ra,   0(sp)
    sd      s0,   8(sp)
    sd      s1,  16(sp)
    sd      s2,  24(sp)
    sd      s3,  32(sp)
    sd      s4,  40(sp)
    sd      s5,  48(sp)
    sd      s6,  56(sp)
    sd      s7,  64(sp)
    sd      s8,  72(sp)
    sd      s9,  80(sp)
    sd      s10, 88(sp)
    sd      s11, 96(sp)
    mv      s0, a0          # image*
    mv      s1, a1          # x0
    mv      s2, a2          # y0
    mv      s3, a3          # x1
    mv      s4, a4          # y1
    mv      s5, a5          # value (byte)

# dx = abs(x1 - x0)
    sub     t0, s3, s1
    neg     t1, t0
    max     s6, t0, t1      # dx
# dy = abs(y1 - y0)
    sub     t0, s4, s2
    neg     t1, t0
    max     s7, t0, t1      # dy
# sx = (x0 < x1) ? 1 : -1
    li      t2, 1
    li      t3, -1
    blt     s1, s3, 1f
    mv      s8, t3
    j       2f
1:  mv      s8, t2
2:
# sy = (y0 < y1) ? 1 : -1
    blt     s2, s4, 3f
    mv      s9, t3
    j       4f
3:  mv      s9, t2
4:
# err = dx - dy
    sub     s10, s6, s7

# --- Fast paths
    beqz    s7, .HORIZ_PATH          # dy == 0
    beqz    s6, .VERT_PATH           # dx == 0
    j       .BRESENHAM

# ------------------------------------------------------------------
# Horizontal line
# ------------------------------------------------------------------
.HORIZ_PATH:
    bltz    s2, .DONE
    li      t4, HEIGHT
    bgeu    s2, t4, .DONE
    min     t5, s1, s3              # x_start
    max     t6, s1, s3              # x_end
    max     t5, t5, x0              # clamp start to 0
    li      t1, WIDTH-1             # (replaces former t7)
    minu    t6, t6, t1              # clamp end to WIDTH-1
    bgt     t5, t6, .DONE
    li      t0, WIDTH
    mul     t1, s2, t0              # y*WIDTH
    add     t1, t1, t5              # + x_start
    add     t1, s0, t1              # start ptr
    sub     t2, t6, t5
    addi    t2, t2, 1               # len
```

```
.HORIZ_LOOP:
    beqz    t2, .DONE
    vsetvli t3, t2, e8,m1,ta,ma
    vmv.v.x v1, s5
    vse8.v  v1, (t1)
    csrr    t4, vl
    add     t1, t1, t4
    sub     t2, t2, t4
    j       .HORIZ_LOOP

# ---------------------------------------------------------------------
# Vertical line
# ---------------------------------------------------------------------
.VERT_PATH:
    bltz    s1, .DONE
    li      t4, WIDTH
    bgeu    s1, t4, .DONE
    min     t5, s2, s4              # y_start
    max     t6, s2, s4              # y_end
    max     t5, t5, x0              # clamp start to 0
    li      t1, HEIGHT-1            # (replaces former t7)
    minu    t6, t6, t1              # clamp end to HEIGHT-1
    bgt     t5, t6, .DONE
    li      t0, WIDTH
    mul     t1, t5, t0
    add     t1, t1, s1
    add     t1, s0, t1              # base
    sub     t2, t6, t5
    addi    t2, t2, 1               # len
    mv      t5, t0                  # stride = WIDTH

.VERT_LOOP:
    beqz    t2, .DONE
    vsetvli t3, t2, e8,m1,ta,ma
    vmv.v.x v1, s5
    vsse8.v v1, (t1), t5
    csrr    t4, vl
    mul     t4, t4, t5              # advance = vl * stride
    add     t1, t1, t4
    sub     t2, t2, t3
    j       .VERT_LOOP
# ---------------------------------------------------------------------
# General Bresenham (scalar)
# ---------------------------------------------------------------------
.BRESENHAM:
.BRESE_LOOP:
    bltz    s1, .SKIP_PX
    bltz    s2, .SKIP_PX
    li      t0, WIDTH
    bgeu    s1, t0, .SKIP_PX
    li      t1, HEIGHT
    bgeu    s2, t1, .SKIP_PX
    mul     t2, s2, t0              # y*WIDTH
    add     t2, t2, s1              # + x
    add     t2, s0, t2              # ptr
    sb      s5, 0(t2)
.SKIP_PX:
    beq     s1, s3, 1f
    bne     s2, s4, 2f
1:  beq     s2, s4, .DONE

2:  slli    t3, s10, 1              # e2 = 2*err
    neg     t4, s7
    ble     t3, t4, 3f              # if (e2 > -dy)
    sub     s10, s10, s7
    add     s1, s1, s8
3:
    bge     t3, s6, 4f              # if (e2 < dx)
    add     s10, s10, s6
    add     s2, s2, s9
4:  j       .BRESE_LOOP
# ---------------------------------------------------------------------
.DONE:
    ld      ra,   0(sp)
    ld      s0,   8(sp)
    ld      s1,  16(sp)
    ld      s2,  24(sp)
    ld      s3,  32(sp)
    ld      s4,  40(sp)
    ld      s5,  48(sp)
    ld      s6,  56(sp)
    ld      s7,  64(sp)
    ld      s8,  72(sp)
    ld      s9,  80(sp)
    ld      s10, 88(sp)
    ld      s11, 96(sp)
    addi    sp, sp, 112
    ret
-----------------------------------------------------------------------------------------------
```

5.2.2.2 Image creation (line) – main test program in C

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define WIDTH   512
#define HEIGHT  512
clock_t startv, endv;
clock_t start, end;
double elapsed_secv;
double elapsed_sec;
int steps=100000;

// Assembly function (RV64GCV+Zbb) you already have:
extern "C" void draw_line_rvv(uint8_t *image,
                      int x0, int y0,
                      int x1, int y1,
                      uint8_t value);

/* Unbiased random integer in [0, n-1] using rejection sampling */
static unsigned rand_bounded(unsigned n) {
    unsigned limit = RAND_MAX - (RAND_MAX % n);  // largest multiple of n ≤ RAND_MAX
    unsigned r;
    do {
        r = (unsigned)rand();
    } while (r >= limit);
    return r % n;
}
// Function to draw a line between two points using Bresenham's algorithm
void draw_line(uint8_t *image, int x0, int y0, int x1, int y1, uint8_t value) {
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int sx = (x0 < x1) ? 1 : -1;
    int sy = (y0 < y1) ? 1 : -1;
    int err = dx - dy;
    while (1) {
        if (x0 >= 0 && x0 < WIDTH && y0 >= 0 && y0 < HEIGHT) {
            image[y0 * WIDTH + x0] = value;
        }
        if (x0 == x1 && y0 == y1) break;
        int e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y0 += sy;
        }
    }
}
// Write an 8-bit grayscale PGM (binary P5)
static int write_pgm(const char *path, const uint8_t *img, int w, int h)
{
    FILE *f = fopen(path, "wb");
    if (!f) {
        perror("fopen");
        return -1;
    }
    // P5 header: magic, width height, maxval
    if (fprintf(f, "P5\n%d %d\n255\n", w, h) < 0) {
        perror("fprintf");
        fclose(f);
        return -1;
    }
    size_t n = (size_t)w * (size_t)h;
    if (fwrite(img, 1, n, f) != n) {
        perror("fwrite");
        fclose(f);
        return -1;
    }
    fclose(f);
    return 0;
}

// Optional helper to clear image
static void clear_image(uint8_t *img, uint8_t value)
{
    memset(img, value, (size_t)WIDTH * (size_t)HEIGHT);
}
int main(void)
{
    // Allocate framebuffer
    uint8_t *image = (uint8_t*)malloc((size_t)WIDTH * (size_t)HEIGHT);
    if (!image) {
        fprintf(stderr, "Out of memory\n");
        return 1;
```

```
    }
    int x0=rand_bounded(512);
    int x1=rand_bounded(512);
    int y0=rand_bounded(512);
    int y1=rand_bounded(512);
    // Border box
    clear_image(image, 0);
    startv=clock();
    for(int i=0;i<steps;i++) {
    draw_line_rvv(image, x0, x1, y0, y1, 255);
    }
    endv=clock();
    elapsed_secv = (double)(endv - startv) / CLOCKS_PER_SEC;
    printf("Vector execution time in sec: %.6f\n",elapsed_secv);
    // Border box
    clear_image(image, 0);
    start=clock();
    for(int i=0;i<steps;i++) {
    draw_line(image, x0, x1, y0, y1, 255);
    }
    end=clock();
    elapsed_sec = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Scalar execution time in sec: %.6f\n",elapsed_sec);
    printf("Speed-up: %.3f\n",elapsed_sec/elapsed_secv);
    // Save as PGM
    const char *out_path = "output.pgm";
    if (write_pgm(out_path, image, WIDTH, HEIGHT) != 0) {
        fprintf(stderr, "Failed to write %s\n", out_path);
        free(image);
        return 1;
    }
    printf("Wrote %s (%dx%d)\n", out_path, WIDTH, HEIGHT);
    free(image);
    return 0;
}
```

```
% g++ -march=rv64gcvzbb draw_line_rvv.s  draw_line.c -o draw_line
% ./draw_line
Vector execution time in sec: 0.424019
Scalar execution time in sec: 1.388368
Speed-up: 3.274
Wrote output.pgm (512x512)
```

After C code optimization:

```
% g++ -O2 -march=rv64gcvzbb draw_line_rvv.s  draw_line.c -o draw_line
% ./draw_line
Vector execution time in sec: 0.421635
Scalar execution time in sec: 0.223875
Speed-up: 0.531
Wrote output.pgm (512x512)
```

Note that the C optimization with vectorization provides better result that manual code with vector instructions !

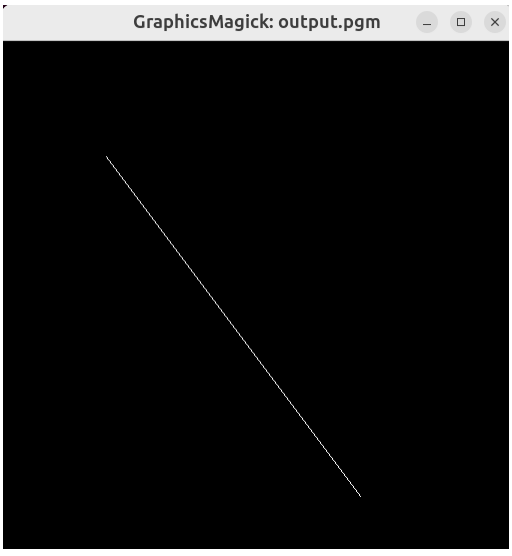

Fig 5.4 Generated grayscale (512x512) image by C code function (**`draw_line.c`**) and assembler vectorized function (**`draw_line_rvv.s`**)

To do

Analyze, compile and execute the above example program.

Note that we are using vector (**v**) and binary (**zbb**) extension instructions (**rv64gcvzbb**).

Take closer look at the following instructions:

```
    min     t5, s1, s3              # x_start
    max     t6, s1, s3              # x_end
    max     t5, t5, x0              # clamp start to 0
```

5.3 Image animation with `openGL` and `rvv` instructions

In these examples, we make use of **OpenGL** functionalities. With this library, image frames are **rendered directly into a byte buffer** for display. This approach enables the creation of **animation examples** accelerated with vector instructions.

5.3.1 Image Animation – Colors

In the first example, we generate random color values (applied uniformly to all pixels) and render them into the image buffer.

5.3.1.1 Image animation – colors: vector assembly: `fill_color_rvv`

```
# -march=rv32gcv -mabi=ilp32     OR
# -march=rv64gcv -mabi=lp64
# void fill_color_rvv(uint8_t *buffer, uint8_t r, uint8_t g, uint8_t b)

    .text
    .align  2
    .globl  fill_color_rvv
    .type   fill_color_rvv, @function

    .equ    WIDTH,    640
    .equ    HEIGHT,   480
    .equ    ROWBYTES, WIDTH*3        # 1920 bytes per row

fill_color_rvv:
    # a0 = buffer, a1 = r, a2 = g, a3 = b
    mv      t0, a0                   # t0 = base pointer to current row
    li      t5, HEIGHT               # t5 = rows remaining
    beqz    t5, .Ldone
    li      t6, 3                    # t6 = stride (3 bytes between consecutive elements)

.Lrow:
    mv      t2, t0                   # t2 = R start (row + 0)
    addi    t3, t0, 1                # t3 = G start (row + 1)
    addi    t4, t0, 2                # t4 = B start (row + 2)
    li      a4, WIDTH                # a4 = pixels remaining in this row

.Lxloop:
    vsetvli t1, a4, e8, m1, ta, ma   # t1 = VL (number of pixels this chunk)
    # Broadcast (r,g,b) across VL elements
    vmv.v.x v0, a1                   # v0 := r
    vmv.v.x v1, a2                   # v1 := g
    vmv.v.x v2, a3                   # v2 := b
    # Strided vector stores: write RGB planes with stride = 3 bytes
    vsse8.v v0, 0(t2), t6            # row[0], row[3], row[6], ...
    vsse8.v v1, 0(t3), t6            # row[1], row[4], row[7], ...
    vsse8.v v2, 0(t4), t6            # row[2], row[5], row[8], ...
    # Decrement pixels remaining in this row
    sub     a4, a4, t1
    # Compute increment = VL * 3 without using t7
    slli    a5, t1, 1                # a5 = VL * 2
    add     a5, a5, t1               # a5 = VL * 3
    # Advance channel pointers by increment
    add     t2, t2, a5
    add     t3, t3, a5
    add     t4, t4, a5
    bnez    a4, .Lxloop              # more pixels in this row?
    # Next row
    addi    t5, t5, -1
    addi    t0, t0, ROWBYTES
    bnez    t5, .Lrow

.Ldone:
    ret
    .size   fill_color_rvv, .-fill_color_rvv
```

5.3.1.2 Image animation – colors C test program/function

```
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define WIDTH  640
#define HEIGHT 480

extern "C" void fill_color_rvv(uint8_t *buffer, uint8_t r, uint8_t g, uint8_t b);

static GLuint tex = 0;
static uint8_t *frame = NULL;
static int color_index = 0; // 0 = red, 1 = green, 2 = blue
```

```
static int color_red = 0;
static int color_green = 0;
static int color_blue = 0;

static void fill_color(uint8_t *buffer , uint8_t r, uint8_t g, uint8_t b)
{
    for (int y = 0; y < HEIGHT; ++y) {
        uint8_t *row = buffer + (size_t)y * WIDTH * 3;
        for (int x = 0; x < WIDTH; ++x) {
            row[3*x + 0] = r;
            row[3*x + 1] = g;
            row[3*x + 2] = b;
        }
    }
}

static void init_texture(void)
{
    frame = (unsigned char*)malloc((size_t)WIDTH * HEIGHT * 3);
    if (!frame) { fprintf(stderr, "OOM\n"); exit(1); }
    glGenTextures(1, &tex);
    glBindTexture(GL_TEXTURE_2D, tex);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, WIDTH, HEIGHT, 0,
                 GL_RGB, GL_UNSIGNED_BYTE, NULL);
}

static void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)WIDTH, 0.0, (GLdouble)HEIGHT);
    glMatrixMode(GL_MODELVIEW); glLoadIdentity();
}

static void display(void)
{
    clock_t startv, endv;
    clock_t start, end;
    double elapsed_secv;
    double elapsed_sec;
    int step=10;
    startv=clock();
    for(int i=0;i<step;i++) fill_color_rvv(frame,color_red, color_green, color_blue);
    endv=clock();
    elapsed_secv = (double)(endv - startv) / CLOCKS_PER_SEC;
    printf("Vector execution time in sec: %.6f\n",elapsed_secv);
    start=clock();
    for(int i=0;i<step;i++) fill_color(frame,color_red, color_green, color_blue);
    end=clock();
    elapsed_sec = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Scalar execution time in sec: %.6f\n",elapsed_sec);
    printf("Speed-up: %.3f\n",elapsed_sec/elapsed_secv);
    // Upload frame
    glBindTexture(GL_TEXTURE_2D, tex);
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, WIDTH, HEIGHT,
                    GL_RGB, GL_UNSIGNED_BYTE, frame);
    // Draw
    glClear(GL_COLOR_BUFFER_BIT);
    glEnable(GL_TEXTURE_2D);
    glBegin(GL_QUADS);
        glTexCoord2f(0.f, 0.f); glVertex2f(0.f,      0.f);
        glTexCoord2f(1.f, 0.f); glVertex2f(WIDTH,    0.f);
        glTexCoord2f(1.f, 1.f); glVertex2f(WIDTH,    HEIGHT);
        glTexCoord2f(0.f, 1.f); glVertex2f(0.f,      HEIGHT);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
}

static void timer(int value)
{
    (void)value;
    // Move to next color
    //color_index = (color_index + 1) % 3;
    color_red = rand()%256;   // (color_red + 1) % 256;
    color_green = rand()%256;   //(color_green + 1) % 256;
    color_blue = rand()%256;   //(color_blue + 1) % 256;
    glutPostRedisplay();
    glutTimerFunc(100, timer, 0); // change color every 1 second
}

int main(int argc, char **argv)
{
    srand(time(NULL));
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

```
    glutInitWindowSize(WIDTH, HEIGHT);
    glutCreateWindow("Simple RGB Animation (640x480)");
    glClearColor(0.f, 0.f, 0.f, 1.f);
    reshape(WIDTH, HEIGHT);
    init_texture();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutTimerFunc(0, timer, 0);
    glutMainLoop();
    free(frame);
    return 0;
}
-------------------------------------------------------------------------------------------

% g++ -march=rv64gcv fill_color.c fill_color_rvv.s -o fill_color -lGLU -lglut -lGL
% ./fill_color
MESA: error: ZINK: vkEnumeratePhysicalDevices failed (VK_ERROR_INITIALIZATION_FAILED)
MESA: error: ZINK: failed to choose pdev
glx: failed to create drisw screen
Vector execution time in sec: 0.012705
Scalar execution time in sec: 0.090637
Speed-up: 7.134
Vector execution time in sec: 0.010012
Scalar execution time in sec: 0.090785
Speed-up: 9.068
-------------------------------------------------------------------------------------------
```

After the compilation with optimization:

```
-------------------------------------------------------------------------------------------
%  g++ -O2 -march=rv64gcv -fopt-info-vec fill_color.c fill_color_rvv.s -o fill_color -lGLU -lglut -
lGL
fill_color.c:24:27: optimized: loop vectorized using variable length vectors
% ./fill_color
WARNING: Some incorrect rendering might occur because the selected Vulkan device (PowerVR B-Series
BXE-2-32 MC1) doesn't support base Zink requirements: feats.features.fillModeNonSolid
DRI3 not available
Vector execution time in sec: 0.013101
Scalar execution time in sec: 0.005844
Speed-up: 0.446
Vector execution time in sec: 0.012548
Scalar execution time in sec: 0.005848
Speed-up: 0.466
Vector execution time in sec: 0.010299
Scalar execution time in sec: 0.005862
Speed-up: 0.569

-------------------------------------------------------------------------------------------
```

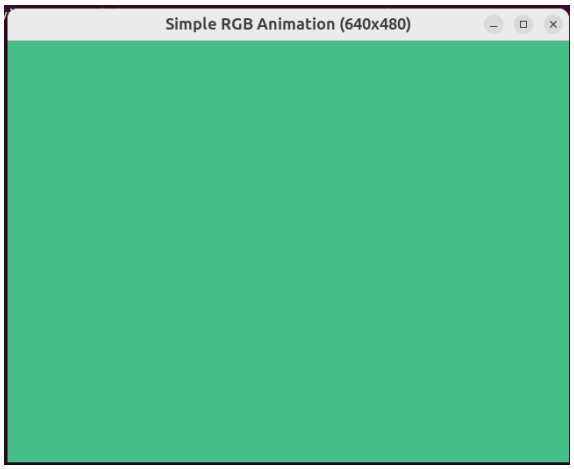

Fig 5.5 Generated color (**640x480**) **frame** displayed by **openGL** function.

5.3.2 Image animation – rotate 90 degrees

In the following example we take (load) an image (`.png`) using **openCV** image read function.

Than we use the byte buffer as an input to rotate 90 degrees function. These function is called by animation timer in order to produce new (rotated) image and project it on the video memory frame.

```
-------------------------------------------------------------------------------------------

# rotate90_rvv.s
# a0 = src (u8*)
# a1 = dst (u8*)
# a2 = N   (size_t, width==height)
#
# dst[(c*N + (N-1-r))*3 + k] = src[((r*N + c)*3) + k], k=0..2
```

```
    .text
    .globl  rotate90_rvv
    .align  2
rotate90_rvv:
    beqz    a2, .Lret               # if (N==0) return
    li      t5, 3                   # bytes per pixel
    mul     a3, a2, t5              # a3 = row_stride = N*3 (bytes)
    mv      a4, a3                  # a4 = dst column stride = N*3
    li      t0, 0                   # r = 0
.Lrow:
    beq     t0, a2, .Lret           # for (r = 0; r < N; ++r)
    # srow = src + r*row_stride
    mul     t2, t0, a3
    add     t2, a0, t2              # t2 = srow
    # dptr = dst + (N-1-r)*3
    addi    t3, a2, -1              # N-1
    sub     t3, t3, t0              # N-1-r
    mul     t3, t3, t5              # *3
    add     t3, a1, t3              # t3 = dptr (base for c=0)
    mv      t6, a2                  # rem = N columns left
.Lcol:
    beqz    t6, .Lnext_row
    # Set vl for this chunk (SEW=8, LMUL=1)
    vsetvli t1, t6, e8, m1, ta, ma   # t1 = vl
    # Load vl RGB triplets from source row (unit stride)
    # -> v0=R, v1=G, v2=B
    vlseg3e8.v  v0, (t2)
    # Store to rotated column with stride = N*3 bytes
    vssseg3e8.v v0, (t3), a4
    # srow += 3*vl  (t4 = 2*vl + vl)
    slli    t4, t1, 1               # t4 = 2*vl
    add     t4, t4, t1              # t4 = 3*vl
    add     t2, t2, t4
    # dptr += (N*3)*vl
    mul     t4, t1, a4              # t4 = vl * (N*3)
    add     t3, t3, t4
    sub     t6, t6, t1              # rem -= vl
    j       .Lcol

.Lnext_row:
    addi    t0, t0, 1               # r++
    j       .Lrow

.Lret:
    ret
```

```
// cv_opengl_drawpixels_rotate_512.cpp
// Keep one CPU buffer (RGB, 8UC3, 512x512). Every 2s, rotate it 90° CW
// and display via glDrawPixels.
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <vector>
#include <opencv2/opencv.hpp>
#if defined(__APPLE__)
  #include <OpenGL/gl.h>
  #include <GLUT/glut.h>
#else
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
#endif
static constexpr int W = 512;
static constexpr int H = 512;
static std::vector<uint8_t> gBuf;   // current image buffer (W*H*3, RGB)
static std::vector<uint8_t> gTmp;   // scratch for rotation
static int gWinW = W, gWinH = H;
static unsigned long gTick = 0;

clock_t startv, endv, start, end;
double elapsed_secv, elapsed_sec;
int steps=10;

extern "C"  void rotate90_rvv(const uint8_t* src, uint8_t* dst, size_t N);

// Inject: copy an RGB Mat (512x512, 8UC3) into gBuf
static void injectImageToBuffer(const cv::Mat& rgb) {
    if (rgb.cols != W || rgb.rows != H || rgb.type() != CV_8UC3) return;
    const size_t bytes = (size_t)W * H * 3;
    if (!rgb.isContinuous()) {
        cv::Mat tmp = rgb.clone();
        memcpy(gBuf.data(), tmp.data, bytes);
    } else {
        memcpy(gBuf.data(), rgb.data, bytes);
    }
}

// Rotate 90° CW: (r,c) -> (c, N-1-r). Operates on RGB interleaved.
static void rotate90_cw_rgb(const uint8_t* src, uint8_t* dst) {
```

```
    const int N = W; // 512
    for (int r = 0; r < N; ++r) {
        for (int c = 0; c < N; ++c) {
            const int si = (r * N + c) * 3;
            const int di = (c * N + (N - 1 - r)) * 3;
            dst[di + 0] = src[si + 0]; dst[di + 1] = src[si + 1];
            dst[di + 2] = src[si + 2];
        }
    }
}

static void displayCB() {
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);  glLoadIdentity();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    // Draw with origin upper-left: flip Y via negative PixelZoom
    glRasterPos2f(-1.f,  1.f);
    glPixelZoom((float)gWinW / W, -(float)gWinH / H);
    glDrawPixels(W, H, GL_RGB, GL_UNSIGNED_BYTE, gBuf.data());
    glPixelZoom(1.f, 1.f);
    glutSwapBuffers();
}

static void reshapeCB(int w, int h) {
    gWinW = (w > 1) ? w : 1;
    gWinH = (h > 1) ? h : 1;
    glViewport(0, 0, gWinW, gWinH);
}

static void keyCB(unsigned char key, int, int) {
    if (key == 27 || key == 'q') { // ESC / q
#ifdef FREEGLUT
        //glutLeaveMainLoop();
#endif
        exit(0);
    }
    if (key == 'r') { // rotate immediately
        rotate90_cw_rgb(gBuf.data(), gTmp.data());
        gBuf.swap(gTmp);
        glutPostRedisplay();
    }
    if (key == 'o') { // reload original file on demand (no path stored here)
        // noop placeholder; reload logic can be added if you store the path
    }
}

// Every 2000 ms: rotate current buffer 90° CW, update title, redraw
static void timerCB(int) {
    ++gTick;
    start=clock();
    for(int i=0;i<steps;i++) rotate90_cw_rgb(gBuf.data(), gTmp.data());
    end=clock();
    elapsed_sec = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Scalar execution time in sec: %.6f\n",elapsed_sec);
    startv=clock();
    for(int i=0;i<steps;i++) rotate90_cw_rgb_ass(gBuf.data(), gTmp.data(), 512);
    endv=clock();
    elapsed_secv = (double)(endv - startv) / CLOCKS_PER_SEC;
    printf("Vector execution time in sec: %.6f\n",elapsed_secv);
    printf("Speed-up: %.3f\n",elapsed_sec/elapsed_secv);
    gBuf.swap(gTmp);
    char title[160];
    snprintf(title, sizeof(title), "Pixels 512x512 — rotate 90 CW");
    glutSetWindowTitle(title);
    glutPostRedisplay();
    glutTimerFunc(200, timerCB, 0); // re-arm for 2s
}

static bool loadAndPrep(const char* path) {
    cv::Mat src = cv::imread(path, cv::IMREAD_UNCHANGED);
    if (src.empty()) {
        printf("Error: cannot open %s\n", path);
        return false;
    }
    if (src.cols != W || src.rows != H) {
        printf("Note: input is %dx%d, resizing to %dx%d.\n", src.cols, src.rows, W, H);
        cv::resize(src, src, cv::Size(W, H), 0, 0, cv::INTER_AREA);
    }
    cv::Mat rgb;
    if (src.channels() == 1)       cv::cvtColor(src, rgb, cv::COLOR_GRAY2RGB);
    else if (src.channels() == 3)  cv::cvtColor(src, rgb, cv::COLOR_BGR2RGB);
    else if (src.channels() == 4)  cv::cvtColor(src, rgb, cv::COLOR_BGRA2RGB);
    else {
        printf("Unsupported channel count: %d\n", src.channels());
        return false;
    }
    injectImageToBuffer(rgb);
    return true;
}
```

```
int main(int argc, char** argv) {
    if (argc != 2) {
        printf("Usage: %s <image_512x512.(png|jpg|...)>\n", argv[0]);
        return 1;
    }
    gBuf.resize((size_t)W * H * 3);
    gTmp.resize((size_t)W * H * 3);
    if (!loadAndPrep(argv[1])) return 1;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(W, H);
    glutCreateWindow("Pixels 512x512 — rotate 90cw degrees per tick");
    glClearColor(0.f, 0.f, 0.f, 1.f);
    glutDisplayFunc(displayCB);
    glutReshapeFunc(reshapeCB);
    glutKeyboardFunc(keyCB);
    glutTimerFunc(2000, timerCB, 0); // start periodic rotation
    glutMainLoop();
    return 0;
}
```

```
% g++ -march=rv64gcv cvgl_ro90.cpp rotate90_rvv.s -o cvgl_rot90 -lGLU -lglut -lGL $(pkg-config --
cflags --libs opencv4)
% ./cvgl_rot90  lena_color.png
Scalar execution time in sec: 0.123390
Vector execution time in sec: 0.023632
Speed-up: 5.221
Scalar execution time in sec: 0.123920
Vector execution time in sec: 0.023594
Speed-up: 5.252
Scalar execution time in sec: 0.123773
Vector execution time in sec: 0.023662
```

After the optimzation:

```
% g++ -O2 -march=rv64gcv -fopt-info-vec cvgl_ro90.cpp rotate90_rvv.s -o cvgl_rot90 -lGLU -lglut -lGL
$(pkg-config --cflags --libs opencv4)
/usr/include/c++/14/bits/stl_vector.h:114:13: optimized: basic block part vectorized using 16 byte
vectors
/usr/include/opencv4/opencv2/core/types.hpp:1679:7: optimized: basic block part vectorized using 8
byte vectors
/usr/include/c++/14/bits/stl_vector.h:99:4: optimized: basic block part vectorized using 16 byte
vectors

% ./cvgl_rot90 pictures/deer.512.512.png
WARNING: Some incorrect rendering might occur because the selected Vulkan device (PowerVR B-Series
BXE-2-32 MC1) doesn't support base Zink requirements: feats.features.fillModeNonSolid
DRI3 not available
Scalar execution time in sec: 0.028909
Vector execution time in sec: 0.024135
Speed-up: 1.198
Scalar execution time in sec: 0.033194
Vector execution time in sec: 0.025592
Speed-up: 1.297
```

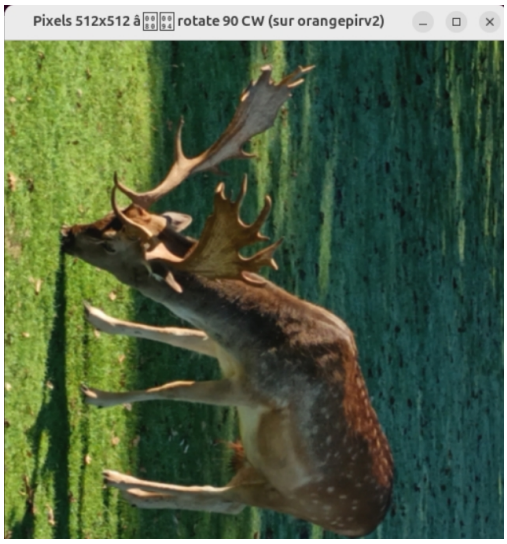

Fig 5.6 Loaded color (**512x512**) image and **rotated continuously** 90 degrees