CSCE 313: Embedded Systems

Final Project

Instructor: Jason D. Bakos

UNIVERSITY OF SOUTH CAROLINA

Final Project

- Next (final) class on 4/29 will meet in 3D22

- Final exam on 5/6 @4:00 to 6:30pm
  - Covers:
    - Memory-mapped I/O
    - Bare metal programming concepts
    - Floating-point vs fixed point
    - Performance analysis
    - Image transformations and Mandelbrot
    - Consecutive vs X-Y frame addressing

- Final project due on May 6 @11:59 p.m.
  - Objective:
    - Begin with Lab 5
    - Convert all floating-point computations to fixed-point
    - Compare CPI and instruction count of innermost loop body with Lab 6

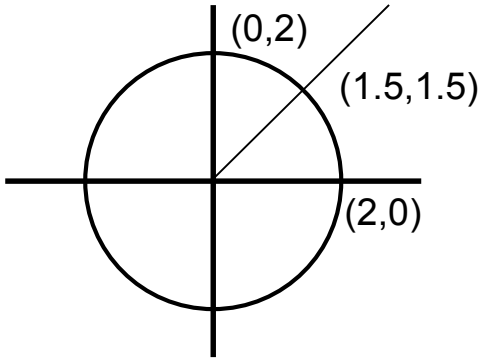
Fixed-Point

- Use a fixed-point representation for:
  - z ("x" and "y")
  - c ("x0" and "y0")
  - min_x, max_x, min_y, max_y
  - x^2+y^2 (for checking for divergence)

Fixed-Point Review

- Recall: fixed-point has fixed range
  - Recall: Range of non-fixed-point n-bit integer:
    - $-2^{n-1} <= val <= 2^{n-1}-1$
  - Range of signed (n,m) value:
    - $-2^{n-m-1} <= val <= 2^{n-m-1} - 2^{-m}$
  - Need to decide where to set decimal point
    - For c, [min|max]_[x|y]:
      - Need to represent values from -2 to 2
      - Use (32,29) representation for [-4,4) (to include +2)
    - z should cover the worst case for a diverged pixel
    - x^2+y^2 should cover the worst case for the r^2 of a diverged pixel

Worst Case Analysis (Assuming All Signed)



z	c	P(z) $(z_x^2-z_y^2+c_x, 2z_xz_y+c_y)$	LHS bits	$Z_x^2+X_y^2$	LHS bits
(0,2)	(0,2)	(-4,2)	4	20	7
(0,2)	(2,0)	(-2,0)	3	4	4
(2,0)	(0,2)	(4,2)	4	20	6
(2,0)	(2,0)	(6,0)	4	36	7
(1.5,1.5)	(1.5,1.5)	(1.5,6)	4	38.25	7

Range and Precision

- For r^2, need 7 bits the left on LHS: (32,25)

- Problem:
  - Multiply (32,n) val with (32,m) val => (64,n+m) val
  - (32,25) x (32,25) = (64,50)
  - Using only the lower 32 bits would cause the loss of all LHS bits and the 18 MS RHS bits

- Need a way to preserve 64-bit product

Example Fixed Point Multiply

- Multiply A = (32,n) val and B = (32,m) val, need C = (32,o) product:
  - Declare A and B as "alt_32"
  - Declare C as "alt_64"
  - Cast A and B as "alt_64",
  - C = (alt_64)A * (alt_64)B;
  - Convert C from (64,n+m) to (32,o):
    - Shift C (n+m)-o bits to the right
  - Return C as alt_32