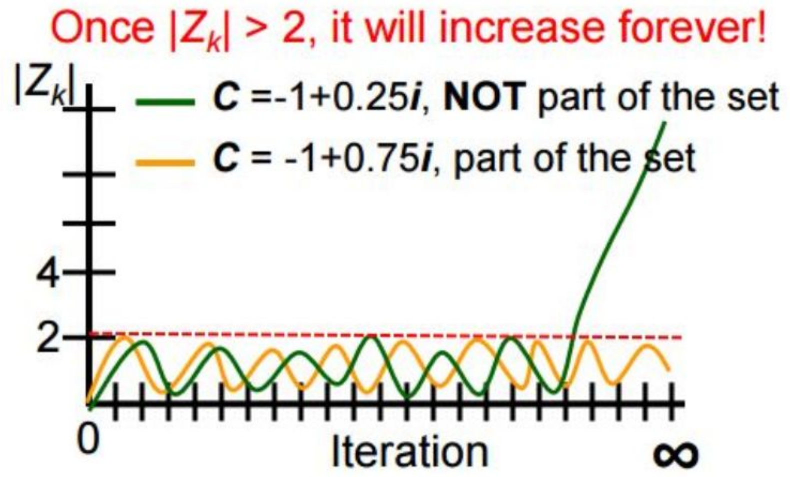
Assignment 3
Mandelbulb Set

Parallel Programming
2025/10/17

Mandelbrot Set

A set of complex numbers ℂ

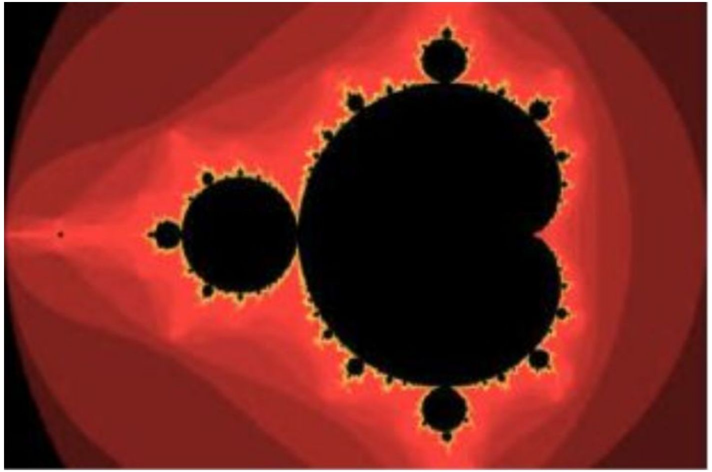
- for every complex number $c \in \mathbb{C}$, under iterations of quadratic map $Z_{k+1} = (Z_k)^2 + c$ remain bounded
    - $Z_0 = c$
    - $Z_{k+1} = (Z_k)^2 + c$
    - $|Z_k| \leq 2$



- if $|Z_k| \leq 2$ for any k, c belongs to the Mandelbrot Set

Mandelbrot Set Visualization

- Convert each pixel to the corresponding coordinates on the complex plane
- Plug into the equation repeatedly until $|Z_k| > 2$
- Color the pixel according to the iteration count
- https://www.youtube.com/watch?v=IrYfMfUURYM



Mandelbulb

- 3D fractal using spherical coordinates (Quaternion, 四元數).
- In this assignment, we refer to power-8 mandelbulb
- https://youtu.be/BLmAV6O_ea0

$$v_{k+1} = v_k^8 + C$$

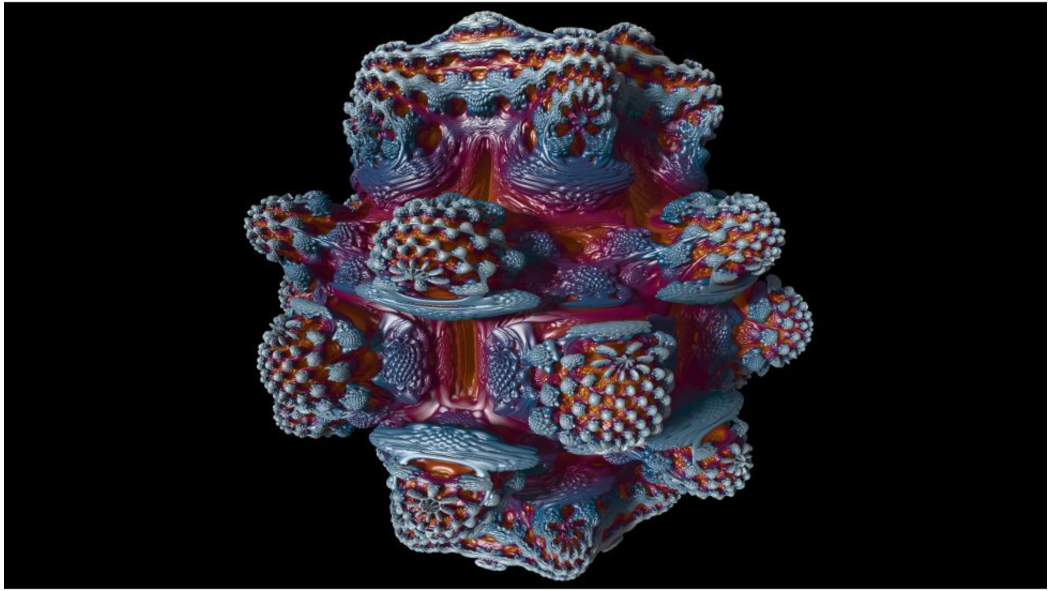
$$v = \langle x, y, z \rangle \text{ in } \mathbb{R}^3,\ v^n := r^n \langle \cos(n\theta)\cos(n\phi), \cos(n\phi)\sin(n\theta), -\sin(\phi) \rangle$$

- $r = \sqrt{x^2 + y^2 + z^2},\ \theta = \arctan\left(\frac{y}{x}\right),\ \phi = \arctan\left(\frac{z}{r}\right)$

$$x = r\sin(\phi)\cos(\theta), y = r\sin(\phi)\sin(\theta), z = r\cos(\phi)$$

Mandelbulb Visualization

- Generate 3D images by ray tracing
- We use ray marching algorithm



Ray Tracing

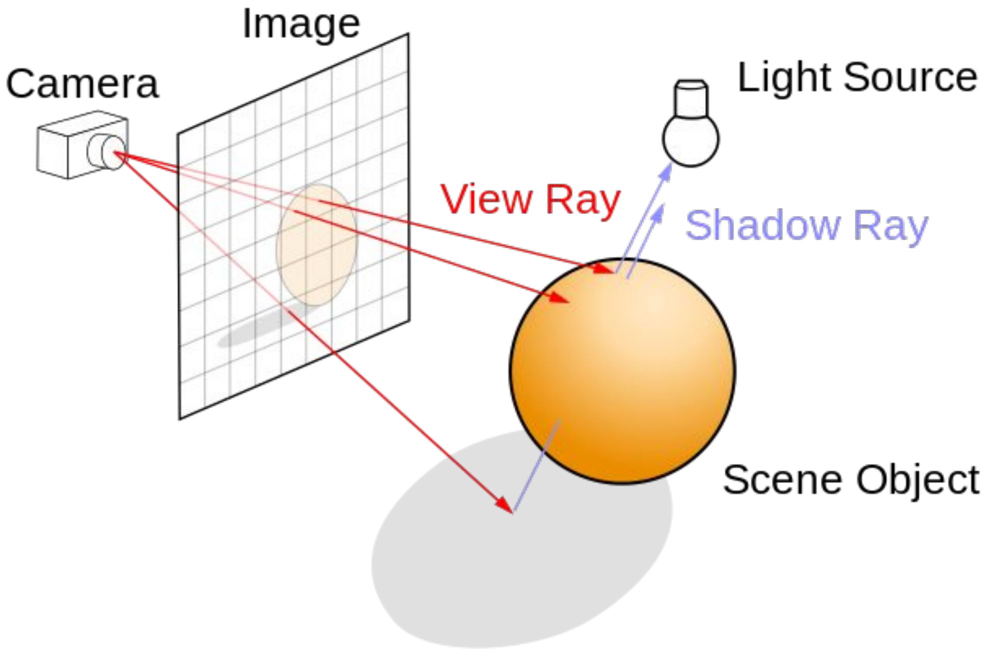
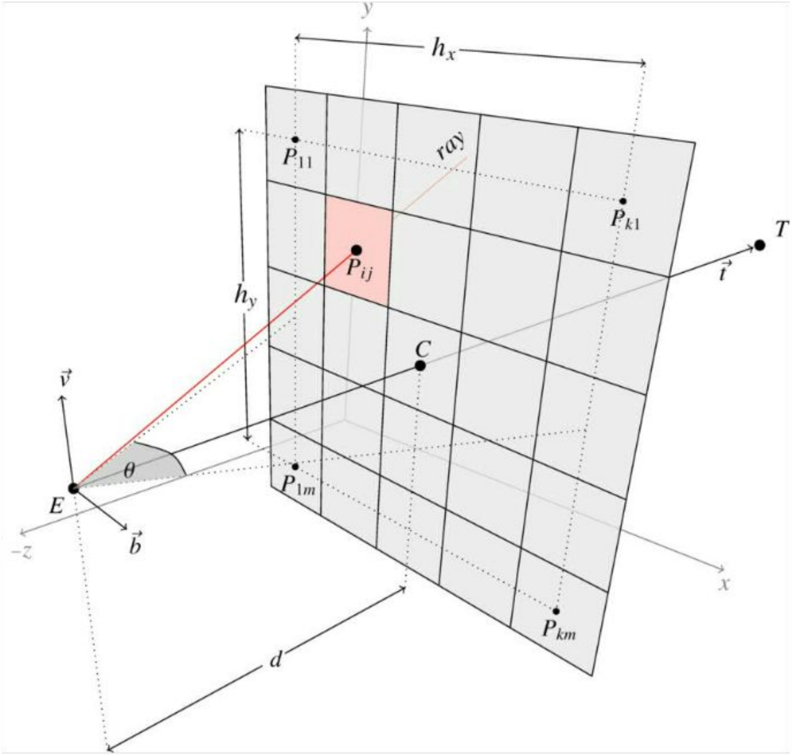


Image
Camera
Light Source
View Ray
Shadow Ray
Scene Object

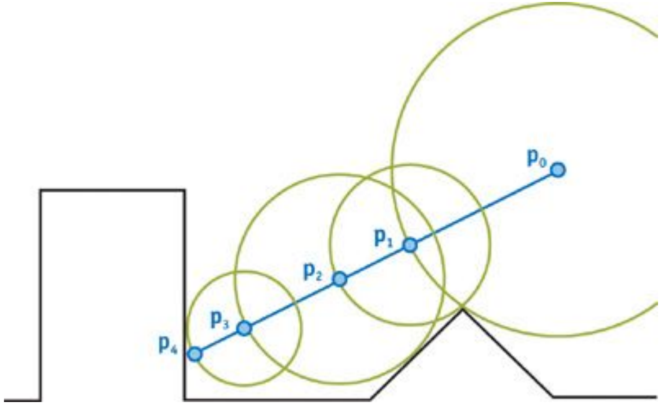


y
h_x
ray
P_{11}
P_{k1}
P_{ij}
T
$\vec{t}$
h_y
C
$\vec{v}$
θ
E
$\vec{b}$
P_{1m}
-z
x
P_{km}
d

Ray Marching

[Often used for 3D fractal rendering]()

1. Start at the “beginning” of the ray
2. Evaluate the distance function to estimate how close is to the object
3. Keep moving forward, the step should be short enough to not tunnel through the surface



Distance Function for Ray Marching

The approximate distance function of the mandelbulb is:

$$DE = \frac{0.5r \ln (r)}{dr}$$

Where $r = |v_k|$ and $dr = |v_k'|$.

We can get dr by scalar derivative $dr_{k+1} = n|v_k|^{n-1}dr_k + 1$ and $dr_0 = 1$

Goal

- We provide a sequential version of sample code named `hw3_cpu.cpp`
- You are asked to accelerate it with GPU
- Learn how to write a cuda program
- Understand the importance of Load Balancing

Input

```
./executable $x1 $y1 $z1 $x2 $y2 $z2 $width $height $filename
```

- `$x1` `double` camera position x
- `$y1` `double` camera position y
- `$z1` `double` camera position z
- `$x2` `double` camera target position x
- `$y2` `double` camera target position y
- `$z2` `double` camera target position z
- `$width` `unsigned int` width of the image
- `$height` `unsigned int` height of the image
- `$filename` `string` filename of the output PNG image

Output

- Save the result to `$filename`
- The output image should be a 32bit PNG image with RGBA channels

Resources

- `/work/b10502010/pp25/hw3/`
  - `hw3_cpu.cpp            # sequential version`
  - `Makefile`
  - `glm/                   # vector arithmetic`
  - `lodepng/               # png i/o`
  - `testcases/`

Execute

- Check `testcases/xx.txt`
- `00.txt:`
  - `pos       = -0.522 2.874 1.340`
  - `tarpos    = 0 0 0`
  - `width     = 64`
  - `height    = 64`
  - `timelimit = 5`

```
srun -N 1 -n 1 --gpus-per-node 1 -A ACD114118 -t 3 \
     ./hw3_cpu -0.522 2.874 1.340 0 0 0 64 64 00.png
```

- It may take a few hours to run large cases using sequential code. Please start with small cases first and remember to set a time limit.
- Remember to terminate your process when finished to avoid wasting resources