Chapter 2

Exterior colouring

$M$is defined in a binary way: each point is either in the set or not in the set. Points not in the set are in the complement of the set, also called the exterior. The exterior can be coloured to expose different properties of M, and achieve a variety of aesthetic effects.

The simplest colouring of the exterior is binary escape: mark all points that escape to infinity in the same colour, and those that don't in a different colour. A natural extension is to see how quickly the iterates escape, with discrete dwell colouring based on the iteration count. Further, the magnitude of the iterate after it escapes can be used to give a continuous dwell colouring which replaces bands of colour with smooth shading.

The iterates are complex numbers, with magnitude and phase. The colourings already mentioned take into account only the magnitude. The phase of the first escaping iterate changes in a direction perpendicular to its magnitude. Squaring a complex number doubles its phase, and phases are modulo a full turn. The sign of the phase gives a binary decomposition showing this angle doubling with two colours. The binary tree structure has neighbouring cells with the same colour, using a ternary decomposition with three colours makes all edges have distinct neighbour cell colours.

Using both magnitude and phase gives local coordinates within each cell of the exterior. These can be used to tile the exterior with textures, whether procedural grids or rasterized images. Procedural techniques can be used to manipulate rasterized images, altering shapes and

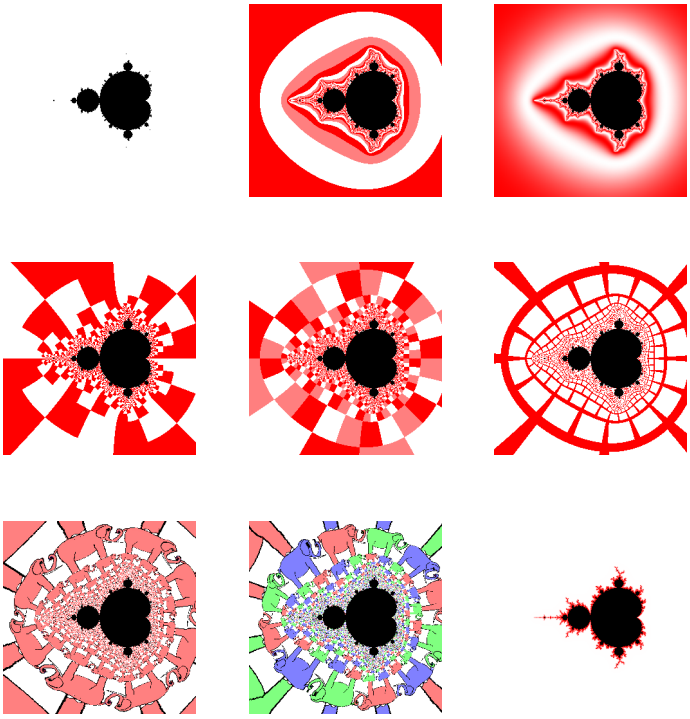

Figure 2.0.1: Comparison of exterior colourings.

colours.

The dwell bands get closer and closer together the closer you get to M. Using the derivative of the iterates, one can get a distance estimate. Mhas fine filaments, which slip between the gaps between sampled points. Distance estimation can colour pixels near M, revealing these intricate details.

Keeping track of the iteration counts of the successive minima of the magnitudes of the iterates, known as partials, can reveal the periods of nearby periodic components in M.

All of these techniques can be combined, creating a combinatorial explosion of variety.

2.1 Binary escape

Figure 2.1.1: Binary escape images.

FIXME text

Algorithm 2.1.1 Binary escape calculation.

```
ℤ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) { return 1; }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.1.2 Binary escape colouring.

```
pixel_t colour(ℤ r) {
  if (r < 0) { return pixel_black; }
  return pixel_white;
}
```

2.2 Discrete dwell

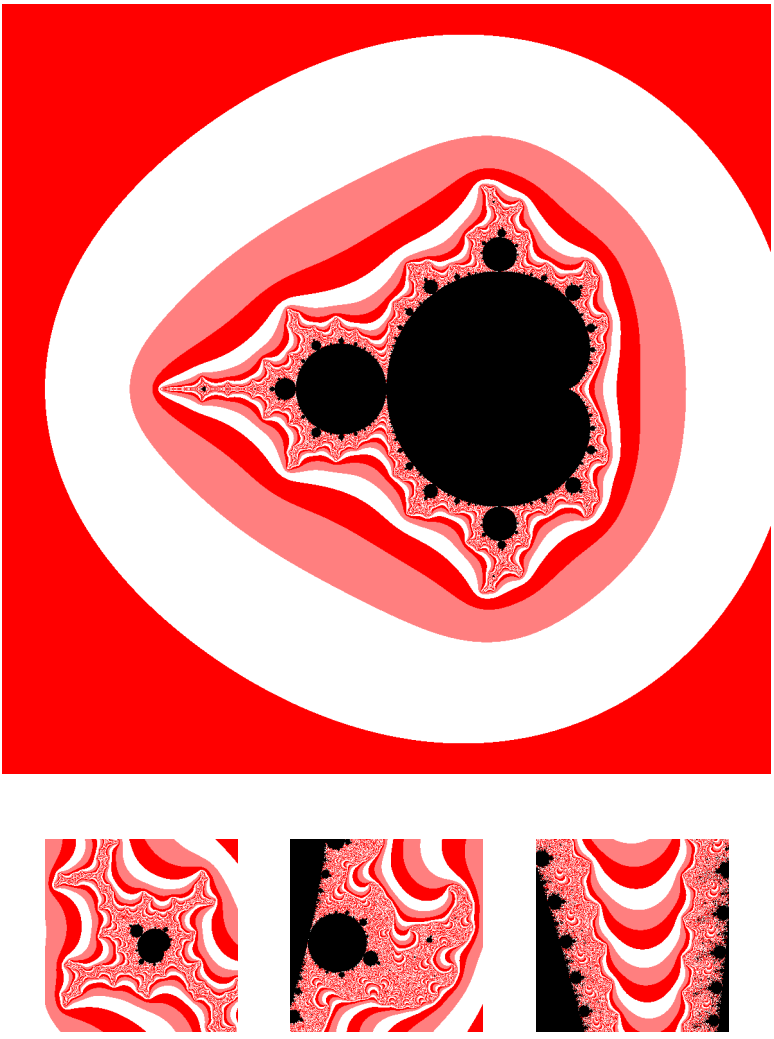

Figure 2.2.1: Discrete dwell images.

FIXME text

Algorithm 2.2.1 Discrete dwell calculation.

```
ℤ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) { return i; }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.2.2 Discrete dwell colouring.

```
pixel_t colour(ℤ r) {
  if (r < 0) { return pixel_black; }
  r ← r % 3;
  return pixel_mix
    ( pixel_white, pixel_red
    , r / 2.0
    );
}
```

2.3 Continuous dwell

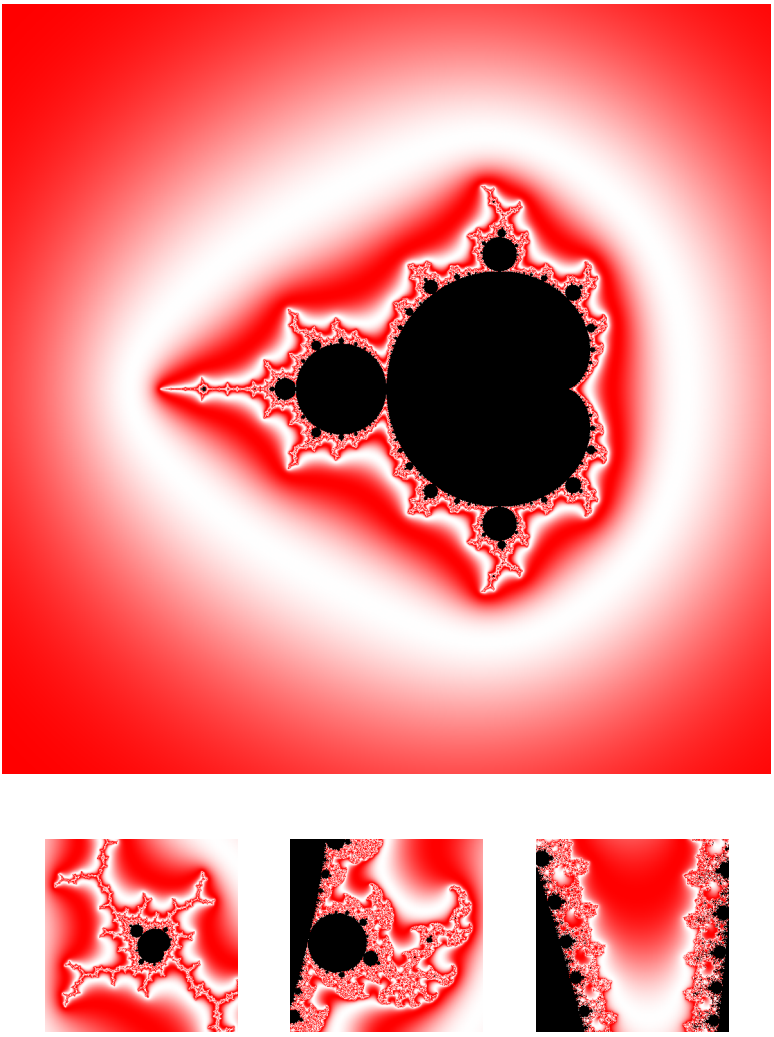

Figure 2.3.1: Continuous dwell images.

FIXME text

Algorithm 2.3.1 Continuous dwell calculation.

```
ℝ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) {
      return 1 + i - log2(log(cabs(z))/log(e));
    }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.3.2 Continuous dwell colouring.

```
pixel_t colour
( ℤ n  // maximum iteration count
, ℝ r  // continuous dwell
) {
  if (r < 0)  { return pixel_black; }
  return pixel_mix
    ( pixel_white, pixel_red
    , (1 - cos(2 * π * sqrt(r))) / 2
    );
}
```

2.4 Binary decomposition

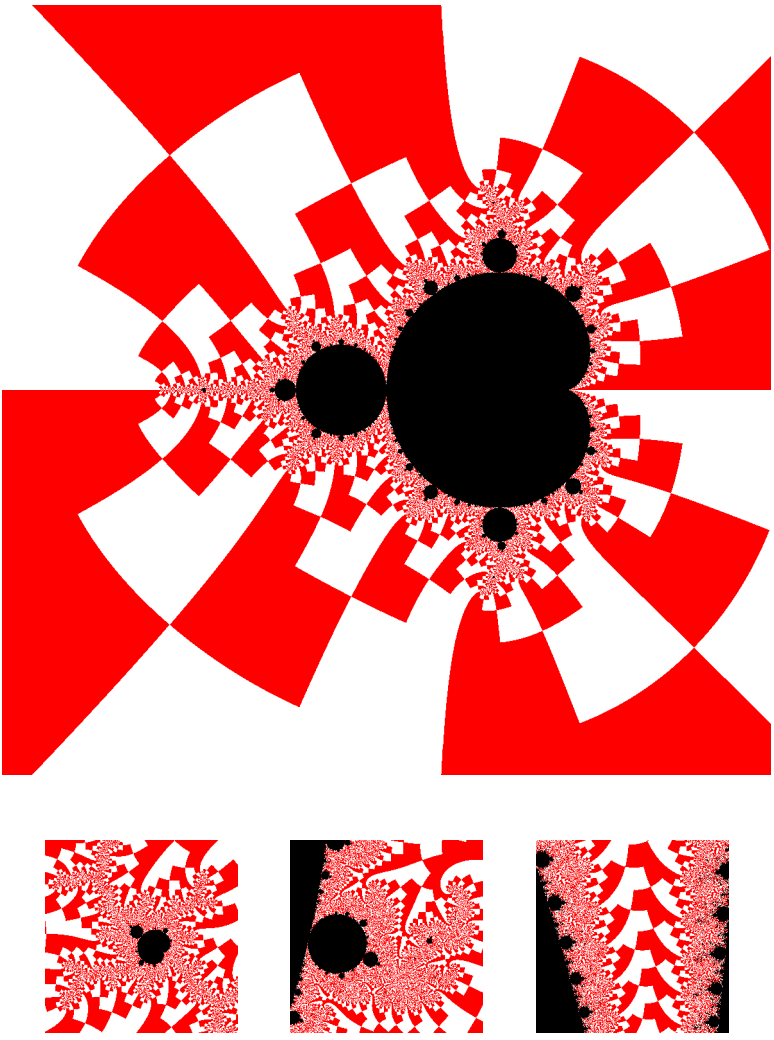

Figure 2.4.1: Binary decomposition images.

FIXME text Discrete grid from final angle (argument of the first iterate to escape). The sign of the angle is the sign of the imaginary part. Changing escape radius changes aspect ratio.

Algorithm 2.4.1 Binary decomposition calculation.

```
ℤ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) { return cimag(z) ≥ 0; }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.4.2 Binary decomposition colouring.

```
pixel_t colour(ℤ r) {
  if (r < 0) { return pixel_black; }
  if (r > 0) { return pixel_red;   }
  return pixel_white;
}
```

2.5 Ternary decomposition

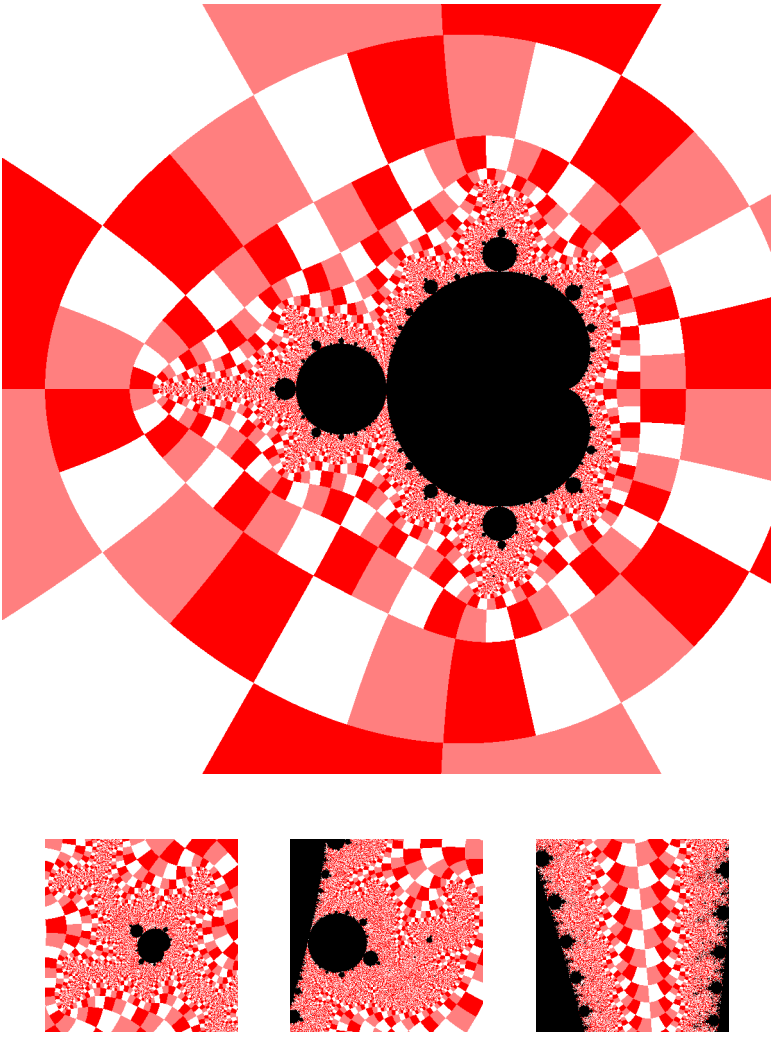

Figure 2.5.1: Ternary decomposition images.

FIXME text Split into 3 colours (instead of 2) by angle. Reverse order of colours, odd and even dwell bands. Visible subcell aspect ratio stays the same when $R_3^3 = R_2^2 = R_1^1$ where R_i is escape radius with i angular subcells per cell. FIXME diagram

Algorithm 2.5.1 Ternary decomposition calculation.

```
ℤ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) {
      ℤ result ← 0;
      ℝ t ← carg(z);
      if (0 ≤ t && t <  2 * π / 3) { result ← 1; }
      if (-2 * π / 3 ≤ t && t < 0) { result ← 2; }
      if (odd(i)) { result ← (3 - result) % 3; }
      return result;
    }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.5.2 Ternary decomposition colouring.

```
pixel_t colour(ℤ r) {
  if (r < 0) { return pixel_black; }
  return pixel_mix
    ( pixel_white, pixel_red
    , r / 2.0
    );
}
```

2.6 Local coordinates

FIXME pictures

FIXME text FIXME code

```
clear(black);
for (c : image) {
  z ← c;
  for (n ← 1; n < maxIters; ++n) {
    if (magnitude(z) > escapeℝadius) {
      plot(c, colour(dwell(n, z, escapeℝadius), arg(z))));
      break;
    }
    z ← z * z + c;
  }
}
```

2.7 Procedural tiling

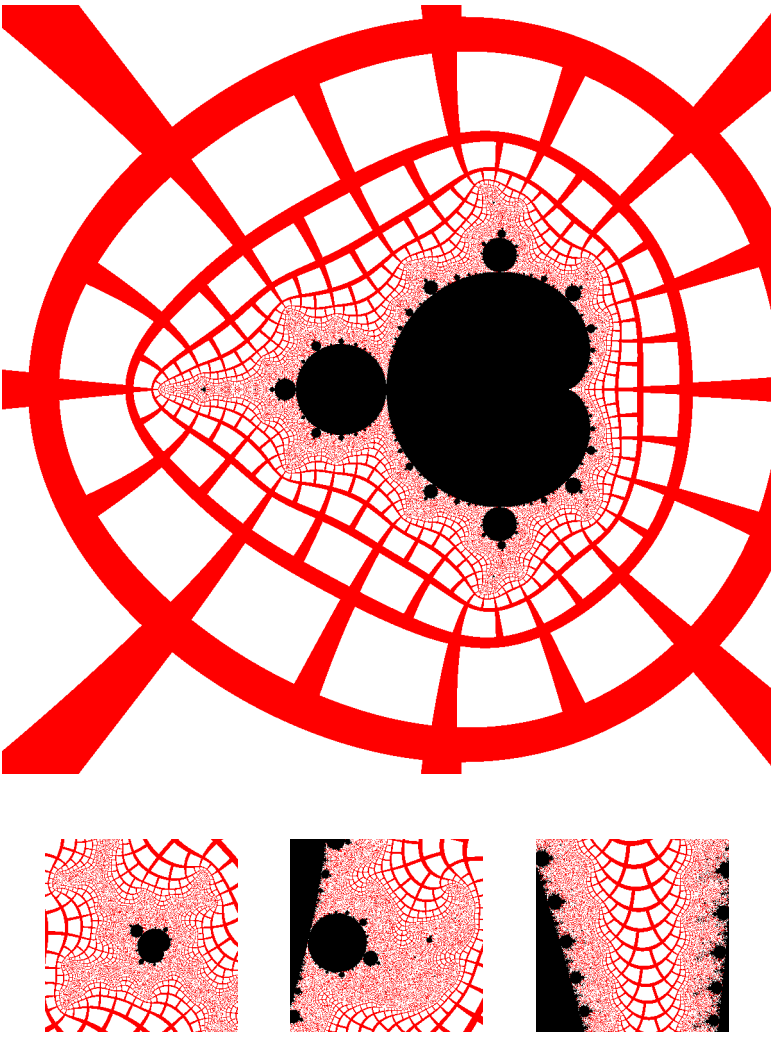

Figure 2.7.1: Procedural tiling images.

FIXME text grid lines give clues to the patterns of external angles

Algorithm 2.7.1 Procedural grid calculation.

```
ℝ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℝ *a // output final angle
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) {
      *a ← carg(z);
      return 1 + i - log2(log(cabs(z))/log(e));
    }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.7.2 Procedural grid colouring.

```
pixel_t colour
( ℤ n  // maximum iteration count
, ℝ *a // final angle
, ℝ r  // continuous dwell
) {
  if (r < 0) { return pixel_black; }
  ℝ u ← fmod(*a / (2 * π) + 1, 1) - 0.5;
  ℝ v ← fmod(r, 1) - 0.5;
  u ← fabs(u) - 0.45 + 0.025 * pow(0.25, v);
  v ← fabs(v) - 0.4;
  return pixel_mix
    ( pixel_white, pixel_red
    , fmax(u, v) > 0
    );
}
```

2.8 Raster tiling

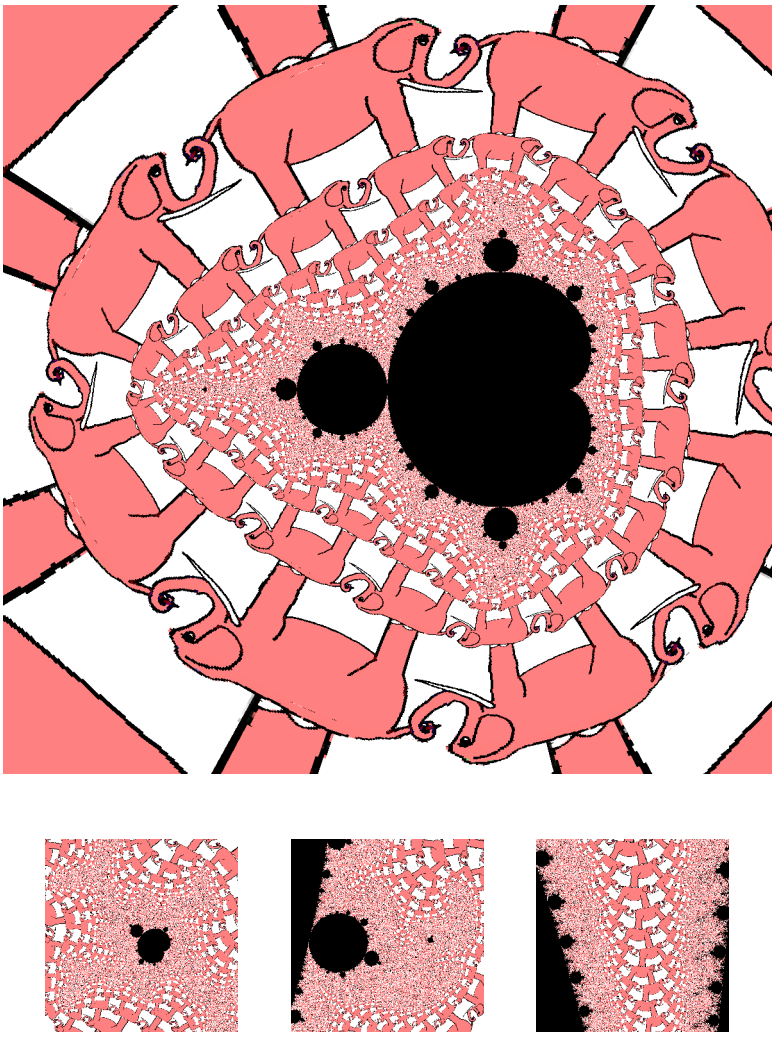

Figure 2.8.1: Raster tiling images.

FIXME text image inlay within each cell, edges should match (hyperbolic?) FIXME diagram linearize continuous dwell within each cell to avoid distortion FIXME code

```
linearDwell(n, z, escapeℝadius) {
  return n - (log(magnitude(z)) / log(escapeℝadius) - 1);
}
```

d = n - log2(log(magnitude(z)) / log(escapeRadius)) = n - log2(k) l = n - (log(magnitude(z)) / log(escapeRadius) - 1) = n - (k - 1)

```
linearizeDwell(d) {
  n ← ceil(d)
  k ← exp2(n - d);
  return n - (k - 1);
}
```

Algorithm 2.8.1 Raster tiling calculation.

```
ℝ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℝ *a // output final angle
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) {
      *a ← carg(z);
      return 1 + i - log2(log(cabs(z))/log(e));
    }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.8.2 Raster tiling colouring.

```
pixel_t colour
( const image_t *tile
, ℝ *a // final angle
, ℝ r  // continuous dwell
) {
  if (r < 0)  { return pixel_black; }
  ℝ u ← fmod(*a / (2 * π) + 1, 1);
  ℝ v ← fmod(r, 1);
  ℤ i ← u * image_width (tile);
  ℤ j ← v * image_height(tile);
  image_clamp(tile, &i, &j);
  return image_peek(tile, i, j);
}
```

2.9 Ternary raster tiling

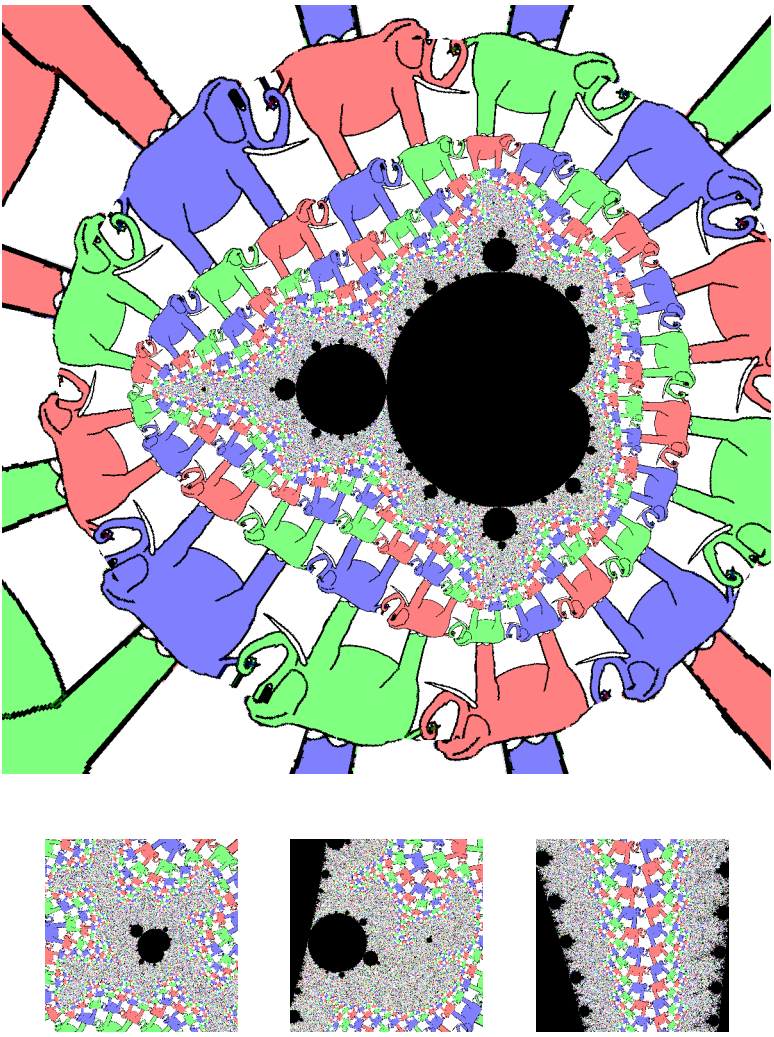

Figure 2.9.1: Ternary raster tiling images.

FIXME text

Algorithm 2.9.1 Ternary raster tiling calculation.

```
ℝ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℝ *a // output final angle
, ℂ c  // parameter
) {
  ℂ z ← c;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) {
      *a ← carg(z);
      return 1 + i - log2(log(cabs(z))/log(e));
    }
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.9.2 Ternary raster tiling colouring.

```
pixel_t colour
( const image_t *tile
, ℝ *a // final angle
, ℝ r  // continuous dwell
) {
  if (r < 0)  { return pixel_black; }
  ℝ u ← fmod(3 * *a / (2 * π) + 3, 3);
  ℤ m ← floor(u);
  u ← fmod(u, 1);
  ℝ v ← fmod(r, 1);
  ℤ n ← floor(r);
  if (odd(n)) { m ← (5 - m) % 3; }
  ℤ i ← u * image_width (tile);
  ℤ j ← v * image_height(tile);
  image_clamp(tile, &i, &j);
  pixel_t p ← image_peek(tile, i, j);
  ℤ c1 ← pixel_r(p);
  ℤ c2 ← pixel_g(p);
  ℤ c3 ← pixel_b(p);
  if (odd(n)) {
    ℤ tmp ← c2; c2 ← c3; c3 ← tmp;
  }
  if (m > 1) {
    ℤ tmp ← c1; c1 ← c2; c2 ← c3; c3 ← tmp;
  }
  if (m > 0) {
    ℤ tmp ← c1; c1 ← c2; c2 ← c3; c3 ← tmp;
  }
  return pixel_rgba8(c1, c2, c3, 255);
}
```

2.10 Distance estimation

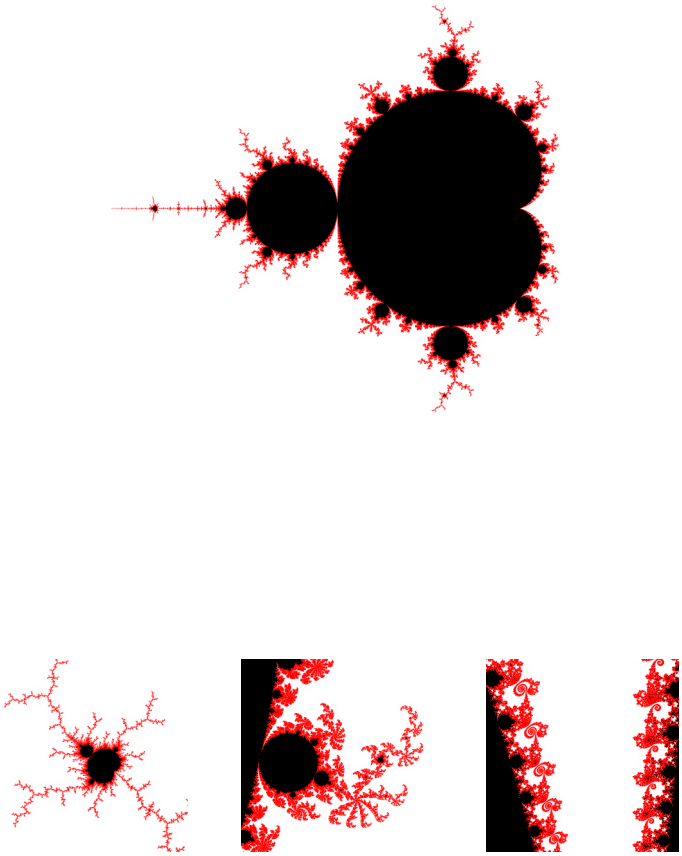

Figure 2.10.1: Distance estimation images.

FIXME text `http://en.wikipedia.org/wiki/Mandelbrot_set#Exterior_distance_estimation` a picture of the Mandelbrot set with the following properties: Every pixel that contains a point of the Mandelbrot set is colored black. Every pixel that is colored black is close to the Mandelbrot set.

Differential calculus. Differentiate with respect to c. Intuition: measures spacing of dwell bands, which get closer and closer as c gets closer to the boundary. Koebe 1/4 theorem: no point in circle(c, de/4) is in the mandelbrot set. Scale distance relative to pixel size. Colouring: use a function that is black at 0 and white at 4. Function should be smooth at the upper end. $\tanh(4) = 0.999329299739067$ Increase escape radius for greater fidelity.

Algorithm 2.10.1 Distance estimation calculation.

```
ℝ calculate
( ℤ n  // maximum iteration count
, ℝ e  // escape radius
, ℂ c  // parameter
) {
  ℂ z ← c;
  ℂ dz ← 1;
  for (ℤ i ← 0; i < n; ++i) {
    if (cabs(z) > e) {
      return 2 * cabs(z) * log(cabs(z)) / cabs(dz);
    }
    dz ← 2 * z * dz + 1;
    z ← csqr(z) + c;
  }
  return -1;
}
```

Algorithm 2.10.2 Distance estimation colouring.

```
pixel_t colour
( ℝ s  // pixel spacing
, ℝ d  // distance estimate
) {
  if (d < 0)  { return pixel_black; }
  return pixel_mix
    ( pixel_red, pixel_white
    , tanh(d / s)
    );
}
```

2.11 Partial periods

FIXME pictures

FIXME text Values of n where |z| reaches a minimum (lower than all previous values). For exterior points, which escape to infinity, the last minimum is well-defined. Conflicting requirements for colouring: periods multiply for children of atoms; periods add along filaments. "Christmas tree": plot mz in interior. FIXME code

```
clear(black);
for (c : image) {
  z ← c;
  mn ← 0;
  mz ← infinity;
  for (n ← 1; n < maxIters; ++n) {
    if (magnitude(z) < mz) {
      mz ← magnitude(z);
      mn ← n;
    }
    if (magnitude(z) > escapeℝadius) {
      plot(c, colour(mn));
      break;
    }
    z ← z * z + c;
  }
}
```