Chap. 1 — Rasterization
Outline

- Raster display technology.
- Basic concepts: pixel, resolution, aspect ratio, dynamic range, image domain, object domain.
- Rasterization and direct illumination.
- Graphics primitives and OpenGL.
- Geometry representations: explicit, parametric and implicit forms.
- Rasterization algorithms for straight line segments, circles and ellipses.
- Rasterization algorithms for triangles and polygons.
- Rasterization versus shading.
Raster display

**Definition:**
- Discrete grid of elements (frame buffer of pixels).
  - Shapes drawn by setting the “right” elements
  - Frame buffer is scanned, one line at a time, to refresh the image (as opposed to vector display)

**Properties:**
- Difficult to draw smooth lines
- Displays only a discrete approximation of any shape
- Refresh of entire frame buffer
# Terminology

## Pixel: Picture Element
- Smallest accessible element in picture.
- Usually rectangular or circular.

## Aspect Ratio:
- Ratio between physical dimensions of pixel (not necessarily 1).

## Dynamic Range:
- Ratio between minimal (not zero!) and maximal light intensity emitted by displayed pixel (black and white, respectively)

## Resolution:
- Number of distinguishable rows and columns on a device measured in:
  - Absolute values (nxm)
  - Relative values (e.g., 300 dpi)
- Usually rectangular or circular.

## Screen space:
- Discrete 2D Cartesian coordinate system of screen pixels.

## Object space:
- Discrete 3D Cartesian coordinate system of the domain or scene or the objects live in.
SCAN CONVERSION
Scan conversion / rasterization
(for direct illumination)

**Definition:**
- The process of converting geometry into pixels.
- Final step in pipeline: *rasterization* (*scan conversion*)
- From screen coordinates (float) to pixels (int)
- Writing pixels into frame buffer.

**Scan conversion:**
- Figuring out which pixels to turn on.

**Shading:**
- Determine a color for each filled pixel.
Graphics primitives

OpenGL Primitive Taxonomy:

- **Point**: POINTS
- **Line**: LINES, LINE_STRIP, LINE_LOOP
- **Triangle**: TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN
- **Polygon**: QUADS, QUAD_STRIP, POLYGON

Other Primitives:

- **Arc**
- **Circle**
- **Ellipsis**
- **Generic Curves**

How is each geometric primitive really drawn on screen?
Geometric representations for lines in IR²

Explicit form:

\[ y = f(x) = mx + b \]

Implicit form:

\[ f(x, y) = Ax + By + C = 0 \]

Parametric form:

\[ x = x(t) = m_0 t + b_0 \]
\[ y = y(t) = m_1 t + b_1 \]
**Scan converting lines**

**Example:**
- Draw from \((x_1, y_1)\) to \((x_2, y_2)\)

**Correctness/quality issues:**
- Gaps exist for line with slope \(m > 1\) (by varying \(x\))

Wrong (steeper line)
- \((x_1, y_1)\) \\
- \((x_2, y_2)\)

Correct
Rasterization rules

Line rasterization rules use a **diamond test area** to determine if a line covers a pixel.
Direct scan conversion

Explicit form:

- $y = mx + b$, where $m = \frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)} = \frac{\Delta y}{\Delta x}$ and $0 \leq m \leq 1$ ($1^{\text{st}}, 4^{\text{th}}, 5^{\text{th}}$ and $8^{\text{th}}$ octants)

- What else?

Key idea:

- Increment $x$ from $x_i$ to $x_f$ and calculate the corresponding value $y = mx + b$

Drawbacks:

- Gaps when $m > 1$. The solution is to increment $y$ instead of $x$ when $m > 1$.

- Floating-point computations: floating-point multiplication and addition for every step in $x$. 

increments in $x$ (fails when $m > 1$)
Chapter 1: Rasterization

**Direct scan conversion (cont’d)**

**Algorithm (m<1):**

- \( m = \frac{y_f - y_i}{x_f - x_i}; \)
- \( b = y_i - m \cdot x_i; \)
- \( x = x_i; y = y_i; \)
- \( \text{DrawPixel}(x, y); \)
- \( \text{for } (x = x_i + 1; x \leq x_f; x++). \)
  - \( y = m \cdot x + b; \)
  - \( \text{DrawPixel}(x, y); \)

 increments in \( x \)
 \((m<1)\)
DDA algorithm (Digital Differential Analyser)

Explicit form:
- \[ y = mx + b, \] where \( m = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \Delta y / \Delta x \) and \( 0 \leq m \leq 1 \) (1st, 4th, 5th and 8th octants)

Key idea:
- Increment \( x \) from \( x_i \) to \( x_f \) and calculate the corresponding value \( y \):
- Current pixel: \( y_i = mx_i + b \)
- Next pixel:
  - \( y_{i+1} = mx_{i+1} + b = m(x_i + 1) + b = y_i + m \)
  - Draw pixel \( (x_{i+1}, y_{i+1}) \), where \( y_{i+1} = \text{ROUND}(y_{i+1}) \)

Drawbacks:
- Gaps when \( m > 1 \). In this case, increment \( y \).
- Floating-point arithmetic: a floating-point addition and a round operation.

Algorithm (\( m < 1 \)):
- \( m = \frac{y_f - y_i}{x_f - x_i} \);
- \( x = x_i; \ y = y_i; \)
- DrawPixel\( (x, y) \);
- for \( (x = x_i + 1; x <= x_f; x++) \):
  - \( y = y + m; \)
  - DrawPixel\( (x, y) \);

Note that the explicit form is not used directly!
DDA algorithm (cont’d)

Algorithm (m>1):

- \( m = \frac{(y_f - y_i)}{(x_f - x_i)}; \)
- \( x = x_i; \ y = y_i; \)
- \( \text{DrawPixel}(x, y); \)
- for \( (y = y_i + 1; \ y \leq y_f; \ y++) \):
  - \( x = x + 1/m; \)
  - \( \text{DrawPixel}(x, y); \)

Why?

Note that the *explicit form* is not used directly!
Bresenham algorithm

Explicit form:

- \( y = mx + b \), where \( m = \frac{\Delta y}{\Delta x} \) and \( 0 \leq m \leq 1 \)

Key idea:

- Increment \( x \) from \( x_i \) to \( x_f \) and calculate the corresponding value \( y \).
- **Current pixel**: \((x_i, y_i)\)
- **Next pixel**: either \((x_{i+1}, y_i)\) or \((x_{i+1}, y_{i+1})\)
  - \( d_1 = y - y_i = mx_{i+1} + b - y_i = m(x_i + 1) + b - y_i \)
  - \( d_2 = y_{i+1} - y = y_i + 1 - y_i - y_i + 1 - m(x_i + 1) + b \)
  - \( \Delta d = d_1 - d_2 = 2m(x_i + 1) - 2y_i + 2b - 1 \)
  - If \( \Delta d > 0 \) choose higher pixel \((x_{i+1}, y_{i+1})\)
  - If \( \Delta d \leq 0 \) choose lower pixel \((x_{i+1}, y_i)\)

Bresenham algorithm (cont’d)

**Integer arithmetic (?):**

- From triangle similarity, we know that
  - $m = \Delta y / \Delta x = d_1 / (x_{i+1} - x_i) = d_1$
  - $d_2 = 1 - d_1 = 1 - m$
- Hence
  - $d_1 - d_2 = 2m - 1$
- To take advantage of integer arithmetic, we use the following decision parameter at the first pixel $(x_i, y_i)$ to choose which is the next pixel:
  - $p_i = \Delta x (d_1 - d_2) = 2 \Delta y - \Delta x$
- But, in general terms, and using $d_1$ and $d_2$ in the previous page, we have:
  - $p_i = \Delta x (d_1 - d_2) = 2 \Delta y x_i - 2 \Delta x y_i + K$, where $K$ is a constant
- Consequently, the decision parameter at $(x_{i+1}, y_{i+1})$ will be:
  - $p_{i+1} = 2 \Delta y x_{i+1} - 2 \Delta x y_{i+1} + K$ or
  - $p_{i+1} = p_i + 2 \Delta y (x_{i+1} - x_i) - 2 \Delta x (y_{i+1} - y_i)$ (note that $x_{i+1} - x_i = 1$)
Bresenham algorithm (cont’d)

Algorithm:

```c
void Bresenham (int xi, int yi, int xf, int yf) {
    int x, y, dx, dy, p;
    x = xi; y = yi;
    p = 2 * dy - dx;
    for(x=xi; x<=xf; x++) {
        DrawPixel (x,y);
        if (p>0) {
            y = y + 1;
            p = p - 2 * dx;
        }
        p = p + 2 * dy;
    }
}
```
Midpoint algorithm


Implicit form:

- \( f(x,y) = Ax + By + C = 0 \)

Key idea:

- Starting from \( y = mx + b \), where \( m = \frac{\Delta y}{\Delta x} \) and \( 0 \leq m \leq 1 \), we have:
  \[
  f(x,y) = \Delta y \cdot x - \Delta x \cdot y + b. \Delta x = 0
  \]
  with \( A = \Delta y \), \( B = -\Delta x \), and \( C = b. \Delta x \)
- Current pixel: \((x_i, y_i)\)
- Next pixel: either \((x_{i+1}, y_i)\) or \((x_{i+1}, y_{i+1})\)
  - Let the decision parameter \( p_i = f(M_p) = f(x_i + 1, y_i + 1/2) \)
  - If \( p_i < 0 \) choose higher pixel \((x_{i+1}, y_i)\) at \( N \)
  - If \( p_i \geq 0 \) choose lower pixel \((x_{i+1}, y_{i+1})\) at \( E \)
Midpoint algorithm (cont’d)

Let us now determine the relation between the function values at consecutive midpoints:

**Key idea (cont’d):**

- \( p_i = f(M_p) = f(x_i + 1, y_i + 1/2) = A(x_i + 1) + B(y_i + 1/2) + C \)
- If \( E \) is chosen:
  - \( p_{i+1} = f(M_E) = f(x_{i+1}, y_i + 1/2) = A(x_{i+1}) + B(y_i + 1/2) + C \)
  - \( = p_i + A = p_i + \Delta y \)
- If \( N \) is chosen:
  - \( p_{i+1} = f(M_N) = f(x_i + 2, y_i + 3/2) = A(x_i + 2) + B(y_i + 3/2) + C \)
  - \( = p_i + A + B = p_i + \Delta y - \Delta x \)

**Integer arithmetic (\( ? \)):**

- Initial decision parameter:
  - \( p_i = f(M_p) = f(x_i + 1, y_i + 1/2) = A(x_i + 1) + B(y_i + 1/2) + C \)
  - \( = f(P) + A + B/2 = f(P) + \Delta y - \Delta x/2 = \Delta y - \Delta x/2 \)

*Multiplying the decision parameter by 2 we realize that we obtain exactly the Bresenham algorithm given before.*
General Bresenham’s algorithm for lines

To generalize lines with arbitrary slopes:

- We need to consider symmetry between various octants and quadrants.

- For $m > 1$, interchange roles of $x$ and $y$, that is step in $y$ direction, and decide whether the $x$ value is above or below the line.

- If $m > 1$, and right endpoint is the first point, both $x$ and $y$ decrease. To ensure uniqueness, independent of direction, always choose upper (or lower) point if the line go through the midpoint.

- Handle special cases without invoking the algorithm: horizontal, vertical and diagonal lines
Scan converting circles

**Explicit form:** \( y = f(x) = \pm \sqrt{R^2 - x^2} \)
- Usually, we draw a quarter circle by incrementing \( x \) from 0 to \( R \) in unit steps and solving for \( +y \) for each step.

**Parametric form:**
\[
\begin{align*}
  x &= R \cos \theta \\
  y &= R \sin \theta
\end{align*}
\]
- Done by stepping the angle from 0 to 90°.
- Solves the gap problem of explicit form.

**Implicit form:** \( f(x,y) = x^2 + y^2 - R^2 = 0 \)
- If \( f(x,y) = 0 \), then it is on the circle;
- If \( f(x,y) > 0 \), then it is outside the circle;
- If \( f(x,y) < 0 \), then it is inside the circle.
Midpoint circle algorithm


**Implicit form:**
- $f(x,y) = x^2 + y^2 - R^2 = 0$

**Key idea:**
- **Current** pixel: $P(x_i, y_i)$
- **Next** pixel: either $(x_{i+1}, y_i)$ or $(x_{i+1}, y_{i-1})$
  - Let the decision parameter $p_i = f(M_p) = f(x_i + 1, y_i - 1/2)$
  - If $p_i < 0$ choose higher pixel $(x_{i+1}, y_i)$ at $E$
  - If $p_i \geq 0$ choose lower pixel $(x_{i+1}, y_{i-1})$ at $S$
Midpoint circle algorithm (cont’d)

Let us now determine the relation between the function values at consecutive midpoints:

Key idea (cont’d):

- \( p_i = f(M_P) = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 + (y_i - 1/2)^2 - R^2 \)
- If \( E \) is chosen:
  - \( p_{i+1} = f(M_E) = f(x_i + 2, y_i - 1/2) = (x_i + 2)^2 + (y_i - 1/2)^2 - R^2 \)
    \( = p_i + (2x_i + 3) \)
- If \( S \) is chosen:
  - \( p_{i+1} = f(M_S) = f(x_i + 2, y_i - 3/2) = (x_i + 2)^2 + (y_i - 3/2)^2 - R^2 \)
    \( = p_i + (2x_i - 2y_i + 5) \)

Integer arithmetic:

- Initial decision parameter at \((x_i, y_i) = (0, R)\):
  - \( p_i = f(M_P) = f(x_i + 1, y_i - 1/2) = (x_i + 1)^2 + (y_i - 1/2)^2 - R^2 \)
    \( = f(P) + 2x_i - y_i + 5/4 = 2x_i - y_i + 5/4 = 5/4 - R \approx 1 - R \)
Midpoint circle algorithm (cont’d)

Algorithm:

```c
void MidPointCircle(int R) {
    int x=0, y=R, d=1-R;
    DrawPixel(x, y);
    while (y>x) {
        if (p< 0) // select E
            p=p + 2 * x + 3;
        else // select S
            { p = p + 2 * (x - y) + 5;
              y = y - 1;
            }
        x = x + 1;
        DrawPixel(x, y);
    }
}
```

The algorithm only calculates the pixels on the 2nd octant. The remaining pixels are found using 8-way-symmetry.
SCAN CONVERSION
of
TRIANGLES/POLYGONS
Scan converting of polygons

**Multiple tasks for scan conversion:**

- Filling polygon (inside/outside)
- Pixel shading (color interpolation)
- Blending (accumulation, not just writing)
- Depth values (z-buffer hidden-surface removal)
- Texture coordinate interpolation (texture mapping)

**Hardware efficiency critical**

**Many algorithms for filling (inside/outside)**

**Much fewer that handle all tasks well**
Chapter 1: Rasterization

Review

Shading:
- Determine a color for each filled pixel.

Scan conversion:
- Figuring out which pixels to turn on.
- Rendering an image of a geometric primitive by setting pixel colors.
- Example:
  - Filling the inside of a triangle.
Triangle scan conversion

Key idea:
- Color all pixels inside triangle.

Inside triangle test:
- A point is inside a triangle if it is in the positive half-space of all three boundary lines.
  - Triangle vertices are ordered counter-clockwise.
  - Point must be on the left side of every boundary line.
- Recall that the implicit equation of a line:
  - On the line: $Ax+By+C=0$
  - On right: $Ax+By+C<0$
  - On left: $Ax+By+C>0$

```c
void ScanCTriangle(Triangle T, Color rgba) {
  for each pixel P(x,y)
    if inside(P, T)
      setPixel(x,y,rgba)
}

Boolean inside(Triangle T, Point P) {
  for each boundary line L of T {
    float dot = L.A*P.x+L.B*P.y+L.C*P.z;
    if dot<0.0 return FALSE;
  }
  return TRUE;
}
```
Summary:

- Raster display technology.
- Basic concepts: pixel, resolution, aspect ratio, dynamic range, image domain, object domain.
- Rasterization and direct illumination.
- Graphics primitives and OpenGL.
- Geometry representations: explicit, parametric and implicit forms.
- Rasterization algorithms for straight line segments, circles and ellipses.
- Rasterization algorithms for triangles and polygons.
- Rasterization versus shading.